MQSeries

GC34-5269-06

Publish/Subscribe User's Guide

Version 1 Release 0.2

MQSeries

GC34-5269-06

Publish/Subscribe User's Guide

Version 1 Release 0.2

Note!

Before using this information and the product it supports, be sure to read the general information under Appendix E, “Notices” on page 201.

Seventh edition (November 2000)

This edition applies to IBM® MQSeries® Publish/Subscribe Version 1.0.2, and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright International Business Machines Corporation 1998, 2000. All rights reserved.**
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

About this book xi
Who this book is for xi
What you need to know to understand this book xi
How to use this book xi
  Appearance of text in this book xi
How to read syntax diagrams xii

Summary of changes xiii
| Changes for this edition (GC34-5269-06) xiii
Changes for the sixth edition (GC34-5269-05) xiii
Changes for the fifth edition (GC34-5269-04) xiii

Part 1. Introduction and system design 1

Chapter 1. Introduction 3
What is publish/subscribe? 3
  What are the components involved? 3
  Example of a single broker configuration 4
  Example of a multiple broker configuration 5
How does it work? 6
How MQSeries Publish/Subscribe relates to MQSeries 7
How MQSeries Publish/Subscribe relates to MQSeries Integrator 8
Installation instructions 10
  Prerequisites 10
  Disk space requirements 11
  The MQSeries Publish/Subscribe package 11
  Installation on AIX 12
  Installation on Compaq Tru64 UNIX 14
  Installation on HP-UX 10 16
  Installation on HP-UX 11 (non-DCE) 17
  Installation on HP-UX 11 (DCE) 18
| Installation on Linux 20
  Installation on Sun Solaris 21
| Installation on Windows NT and Windows 2000 22

Chapter 2. System design 23
Topics 23
  Matching topic strings 23
Streams 24
Broker networks 25
  Passing subscription information between brokers 26
Different types of publication 28
  Local and global publications 28
  State and event information 28
  Retained publications 28
Sample application 30

© Copyright IBM Corp. 1998, 2000

Part 2. Writing applications 33

Chapter 3. Introduction to writing applications 37
Message flows 38
  Simplified message flow 39
Message ordering 42
  Ensuring that messages are retrieved in the correct order 42
Publisher and subscriber identity 43
The message descriptor 44
  Messages sent to the broker 44
  Publications forwarded by the broker 45
Persistence and units of work 46
Limitations 47
  Group messages 47
  Segmented messages 47
  Cluster queues 47
  Data conversion of MQRFH structure 47
Using the Application Messaging Interface 47
  AMI publish/subscribe functions 48

Chapter 4. Writing publisher applications 49
Registering with the broker 49
  Choosing not to register 50
  Options you can specify when registering as a publisher 50
  Broker restart 51
  Changing an application's registration 51
Publishing information 51
  Publication data 51
  Retained publications 52
  Publishing locally and globally 52
Deleting information 53
Deregistering with the broker 53

Chapter 5. Writing subscriber applications 55
Registering as a subscriber 55
  Subscriber queues 56
  Options you can specify when registering as a subscriber 56
  Broker restart 57
  Changing an application's registration 57
Requesting information 58
  Requesting information from the broker 58
  Requesting information from a publisher 58
Deregistering as a subscriber 59

Chapter 6. Format of command messages 61
MQRFH – Rules and formatting header 62
  Fields 62
  Structure definition in C 65
Publish/Subscribe name/value strings 65
Publication data 68

Chapter 7. Publish/Subscribe command messages 71
Delete Publication 72

  Required parameters . . . 72
  Optional parameters . . . 72
  Example . . . 73
  Error codes . . . 73
Deregister Publisher . . . 74
  Required parameters . . . 74
  Optional parameters . . . 74
  Example . . . 75
  Error codes . . . 75
Deregister Subscriber . . . 76
  Required parameters . . . 76
  Optional parameters . . . 76
  Example . . . 77
  Error codes . . . 77
Publish . . . 78
  Required parameters . . . 78
  Optional parameters . . . 78
  Example . . . 82
  Error codes . . . 82
Register Publisher . . . 83
  Required parameters . . . 83
  Optional parameters . . . 83
  Example . . . 84
  Error codes . . . 85
Register Subscriber . . . 86
  Required parameters . . . 86
  Optional parameters . . . 86
  Example . . . 88
  Error codes . . . 88
Request Update . . . 89
  Required parameters . . . 89
  Optional parameters . . . 89
  Example . . . 90
  Error codes . . . 90

Chapter 8. Error handling and response messages . . . 91
Error handling by the broker . . . 91
Response messages . . . 92
  Message descriptor for response messages . . . 92
  Types of error response . . . 93
Broker responses . . . 94
  Standard parameters . . . 94
  Optional parameters . . . 95
  Examples . . . 96
  Error codes applicable to all commands . . . 97
Problem determination . . . 98

Chapter 9. Sample programs . . . 101
Sample application . . . 102
  Running the application . . . 102
  Possible extensions . . . 104
Application Messaging Interface samples . . . 105

Part 3. Managing the broker 107

Chapter 10. Setting up a broker 109
Broker queues 109
  System queues 109
  Other stream queues 110
  Internal queues 111
  Dead-letter queue 111
Other considerations 111
  Access control 111
  Backup 112
Broker configuration stanza 112
  Broker configuration tool 112
  Broker configuration parameters 113

Chapter 11. Controlling the broker 119
Starting a broker 119
  Using triggering to start the broker 119
Stopping a broker 119
Displaying the status of a broker 119
Adding a stream 120
  Creating a stream queue 120
  Informing other brokers about the stream 120
Deleting a stream 120
  Deleting a stream on an isolated broker 121
  Deleting a stream on a broker that is part of a network 121
Adding a broker to a network 121
Deleting a broker from the network 122
  Problems when deleting brokers 122
  Deleting a broker that has a child broker 122
Sequence of commands for adding and deleting brokers 123

Chapter 12. Control commands 125
clrmqbrk (Clear broker’s memory of a neighboring target broker) 126
dltmqbrk (Delete broker) 129
dspmqbrk (Display broker status) 131
endmqbrk (End broker function) 132
migmqbrk (Migrate broker to MQSeries Integrator) 133
strmqbrk (Start broker function) 135

Chapter 13. Message broker exit 137
Publish/subscribe routing exit 138
  Publish/subscribe routing exit parameter structure 139
Writing a publish/subscribe routing exit program 145
  Limitations on MQSeries work done in the routing exit 146
  Security considerations 146
Compiling a publish/subscribe routing exit program 146
Sample routing exit 146

Part 4. System programming 147

Chapter 14. Writing system management applications 149
Format of broker administration messages 150
  Subscription deregistered message 150
  Stream deleted message 150
  Broker deleted message 151
  Stream support messages 151
  Children messages 151
  Parent messages 151
MQCFH - PCF header 152
  Reason codes returned from publish/subscribe messages 153
PCF Command Messages 155
  Delete Publication 155
  Deregister Publisher 155
  Deregister Subscriber 156
  Publish 156
  Register Publisher 157
  Register Subscriber 157
  Request Update 158

Chapter 15. Finding out about other publishers and subscribers 159
Metatopics 160
Subscribing to metatopics 161
  Using wildcards 161
  Example requests 161
Authorized metatopics 162
Finding out about brokers 162
Message format for metatopics 162
  Parameters 163
Sample program for administration information 166
  Operation 166
  Example of metatopic information 167

Part 5. Appendixes 171

Appendix A. Reason codes 173

Appendix B. Error messages 179

Appendix C. Constants 191
String constants 191
  MQPS_* (Publish/Subscribe tag names) 191
  MQPS_* (Command tag values) 193
  MQPS_* (Delete, publication and registration options) 193
  MQRFH_* (Rules and formatting header structure identifier) 194
Integer constants 195
  MQAT_* (Application type for message descriptor) 195
  MQCACF_* (Character parameter identifiers for PCF) 195
  MQCMD_* (Command identifiers for PCF) 195
  MQDELO_* (Delete options) 195
  MQDT_* (Destination type for routing exit) 195

MQIACF_★ (Integer parameter identifiers for PCF) 195
MQPUBO_★ (Publication options) 196
MQREGO_★ (Registration options) 196
MQRFH_★ (Rules and formatting header) 196
MQUA_★ (User-attribute selectors for PCF) 196
Reason codes 197
MQRC_RFH_★ (RFH reason codes) 197
MQRCCF_★ (PCF reason codes) 197

Appendix D. Header files 199

Appendix E. Notices 201
Trademarks 203

Glossary of terms and abbreviations 205

Bibliography 207
MQSeries cross-platform publications 207
MQSeries platform-specific publications 207
MQSeries Integrator publications 208
Softcopy books 208
HTML format 208
Portable Document Format (PDF) 209
BookManager format 209
PostScript format 209
Windows Help format 209
MQSeries information available on the Internet 209

Index 211

Figures

1. Simple publish/subscribe example 4
2. Publish/subscribe example with two brokers 5
3. Communication between publishers, subscribers, and brokers 7
4. Simple broker hierarchy 25
5. Propagation of subscriptions through a broker network 26
6. Multiple subscriptions 27
7. Propagation of publications through a broker network 27
8. The results service application 31
9. Basic flow of messages 38
10. Simplified flow of messages 39
11. Flow of messages in a single-broker system 40
12. Flow of messages in a multi-broker system 40
13. Flow of messages using retained publications 41
14. Flow of messages using publish on request only 41
15. Message descriptor and RFH structure 67
16. Publication data after the RFH structure 68
17. Publishing data within the NameValueString 69
18. User-defined publication data 69

19. Inheriting the CCSID 70
20. Results service running with four match simulators 104
21. Sample Broker stanza for qm.ini 112
22. Sequence of commands to create brokers in a network 123
23. Sequence of commands to delete brokers in a network 123

Tables

1. How to read syntax diagrams xii
2. Fields in MQRFH 62
3. Initial values of fields in MQRFH 65
4. Sample programs 101
5. Fields in MQPXP 139
6. Parameters for publisher and subscriber information messages 163

Tables

About this book

This book describes how to use MQSeries Publish/Subscribe. It is available in portable document format (PDF) only. To view it you need the Adobe Acrobat Reader, Version 3 or later. Click on an entry in the table of contents, or a cross reference within the text, to move directly to that page. Use the Acrobat Reader controls to return to the previous page.

This book is not available in hard copy.

Who this book is for

This book is for experienced users of MQSeries who wish to use MQSeries Publish/Subscribe. Familiarity with these MQSeries books is assumed:

- *MQSeries Application Programming Reference*
- *MQSeries Application Programming Guide*
- *MQSeries Programmable System Management*
- *MQSeries System Administration*

What you need to know to understand this book

To use MQSeries Publish/Subscribe you need to have a good knowledge of MQSeries in general. All the sample programs and header files are in the C programming language.

How to use this book

This book contains the following parts:

- Part 1, "Introduction and system design" on page 1 explains what you can do using MQSeries Publish/Subscribe.
- Part 2, "Writing applications" on page 33 discusses how to write programs to use MQSeries Publish/Subscribe.
- Part 3, "Managing the broker" on page 107 describes how to set up and manage your brokers.
- Part 4, "System programming" on page 147 contains information needed to write system management programs.

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

Appearance of text in this book

This book uses the following type styles:

CompCode	The name of a parameter of a call, a field in a structure, or an attribute of an object
dltmqbrk	A control command or command message
MQRFH	The name of a data type or structure
MQPS_COMMAND	The name of a constant

© Copyright IBM Corp. 1998, 2000

`MQPSCommand Publish`	Examples
`"MQPSTopic"`	A character string

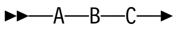
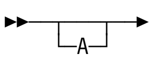
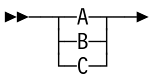
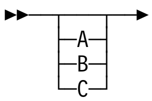
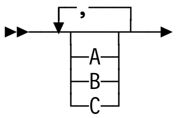
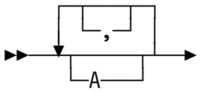
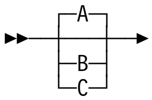
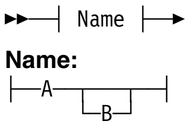
How to read syntax diagrams

This book contains syntax diagrams (sometimes referred to as “railroad” diagrams).

Each syntax diagram begins with a double right arrow and ends with a right and left arrow pair. Lines beginning with a single right arrow are continuation lines. You read a syntax diagram from left to right and from top to bottom, following the direction of the arrows.

Other conventions used in syntax diagrams are:

Table 1. How to read syntax diagrams

Convention	**Meaning**
	You must specify values `A`, `B`, and `C`. Required values are shown on the main line of a syntax diagram.
	You may specify value `A`. Optional values are shown below the main line of a syntax diagram.
	Values `A`, `B`, and `C` are alternatives, one of which you must specify.
	Values `A`, `B`, and `C` are alternatives, one of which you may specify.
	You may specify one or more of the values `A`, `B`, and `C`. Any required separator for multiple or repeated values (in this example, the comma (,)) is shown on the arrow.
	You may specify value `A` multiple times. The separator in this example is optional.
	Values `A`, `B`, and `C` are alternatives, one of which you may specify. If you specify none of the values shown, the default `A` (the value shown above the main line) is used.
	The syntax fragment `Name` is shown separately from the main syntax diagram.
Punctuation and uppercase values	Specify exactly as shown.
Lowercase values (for example, *`name`*)	Supply your own text in place of the *`name`* variable.

Summary of changes

This section describes changes to this edition of *MQSeries Publish/Subscribe User's Guide*. Changes since the previous edition of the book are marked by vertical lines to the left of the changes.

Changes for this edition (GC34-5269-06)

- MQSeries Publish/Subscribe now runs on the Linux and Microsoft Windows 2000 platforms.

Changes for the sixth edition (GC34-5269-05)

- MQSeries Publish/Subscribe now runs on the Compaq Tru64 UNIX® platform.
- A section on the Publish/Subscribe, Integrator relationship ("How MQSeries Publish/Subscribe relates to MQSeries Integrator" on page 8) has been added.
- The section on Publish/Subscribe prerequisites ("Prerequisites" on page 10) has been updated.
- A section on Publish/Subscribe installation ("Installation on Compaq Tru64 UNIX" on page 14) has been added.
- The examples for the **clrmqbrk** command ("clrmqbrk (Clear broker's memory of a neighboring target broker)" on page 126) have been clarified.
- A section on the **migmqbrk** command ("migmqbrk (Migrate broker to MQSeries Integrator)" on page 133) has been added.
- A section on compiling a routing exit ("Compiling a publish/subscribe routing exit program" on page 146) has been added.
- The bibliography ("Bibliography" on page 207) has been updated.
- References to Compaq Tru64 UNIX have been added to the following sections:
  - "How MQSeries Publish/Subscribe relates to MQSeries" on page 7
  - "MQRFH – Rules and formatting header" on page 62
  - Chapter 9, "Sample programs" on page 101
  - "Broker configuration parameters" on page 113
  - "Publish/subscribe routing exit parameter structure" on page 139

Changes for the fifth edition (GC34-5269-04)

- MQSeries Publish/Subscribe now runs on the HP-UX 11 platform (including DCE).
- A new section ("Using the Application Messaging Interface" on page 47) explains how to use the MQSeries Application Messaging Interface (available as a SupportPac) to simplify the writing of publisher and subscriber applications.
- A new section ("Limitations" on page 47) explains some of the limitations of MQSeries Publish/Subscribe.

© Copyright IBM Corp. 1998, 2000

Part 1. Introduction and system design

Chapter 1. Introduction 3
What is publish/subscribe? 3
  What are the components involved? 3
  Example of a single broker configuration 4
  Example of a multiple broker configuration 5
How does it work? 6
How MQSeries Publish/Subscribe relates to MQSeries 7
How MQSeries Publish/Subscribe relates to MQSeries Integrator 8
Installation instructions 10
  Prerequisites 10
  Disk space requirements 11
  The MQSeries Publish/Subscribe package 11
  Installation on AIX 12
    Testing the installation 13
  Installation on Compaq Tru64 UNIX 14
    Testing the installation 15
  Installation on HP-UX 10 16
    Testing the installation 16
  Installation on HP-UX 11 (non-DCE) 17
    Testing the installation 17
  Installation on HP-UX 11 (DCE) 18
    Testing the installation 19
| Installation on Linux 20
| Testing the installation 20
  Installation on Sun Solaris 21
    Testing the installation 21
| Installation on Windows NT and Windows 2000 22
    Testing the installation 22

Chapter 2. System design 23
Topics 23
  Matching topic strings 23
Streams 24
Broker networks 25
  Passing subscription information between brokers 26
Different types of publication 28
  Local and global publications 28
  State and event information 28
  Retained publications 28
Sample application 30

© Copyright IBM Corp. 1998, 2000

Chapter 1. Introduction

This chapter explains what MQSeries Publish/Subscribe is and introduces the concepts and terminology used in this manual. It contains the following sections:

- "What is publish/subscribe?"
- "How does it work?" on page 6
- "How MQSeries Publish/Subscribe relates to MQSeries" on page 7
- "How MQSeries Publish/Subscribe relates to MQSeries Integrator" on page 8
- "Installation instructions" on page 10

What is publish/subscribe?

MQSeries Publish/Subscribe allows you to decouple the provider of information from the consumers of that information.

Before a standard MQSeries application can send some information to another application, it needs to know something about that application. For example, it needs to know the name of the queue to which to send the information, and might also specify a queue manager name.

MQSeries Publish/Subscribe removes the need for your application to know anything about the target application. All it has to do is send information it wants to share to a standard destination managed by MQSeries Publish/Subscribe, and let MQSeries Publish/Subscribe deal with the distribution. Similarly, the target application does not have to know anything about the source of the information it receives.

What are the components involved?

The provider of the information is called a *publisher*. Publishers supply information about a subject, without needing to know anything about the applications that are interested in the information.

The consumer of the information is called a *subscriber*. The subscriber decides what information it is interested in, and then waits to receive that information. Subscribers can receive information from many different publishers, and the information they receive can also be sent to other subscribers.

The information is sent in an MQSeries message, and the subject of the information is identified by a *topic*. The publisher specifies the topic when it publishes the information, and the subscriber specifies the topics on which it wishes to receive publications. The subscriber is only sent information about those topics it subscribes to.

Interactions between publishers and subscribers are all controlled by a *broker*. The broker receives messages from publishers, and subscription requests from subscribers (to a range of topics). The broker's job is to route the published data to the target subscribers.

Related topics can be grouped together to form a *stream*. Publishers can choose to use streams, for example to restrict the range of publications and subscriptions

© Copyright IBM Corp. 1998, 2000

that a broker has to support, or to provide access control. The broker has a default stream that is used for all topics that do not belong to another stream.

The broker uses standard MQSeries facilities to do this, so your applications can use all the features that are available to existing MQSeries applications. This means that you can use persistent messages to get once-only assured delivery, and that your messages can be part of a transactional unit-of-work to ensure that messages are delivered to the subscriber only if they are committed by the publisher.

Example of a single broker configuration

Figure 1 illustrates a basic broker configuration. The example shows the configuration for a news service, where information is available about several topics within a single stream:

- Publisher 1 is publishing information about sports results using a topic of Sport
- Publisher 2 is publishing information about stock prices using a topic of Stock
- Publisher 3 is publishing information about film reviews using a topic of Films, and about television listings using a topic of TV

Three subscribers have registered an interest in different topics, so the broker sends them the information that they are interested in:

- Subscriber 1 receives the sports results and stock prices
- Subscriber 2 receives the film reviews
- Subscriber 3 receives the sports results

None of the subscribers have registered an interest in the television listings, so these are not distributed.

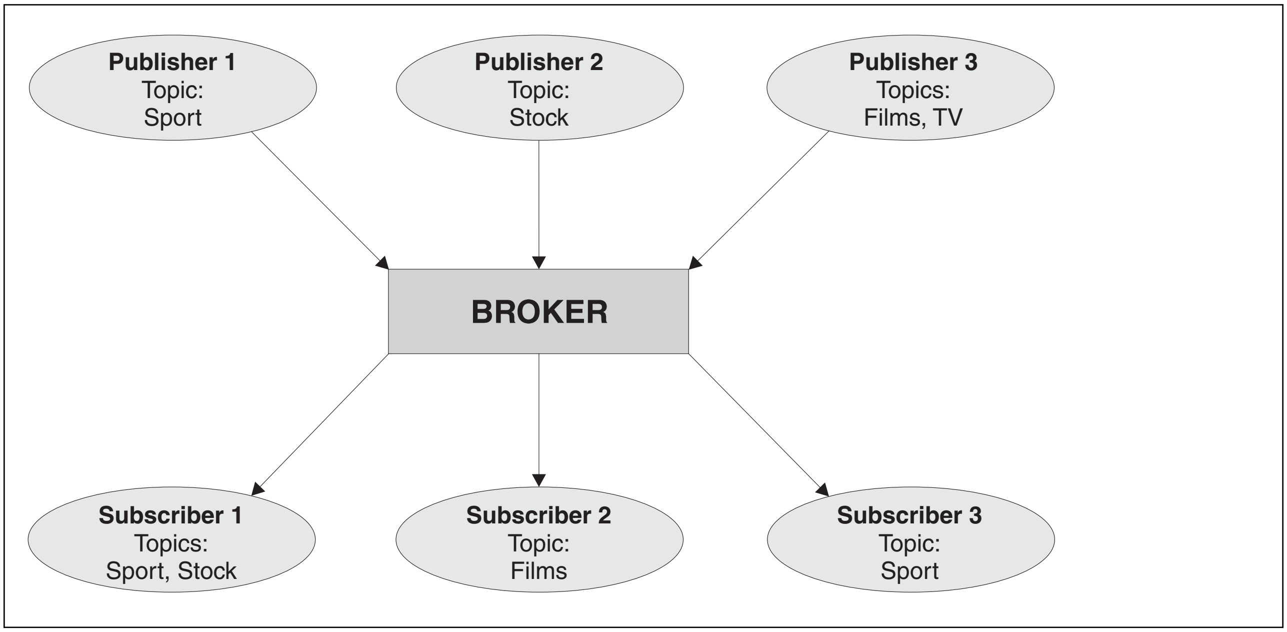


Figure 1. Simple publish/subscribe example. This shows the relationship between publishers, subscribers, and brokers.

Example of a multiple broker configuration

You can have only one broker on each MQSeries queue manager; however, brokers can communicate with other brokers in your MQSeries system, so subscribers can subscribe to one broker and receive messages that were initially published to another broker. This is illustrated in Figure 2.

In this example, a second broker has been added.

- Broker 2 is used by Publisher 4 to publish weather forecast information, using a topic of Weather, and information about traffic conditions on major roads, using a topic of Traffic.
- Subscriber 4 also uses this broker, and subscribes to information about traffic conditions using topic Traffic.
- Subscriber 3 also subscribes to information about weather conditions, even though it uses a different broker from the publisher. This is possible because the brokers are linked to each other.

A publication is propagated to another broker only if a subscription to that topic exists on the other broker.

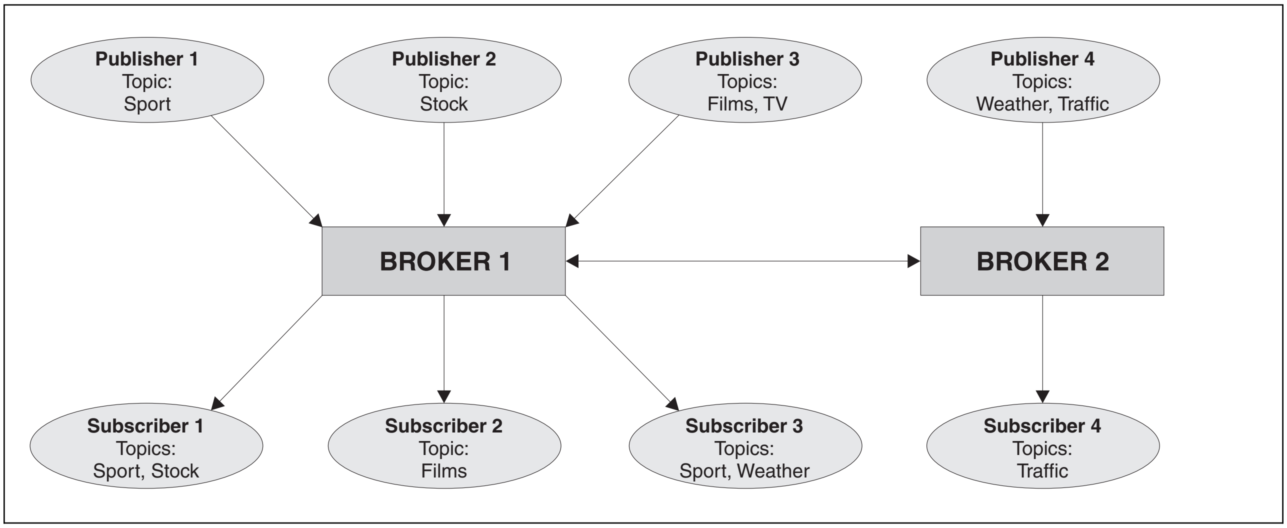

Publisher 1
Topic:
Sport
Publisher 2
Topic:
Stock
Publisher 3
Topics:
Films, TV
Publisher 4
Topics:
Weather, Traffic
BROKER 1
BROKER 2
Subscriber 1
Topics:
Sport, Stock
Subscriber 2
Topic:
Films
Subscriber 3
Topics:
Sport, Weather
Subscriber 4
Topics:
Traffic

Figure 2. Publish/subscribe example with two brokers

How does it work?

Publishers, subscribers, and brokers communicate with each other using *command messages*. These messages are used to do the following things:

Publisher and broker

The following communications take place between publishers and brokers:

1. A publisher can register its intention to publish information about certain topics (this is optional: registration can take place with the first publication, or not at all, as described in “Registering with the broker” on page 49).
2. A publisher sends publication messages to the broker, containing the publication data (or referring to it). The messages can be forwarded directly to the subscribers, or, in the case of retained publications, be held at the broker until requested by a subscriber.
3. A publisher can send a message to the broker requesting that a retained publication held at the broker be deleted.
4. A publisher can deregister with the broker when it has finished sending messages about a certain topic.

These interactions are all described in Chapter 4, “Writing publisher applications” on page 49.

Subscriber and broker

The following communications take place between subscribers and brokers:

1. A subscriber registers with a broker, specifying the topics that it is interested in.
2. The broker sends to the subscriber subsequent publications that match the topics specified. Alternatively, the subscriber can request retained publications held at the broker.
3. The subscriber can deregister with the broker for certain topics when it is no longer interested in them.

These interactions are all described in Chapter 5, “Writing subscriber applications” on page 55.

Broker and broker

The following communications take place between brokers:

1. Brokers can exchange subscription registrations and deregistrations.
2. Brokers can exchange publications, and requests to delete publications.
3. Brokers can exchange information about themselves.

These interactions are illustrated in Figure 3 on page 7.

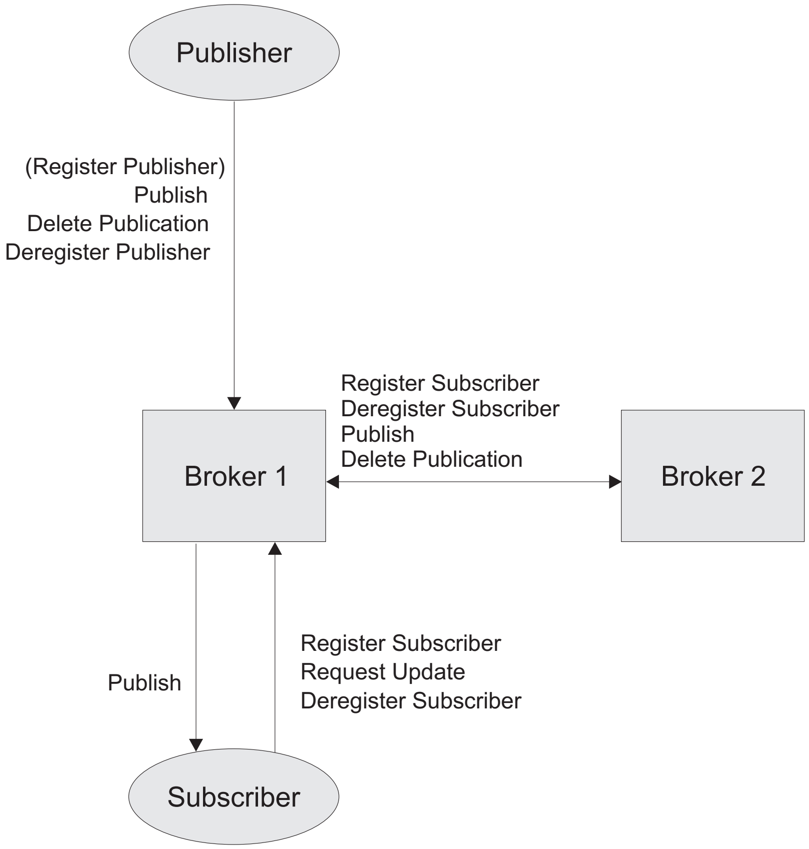

Publisher
(Register Publisher)
Publish
Delete Publication
Deregister Publisher
Register Subscriber
Deregister Subscriber
Publish
Delete Publication
Broker 1
Broker 2
Publish
Register Subscriber
Request Update
Deregister Subscriber
Subscriber

Figure 3. Communication between publishers, subscribers, and brokers

How MQSeries Publish/Subscribe relates to MQSeries

MQSeries Publish/Subscribe is a function of MQSeries. The broker runs on MQSeries for AIX®, Compaq Tru64 UNIX, HP-UX, Sun Solaris, Linux, Microsoft® Windows NT®, and Microsoft Windows 2000. It uses standard MQSeries facilities (but note that it does not support message groups or segmented messages).

You can have one broker on each MQSeries queue manager. The broker uses the same name as the queue manager.

Applications can be written with standard MQSeries programming techniques, using the Message Queue Interface (MQI) or the Application Messaging Interface (AMI).

Publishers and subscribers do not have to be on the same machine as a broker. They can reside anywhere in the network, provided that there is a route from their queue manager to the broker. So, for example, you could have a publisher on OS/390® and a subscriber on OS/2®.

How MQSeries Publish/Subscribe relates to MQSeries Integrator

MQSeries Integrator works with MQSeries messaging, extending its basic connectivity and transport capabilities to provide a powerful *message broker* solution driven by business *rules*. Messages are formed, routed, and transformed according to the rules defined by an easy-to-use graphical user interface.

Diverse applications can exchange information in unlike forms, with brokers handling the processing required for the information to arrive in the right place in the correct format, according to the rules you have defined. The applications have no need to know anything other than their own conventions and requirements.

Applications also have much greater flexibility in selecting which messages they wish to receive, because they can specify a *topic* filter, or a *content-based* filter, or both, to control the messages made available to them.

MQSeries Integrator provides a framework that supports supplied, basic, functions along with *plug-in* enhancements, to enable rapid construction and modification of business processing rules that are applied to messages in the system.

MQSeries Integrator addresses the needs of business and application integration through management of information flow. It provides services based on message brokers to allow you to:

- Route a message to several destinations, using rules that act on the contents of one or more of the fields in the message or message header.
- Transform a message, so that applications using different formats can exchange messages in their own formats.
- Store and retrieve a message, or part of a message, in a database.
- Modify the contents of a message (for example, by adding data extracted from a database).
- Publish a message to make it available to other applications. Other applications can choose to receive publications that relate to specific topics, or that have specific content, or both.
- Create structured topic names, topic-based access control functions, content-based subscriptions, and subscription points.
- Exploit a plug-in interface to develop message processing node types that can be incorporated into the broker framework to complement or replace the supplied nodes, or to incorporate node types developed by Independent Software Vendors (ISVs).
- Enable instrumentation by products such as those developed by Tivoli®, using system management hooks.

The benefits of MQSeries Integrator can be realized both within and beyond your enterprise:

- Your processes and applications can be integrated by providing message and data transformations in a single place, the broker. This helps reduce costs of application upgrades and modifications.
- You can extend your systems to reach your suppliers and customers, by meeting their interface requirements within your brokers. This can help you

improve the quality of your interactions and allow you to respond more quickly to changing or additional requirements.

MQSeries Integrator Version 2.0 extends the capabilities of MQSeries Publish/Subscribe by supporting:

- Enhanced publish/subscribe function through exploitation of structured topic names, *access control*, content-based subscriptions, and *subscription points*.
- Enhancement of message processing through the addition of new *message processing nodes* to complement or replace the supplied nodes.
- Interfaces that allow messages to be enriched with information from a database, or to be stored in a database.

You can upgrade your applications, messages, and brokers to take advantage of the enhancements in MQSeries Integrator Version 2.0. You can also continue to use your existing MQSeries Publish/Subscribe applications and messages unchanged, by tailoring your Version 2.0 system to provide compatible support.

MQSeries Integrator Version 2.0 brokers can interact with MQSeries Publish/Subscribe brokers in a common publish/subscribe environment, to provide coexistence within a single mixed broker network.

Individual MQSeries Publish/Subscribe brokers can also be migrated to become equivalent MQSeries Integrator brokers with support for their existing client applications intact.

Installation instructions

This section discusses the prerequisites, package contents, and installation instructions for MQSeries Publish/Subscribe.

Prerequisites

The MQSeries Publish/Subscribe prerequisites are as follows.

- MQSeries Publish/Subscribe base

  MQSeries Publish/Subscribe requires one of the following:

  - MQSeries for AIX Version 5.2, 5.1, 5.0 with CSD05 (PTF U461602), or later
  - MQSeries for Compaq Tru64 UNIX Version 5.1, or later
  - MQSeries for HP-UX Version 5.2, 5.1, 5.0 with CSD05 (PTF U461603), or later (for HP-UX 10)
  - MQSeries for HP-UX Version 5.2, 5.1, or later (for HP-UX 11)
  - MQSeries for Linux Version 5.2
  - MQSeries for Sun Solaris Version 5.2, 5.1, 5.0 with CSD05 (PTF U461609), or later
  - MQSeries for Windows NT and Windows 2000 Version 5.2, MQSeries for Windows NT Version 5.1, 5.0 with CSD05 (PTF U200095), or later

- MQSeries Publish/Subscribe, MQSeries Integrator Version 2.0 mixed network

  If you run a mixed network of MQSeries Publish/Subscribe and MQSeries Integrator brokers, the Publish/Subscribe brokers must be at these levels:

  - MQSeries for AIX Version 5.2, 5.1 with CSD02 (PTF U467826), 5.0 with CSD07 (PTF U462315), or later
  - MQSeries for Compaq Tru64 UNIX Version 5.1 with APAR IC24624, or later
  - MQSeries for HP-UX Version 5.2, 5.1 with CSD01 (PTF U465344), 5.0 with CSD07 (PTF U462316), or later (for HP-UX 10)
  - MQSeries for HP-UX Version 5.2, 5.1 with CSD01 (PTF U465427), or later (for HP-UX 11)
  - MQSeries for Linux Version 5.2
  - MQSeries for Sun Solaris Version 5.2, 5.1 with CSD01 (PTF U469913), 5.0 with CSD07 (PTF U462317), or later
  - MQSeries for Windows NT and Windows 2000 Version 5.2, MQSeries for Windows NT Version 5.1 with CSD03 (PTF U200113), 5.0 with CSD07 (PTF U200100), or later

- MQSeries Publish/Subscribe and the **migmqbrk** command

  If you wish to run the **migmqbrk** command, MQSeries Publish/Subscribe requires the following:

  - MQSeries for Windows NT and Windows 2000 Version 5.2, MQSeries for Windows NT Version 5.1 with CSD03 (PTF U200113), or later

Disk space requirements

1.0 MB of disk space is required for the MQSeries Publish/Subscribe executable code and samples.

In addition, 1.2 MB is required to hold the Portable Document Format (PDF) file of the *MQSeries Publish/Subscribe User's Guide*.

The MQSeries Publish/Subscribe package

The package consists of the following files:

- Executables for the functions that control the MQSeries Publish/Subscribe broker. They are described in Chapter 12, “Control commands” on page 125.

```
clrmqbrk      (clrmqbrk.exe on Windows NT and Windows 2000)
dltmqbrk      (dltmqbrk.exe on Windows NT and Windows 2000)
dspmqbrk      (dspmqbrk.exe on Windows NT and Windows 2000)
endmqbrk      (endmqbrk.exe on Windows NT and Windows 2000)
migmqbrk      (migmqbrk.exe on Windows NT and Windows 2000)
strmqbrk      (strmqbrk.exe on Windows NT and Windows 2000)
```

- Broker control processes.

```
amqfcxaa      (amqfcxaa.exe on Windows NT and Windows 2000)
amqfcxba      (amqfcxba.exe on Windows NT and Windows 2000)
```

- Broker configuration tool. This is described in “Broker configuration tool” on page 112.

```
cfgmqbrk.exe  (Windows NT and Windows 2000 only)
```

- Sample programs for MQSeries Publish/Subscribe. They are described in Chapter 9, “Sample programs” on page 101.

```
amqsgam       (amqsgam.exe on Windows NT and Windows 2000)
amqspsd       (amqspsd.exe on Windows NT and Windows 2000)
amqsres       (amqsres.exe on Windows NT and Windows 2000)
amqsfmda.tst
amqsgama.c
amqsgama.tst
amqspsra.c
amqsresa.c
amqsresa.tst
amqspsda.c
amqspsda.tst
```

Installation on AIX

1. For MQSeries for AIX Version 5.2
   1. Login as root
   2. Download `ma0c_axmq52.tar.Z` in binary and store in `/tmp`
   3. Execute `uncompress -fv /tmp/ma0c_axmq52.tar.Z`
   4. Execute `tar -xvf /tmp/ma0c_axmq52.tar`
   5. Execute `rm /tmp/ma0c_axmq52.tar`
2. For MQSeries for AIX Version 5.0 and 5.1
   1. Login as root
   2. Download `ma0c_ax.tar.Z` in binary and store in `/tmp`
   3. Execute `uncompress -fv /tmp/ma0c_ax.tar.Z`
   4. Execute `tar -xvf /tmp/ma0c_ax.tar`
   5. Execute `rm /tmp/ma0c_ax.tar`

The following files will be created:

```
/usr/lpp/mqm/bin/amqfcxaa
/usr/lpp/mqm/bin/amqfcxba
/usr/lpp/mqm/bin/clrmqbrk
/usr/lpp/mqm/bin/dltmqbrk
/usr/lpp/mqm/bin/dspmqbrk
/usr/lpp/mqm/bin/endmqbrk
/usr/lpp/mqm/bin/migmqbrk
/usr/lpp/mqm/bin/strmqbrk
/usr/bin/amqfcxaa -> /usr/lpp/mqm/bin/amqfcxaa
/usr/bin/amqfcxba -> /usr/lpp/mqm/bin/amqfcxba
/usr/bin/clrmqbrk -> /usr/lpp/mqm/bin/clrmqbrk
/usr/bin/dltmqbrk -> /usr/lpp/mqm/bin/dltmqbrk
/usr/bin/dspmqbrk -> /usr/lpp/mqm/bin/dspmqbrk
/usr/bin/endmqbrk -> /usr/lpp/mqm/bin/endmqbrk
/usr/bin/migmqbrk -> /usr/lpp/mqm/bin/migmqbrk
/usr/bin/strmqbrk -> /usr/lpp/mqm/bin/strmqbrk
/usr/lpp/mqm/samp/bin/amqsgam
/usr/lpp/mqm/samp/bin/amqspsd
/usr/lpp/mqm/samp/bin/amqsres
/usr/lpp/mqm/samp/pubsub/amqsfmda.tst
/usr/lpp/mqm/samp/pubsub/amqsgama.c
/usr/lpp/mqm/samp/pubsub/amqsgama.tst
/usr/lpp/mqm/samp/pubsub/amqspsra.c
/usr/lpp/mqm/samp/pubsub/amqsresa.c
/usr/lpp/mqm/samp/pubsub/amqsresa.tst
/usr/lpp/mqm/samp/pubsub/admin/amqspsda.c
/usr/lpp/mqm/samp/pubsub/admin/amqspsda.tst
/tmp/li
/tmp/ipla
```

To uninstall MQSeries Publish/Subscribe on AIX platforms, remove the files created above.

Testing the installation

After installation, it is recommended that you run the sample application (see “Running the application” on page 102) to verify that the MQSeries Publish/Subscribe system has been installed correctly.

Installation on Compaq Tru64 UNIX

1. For Compaq Tru64 UNIX Version 4.0
    a. Login as root
    b. Download `ma0c_tru64v4.tar.Z` in binary and store in `/tmp`
    c. Execute `uncompress -fv /tmp/ma0c_tru64v4.tar.Z`
    d. Execute `tar -xvf /tmp/ma0c_tru64v4.tar`
    e. Execute `rm /tmp/ma0c_tru64v4.tar`
2. For Compaq Tru64 UNIX Version 5.0
    a. Login as root
    b. Download `ma0c_tru64v5.tar.Z` in binary and store in `/tmp`
    c. Execute `uncompress -fv /tmp/ma0c_tru64v5.tar.Z`
    d. Execute `tar -xvf /tmp/ma0c_tru64v5.tar`
    e. Execute `rm /tmp/ma0c_tru64v5.tar`

The following files will be created:

```
/opt/mqm/bin/amqfcxaa
/opt/mqm/bin/amqfcxba
/opt/mqm/bin/clrmqbrk
/opt/mqm/bin/dltmqbrk
/opt/mqm/bin/dspmqbrk
/opt/mqm/bin/endmqbrk
/opt/mqm/bin/strmqbrk
/usr/bin/amqfcxaa -> /opt/mqm/bin/amqfcxaa
/usr/bin/amqfcxba -> /opt/mqm/bin/amqfcxba
/usr/bin/clrmqbrk -> /opt/mqm/bin/clrmqbrk
/usr/bin/dltmqbrk -> /opt/mqm/bin/dltmqbrk
/usr/bin/dspmqbrk -> /opt/mqm/bin/dspmqbrk
/usr/bin/endmqbrk -> /opt/mqm/bin/endmqbrk
/usr/bin/strmqbrk -> /opt/mqm/bin/strmqbrk
/opt/mqm/samp/bin/amqsgam
/opt/mqm/samp/bin/amqspsd
/opt/mqm/samp/bin/amqsres
/opt/mqm/samp/pubsub/amqsfmda.tst
/opt/mqm/samp/pubsub/amqsgama.c
/opt/mqm/samp/pubsub/amqsgama.tst
/opt/mqm/samp/pubsub/amqspsra.c
/opt/mqm/samp/pubsub/amqsresa.c
/opt/mqm/samp/pubsub/amqsresa.tst
/opt/mqm/samp/pubsub/admin/amqspsda.c
/opt/mqm/samp/pubsub/admin/amqspsda.tst
/tmp/li
/tmp/ipla
```

To uninstall MQSeries Publish/Subscribe on Compaq Tru64 UNIX platforms, remove the files created above.

Testing the installation

After installation, it is recommended that you run the sample application (see "Running the application" on page 102) to verify that the MQSeries Publish/Subscribe system has been installed correctly.

Installation on HP-UX 10

1. Login as root
2. Download `ma0c_hp.tar.Z` in binary and store in `/tmp`
3. Execute `uncompress -fv /tmp/ma0c_hp.tar.Z`
4. Execute `tar -xvf /tmp/ma0c_hp.tar`
5. Execute `rm /tmp/ma0c_hp.tar`

The following files will be created:

```
/opt/mqm/bin/amqfcxaa
/opt/mqm/bin/amqfcxba
/opt/mqm/bin/clrmqbrk
/opt/mqm/bin/dltmqbrk
/opt/mqm/bin/dspmqbrk
/opt/mqm/bin/endmqbrk
/opt/mqm/bin/migmqbrk
/opt/mqm/bin/strmqbrk
/usr/bin/amqfcxaa -> /opt/mqm/bin/amqfcxaa
/usr/bin/amqfcxba -> /opt/mqm/bin/amqfcxba
/usr/bin/clrmqbrk -> /opt/mqm/bin/clrmqbrk
/usr/bin/dltmqbrk -> /opt/mqm/bin/dltmqbrk
/usr/bin/dspmqbrk -> /opt/mqm/bin/dspmqbrk
/usr/bin/endmqbrk -> /opt/mqm/bin/endmqbrk
/usr/bin/migmqbrk -> /opt/mqm/bin/migmqbrk
/usr/bin/strmqbrk -> /opt/mqm/bin/strmqbrk
/opt/mqm/samp/bin/amqsgam
/opt/mqm/samp/bin/amqspsd
/opt/mqm/samp/bin/amqsres
/opt/mqm/samp/pubsub/amqsfmda.tst
/opt/mqm/samp/pubsub/amqsgama.c
/opt/mqm/samp/pubsub/amqsgama.tst
/opt/mqm/samp/pubsub/amqspsra.c
/opt/mqm/samp/pubsub/amqsresa.c
/opt/mqm/samp/pubsub/amqsresa.tst
/opt/mqm/samp/pubsub/admin/amqspsda.c
/opt/mqm/samp/pubsub/admin/amqspsda.tst
/tmp/li
/tmp/ipla
```

To uninstall MQSeries Publish/Subscribe on HP-UX platforms, remove the files created above.

Testing the installation

After installation, it is recommended that you run the sample application (see “Running the application” on page 102) to verify that the MQSeries Publish/Subscribe system has been installed correctly.

Installation on HP-UX 11 (non-DCE)

1. Login as root
2. Download `ma0c_hp11.tar.Z` in binary and store in `/tmp`
3. Execute `uncompress -fv /tmp/ma0c_hp11.tar.Z`
4. Execute `tar -xvf /tmp/ma0c_hp11.tar`
5. Execute `rm /tmp/ma0c_hp11.tar`

The following files will be created:

```
/opt/mqm/bin/amqfcxaa
/opt/mqm/bin/amqfcxba
/opt/mqm/bin/clrmqbrk
/opt/mqm/bin/dltmqbrk
/opt/mqm/bin/dspmqbrk
/opt/mqm/bin/endmqbrk
/opt/mqm/bin/migmqbrk
/opt/mqm/bin/strmqbrk
/usr/bin/amqfcxaa -> /opt/mqm/bin/amqfcxaa
/usr/bin/amqfcxba -> /opt/mqm/bin/amqfcxba
/usr/bin/clrmqbrk -> /opt/mqm/bin/clrmqbrk
/usr/bin/dltmqbrk -> /opt/mqm/bin/dltmqbrk
/usr/bin/dspmqbrk -> /opt/mqm/bin/dspmqbrk
/usr/bin/endmqbrk -> /opt/mqm/bin/endmqbrk
/usr/bin/migmqbrk -> /opt/mqm/bin/migmqbrk
/usr/bin/strmqbrk -> /opt/mqm/bin/strmqbrk
/opt/mqm/samp/bin/amqsgam
/opt/mqm/samp/bin/amqspsd
/opt/mqm/samp/bin/amqsres
/opt/mqm/samp/pubsub/amqsfmda.tst
/opt/mqm/samp/pubsub/amqsgama.c
/opt/mqm/samp/pubsub/amqsgama.tst
/opt/mqm/samp/pubsub/amqspsra.c
/opt/mqm/samp/pubsub/amqsresa.c
/opt/mqm/samp/pubsub/amqsresa.tst
/opt/mqm/samp/pubsub/admin/amqspsda.c
/opt/mqm/samp/pubsub/admin/amqspsda.tst
/tmp/li
/tmp/ipla
```

To uninstall MQSeries Publish/Subscribe on HP-UX platforms, remove the files created above.

Testing the installation

After installation, it is recommended that you run the sample application (see “Running the application” on page 102) to verify that the MQSeries Publish/Subscribe system has been installed correctly.

Installation on HP-UX 11 (DCE)

1. Login as root
2. Download `ma0c_hp11dce.tar.Z` in binary and store in `/tmp`
3. Execute `uncompress -fv /tmp/ma0c_hp11dce.tar.Z`
4. Execute `tar -xvf /tmp/ma0c_hp11dce.tar`
5. Execute `rm /tmp/ma0c_hp11dce.tar`

The following files will be created:

```
/opt/mqm/bin/amqfcxaa_d
/opt/mqm/bin/amqfcxba_d
/opt/mqm/bin/clrmqbrk_d
/opt/mqm/bin/dltmqbrk_d
/opt/mqm/bin/dspmqbrk_d
/opt/mqm/bin/endmqbrk_d
/opt/mqm/bin/migmqbrk_d
/opt/mqm/bin/strmqbrk_d
/opt/mqm/bin/amqfcxaa -> /opt/mqm/bin/amqfcxaa_d
/opt/mqm/bin/amqfcxba -> /opt/mqm/bin/amqfcxba_d
/opt/mqm/bin/clrmqbrk -> /opt/mqm/bin/clrmqbrk_d
/opt/mqm/bin/dltmqbrk -> /opt/mqm/bin/dltmqbrk_d
/opt/mqm/bin/dspmqbrk -> /opt/mqm/bin/dspmqbrk_d
/opt/mqm/bin/endmqbrk -> /opt/mqm/bin/endmqbrk_d
/opt/mqm/bin/migmqbrk -> /opt/mqm/bin/migmqbrk_d
/opt/mqm/bin/strmqbrk -> /opt/mqm/bin/strmqbrk_d
/usr/bin/amqfcxaa -> /opt/mqm/bin/amqfcxaa_d
/usr/bin/amqfcxba -> /opt/mqm/bin/amqfcxba_d
/usr/bin/clrmqbrk -> /opt/mqm/bin/clrmqbrk_d
/usr/bin/dltmqbrk -> /opt/mqm/bin/dltmqbrk_d
/usr/bin/dspmqbrk -> /opt/mqm/bin/dspmqbrk_d
/usr/bin/endmqbrk -> /opt/mqm/bin/endmqbrk_d
/usr/bin/migmqbrk -> /opt/mqm/bin/migmqbrk_d
/usr/bin/strmqbrk -> /opt/mqm/bin/strmqbrk_d
/opt/mqm/samp/bin/amqsgam_d
/opt/mqm/samp/bin/amqspsd_d
/opt/mqm/samp/bin/amqsres_d
/opt/mqm/samp/bin/amqsgam -> /opt/mqm/samp/bin/amqsgam_d
/opt/mqm/samp/bin/amqspsd -> /opt/mqm/samp/bin/amqspsd_d
/opt/mqm/samp/bin/amqsres -> /opt/mqm/samp/bin/amqsres_d
/opt/mqm/samp/pubsub/amqsfmda.tst
/opt/mqm/samp/pubsub/amqsgama.c
/opt/mqm/samp/pubsub/amqsgama.tst
/opt/mqm/samp/pubsub/amqspsra.c
/opt/mqm/samp/pubsub/amqsresa.c
/opt/mqm/samp/pubsub/amqsresa.tst
/opt/mqm/samp/pubsub/admin/amqspsda.c
/opt/mqm/samp/pubsub/admin/amqspsda.tst
/tmp/li
/tmp/ipla
```

To uninstall MQSeries Publish/Subscribe on HP-UX platforms, remove the files created above.

Testing the installation

After installation, it is recommended that you run the sample application (see “Running the application” on page 102) to verify that the MQSeries Publish/Subscribe system has been installed correctly.

Installation on Linux

1. Login as root
2. Download `ma0c_linux.tar.gz` in binary and store in `/tmp`
3. Execute `gunzip /tmp/ma0c_linux.tar.gz`
4. Execute `tar -xvPf /tmp/ma0c_linux.tar`
5. Execute `rm /tmp/ma0c_linux.tar`

The following files will be created:

```
/opt/mqm/bin/amqfcxaa
/opt/mqm/bin/amqfcxba
/opt/mqm/bin/clrmqbrk
/opt/mqm/bin/dltmqbrk
/opt/mqm/bin/dspmqbrk
/opt/mqm/bin/endmqbrk
/opt/mqm/bin/migmqbrk
/opt/mqm/bin/strmqbrk
/usr/bin/amqfcxaa -> /opt/mqm/bin/amqfcxaa
/usr/bin/amqfcxba -> /opt/mqm/bin/amqfcxba
/usr/bin/clrmqbrk -> /opt/mqm/bin/clrmqbrk
/usr/bin/dltmqbrk -> /opt/mqm/bin/dltmqbrk
/usr/bin/dspmqbrk -> /opt/mqm/bin/dspmqbrk
/usr/bin/endmqbrk -> /opt/mqm/bin/endmqbrk
/usr/bin/migmqbrk -> /opt/mqm/bin/migmqbrk
/usr/bin/strmqbrk -> /opt/mqm/bin/strmqbrk
/opt/mqm/samp/bin/amqsgam
/opt/mqm/samp/bin/amqspsd
/opt/mqm/samp/bin/amqsres
/opt/mqm/samp/pubsub/amqsfmda.tst
/opt/mqm/samp/pubsub/amqsgama.c
/opt/mqm/samp/pubsub/amqsgama.tst
/opt/mqm/samp/pubsub/amqspsra.c
/opt/mqm/samp/pubsub/amqsresa.c
/opt/mqm/samp/pubsub/amqsresa.tst
/opt/mqm/samp/pubsub/admin/amqspsda.c
/opt/mqm/samp/pubsub/admin/amqspsda.tst
/tmp/li
/tmp/ipla
```

To uninstall MQSeries Publish/Subscribe on Linux platforms, remove the files created above.

Note: See the `GroupId` parameter in “Broker configuration parameters” on page 113 as the group nobody does not exist by default on some Linux installations.

Testing the installation

After installation, it is recommended that you run the sample application (see “Running the application” on page 102) to verify that the MQSeries Publish/Subscribe system has been installed correctly.

Installation on Sun Solaris

1. Login as root
2. Download `ma0c_sol.tar.Z` in binary and store in `/tmp`
3. Execute `uncompress -fv /tmp/ma0c_sol.tar.Z`
4. Execute `tar -xvf /tmp/ma0c_sol.tar`
5. Execute `rm /tmp/ma0c_sol.tar`

The following files will be created:

```
/opt/mqm/bin/amqfcxaa
/opt/mqm/bin/amqfcxba
/opt/mqm/bin/clrmqbrk
/opt/mqm/bin/dltmqbrk
/opt/mqm/bin/dspmqbrk
/opt/mqm/bin/endmqbrk
/opt/mqm/bin/migmqbrk
/opt/mqm/bin/strmqbrk
/usr/bin/amqfcxaa -> /opt/mqm/bin/amqfcxaa
/usr/bin/amqfcxba -> /opt/mqm/bin/amqfcxba
/usr/bin/clrmqbrk -> /opt/mqm/bin/clrmqbrk
/usr/bin/dltmqbrk -> /opt/mqm/bin/dltmqbrk
/usr/bin/dspmqbrk -> /opt/mqm/bin/dspmqbrk
/usr/bin/endmqbrk -> /opt/mqm/bin/endmqbrk
/usr/bin/migmqbrk -> /opt/mqm/bin/migmqbrk
/usr/bin/strmqbrk -> /opt/mqm/bin/strmqbrk
/opt/mqm/samp/bin/amqsgam
/opt/mqm/samp/bin/amqspsd
/opt/mqm/samp/bin/amqsres
/opt/mqm/samp/pubsub/amqsfmda.tst
/opt/mqm/samp/pubsub/amqsgama.c
/opt/mqm/samp/pubsub/amqsgama.tst
/opt/mqm/samp/pubsub/amqspsra.c
/opt/mqm/samp/pubsub/amqsresa.c
/opt/mqm/samp/pubsub/amqsresa.tst
/opt/mqm/samp/pubsub/admin/amqspsda.c
/opt/mqm/samp/pubsub/admin/amqspsda.tst
/tmp/li
/tmp/ipla
```

To uninstall MQSeries Publish/Subscribe on Sun Solaris platforms, remove the files created above.

Testing the installation

After installation, it is recommended that you run the sample application (see “Running the application” on page 102) to verify that the MQSeries Publish/Subscribe system has been installed correctly.

Installation on Windows NT and Windows 2000

1. For MQSeries for Windows NT and Windows 2000 Version 5.2
   1. Create an empty directory called `tmp` and make it current.
   2. Download file `ma0c_ntmq52.zip` to this directory.
   3. Uncompress using Info-ZIP's UnZip program.
   4. Enter `install <drive:directory>` for the drive and directory where MQSeries has been installed. For example `install C:\mqm` or `install "C:\Program Files\mqm"`
2. For MQSeries for Windows NT Version 5.0 and 5.1
   1. Create an empty directory called `tmp` and make it current.
   2. Download file `ma0c_nt.zip` to this directory.
   3. Uncompress using Info-ZIP's UnZip program.
   4. Enter `install <drive:directory>` for the drive and directory where MQSeries has been installed. For example `install C:\mqm` or `install "C:\Program Files\mqm"`

Alternatively, you can copy the files manually. Enter the following if you installed MQSeries to `C:\mqm`, otherwise use the drive and directory where MQSeries was installed.

```
copy bin\amqfcxaa.exe           c:\mqm\bin
copy bin\amqfcxba.exe           c:\mqm\bin
copy bin\cfgmqbrk.exe           c:\mqm\bin
copy bin\clrmqbrk.exe           c:\mqm\bin
copy bin\dltmqbrk.exe           c:\mqm\bin
copy bin\dspmqbrk.exe           c:\mqm\bin
copy bin\endmqbrk.exe           c:\mqm\bin
copy bin\migmqbrk.exe           c:\mqm\bin
copy bin\strmqbrk.exe           c:\mqm\bin
copy bin\samples\amqsgam.exe    c:\mqm\tools\c\samples\bin
copy bin\samples\amqspsd.exe    c:\mqm\tools\c\samples\bin
copy bin\samples\amqsres.exe    c:\mqm\tools\c\samples\bin
copy samples\amqsfmda.tst       c:\mqm\tools\c\samples\pubsub
copy samples\amqsgama.c         c:\mqm\tools\c\samples\pubsub
copy samples\amqsgama.tst       c:\mqm\tools\c\samples\pubsub
copy samples\amqspsra.c         c:\mqm\tools\c\samples\pubsub
copy samples\amqsresa.c         c:\mqm\tools\c\samples\pubsub
copy samples\amqsresa.tst       c:\mqm\tools\c\samples\pubsub
copy samples\admin\amqspsda.c   c:\mqm\tools\c\samples\pubsub\admin
copy samples\admin\amqspsda.tst c:\mqm\tools\c\samples\pubsub\admin
```

To uninstall MQSeries Publish/Subscribe on Windows NT and Windows 2000 platforms, delete the files that were created above.

Testing the installation

After installation, it is recommended that you run the sample application (see “Running the application” on page 102) to verify that the MQSeries Publish/Subscribe system has been installed correctly.

Chapter 2. System design

This chapter discusses the things that you need to consider when you design your MQSeries Publish/Subscribe system.

- “Topics”
- “Streams” on page 24
- “Broker networks” on page 25
- “Different types of publication” on page 28

The “Sample application” on page 30 illustrates how these features can be used in practice.

Topics

A topic identifies what a publication is about. It consists of a character string.

You can use any characters within the single-byte character set for which the machine is configured in a topic string. However, a topic string might need to be translated to a different character representation, so you are recommended to use only those characters that are available in the configured character set of all relevant machines. Topic strings are case sensitive, and a blank character has no special meaning. A null character terminates the string and is not considered to be part of it.

Subscribers can specify a topic or range of topics, using wildcards, for the information that they want.

Matching topic strings

The wildcard characters recognized by MQSeries Publish/Subscribe are:

*****	Zero or more characters
?	One character

In the example shown in Figure 1 on page 4, the high-level topic of 'Sport' might be divided into separate topics covering different sports, such as:

```
Sport/Soccer
Sport/Golf
Sport/Tennis
```

These might be divided further, to separate different types of information about each sport, such as:

```
Sport/Soccer/Fixtures
Sport/Soccer/Results
Sport/Soccer/Reports
```

Note: MQSeries Publish/Subscribe does not recognize that the ‘/’ character is being used in a special way. However, it is recommended that the ‘/’ character is used as a separator to ensure compatibility with other MQSeries business integration functions.

The following topic strings could be used in subscriptions to retrieve particular sets of information:

© Copyright IBM Corp. 1998, 2000

`*`	All information on Sport, Stock, Films and TV.
`Sport/*`	All information on Soccer, Golf and Tennis.
`Sport/Soccer/*`	All information on Soccer (Fixtures, Results and Reports).
`Sport/*/Results`	All Results for Soccer, Golf and Tennis.

Note that wildcards do not span streams (see “Streams”).

The percent character ‘%’ is used as an escape character, to allow these characters to be used in a topic string. For example, the string ‘ABC%*D’ represents the actual topic `ABC*D`. If the string `ABC%*D` is specified in a **Publish** message (where wildcard characters are not allowed), the string could be matched by a subscriber specifying the string `ABC?D`.

To use a % character in a topic string, specify two percent characters ‘%%’. A percent character in a string must always be followed by a ‘*’, a ‘?’, or another ‘%’ character.

If wildcard characters are not allowed in a message, a ‘*’ or ‘?’ character can only be present if it is immediately preceded by a ‘%’ character so that the ‘*’ or ‘?’ character loses its wildcard semantics. Therefore, `ABC%*D` is a valid topic string in a **Publish** message but `ABC*D` is not.

Streams

Streams provide a way of separating the flow of information for different topics. A stream is implemented as a set of queues, one at each broker that supports the stream. Each of these queues has the same name (the name of the stream). The default stream set up between all the brokers in a network is called SYSTEM.BROKER.DEFAULT.STREAM.

Streams can be created by an application or by the administrator. Stream names are case sensitive, and stream queues must be local queues (not alias queues). Stream names beginning with the characters ‘SYSTEM.BROKER.’ are reserved for MQSeries use. For more information see “Broker queues” on page 109.

A broker has a separate thread for each stream that it supports. If multiple streams are used the broker can process publications arriving at different stream queues in parallel. Other advantages of using streams are as follows:

- To provide a high level grouping of topics.

  Streams act as high-level qualifiers for topics. For instance, in the example shown in Figure 1 on page 4, a separate stream might be set up for Sport. In this case, to get the soccer results you need to subscribe to the `Soccer/Results` topic specifying the ‘Sport’ stream. The other topics (Stock, Films, TV) will remain on the default stream, unless other streams are set up for them.

  Note that wildcard characters are not used for stream names, and that wildcards do not span streams. For example, a subscriber to topic ‘*’ on the ‘Sport’ stream will not receive publications published on other streams.

- To restrict the range of publications and subscriptions that a broker has to deal with.

A given stream can be restricted to a subtree of a hierarchy or the stream can be split into separate hierarchies that are not connected (see "Broker networks" on page 25). For example, if broker 1 in Figure 4 on page 25 does not support a stream supported by its children, brokers 2 and 3 will each form the root of a separate hierarchy for that stream, and no subscriptions or publications will flow between the two hierarchies.

- To provide access control.

  A broker has a stream queue for each stream that it supports. Normal MQSeries access control techniques can be used to control whether a particular application is authorized to put messages onto this queue (publish to this stream), or to browse messages from the queue (subscribe to it). Although a subscribing application does not get messages from the broker's queue directly, the broker checks the subscriber's authorization to subscribe to the broker's queue when it registers the subscription. This authorization check takes place at the broker to which the application publishes or subscribes, not at other brokers to which the publication or subscription might be propagated.

  The administrator can change publishers' and subscribers' stream queue authorizations dynamically (using normal MQSeries queue management facilities), although the changes might not take effect until the broker is restarted.

- To define a certain quality of service for broker-to-broker communication of publications.

  You can send information associated with one stream along different channels from those used for another stream. For example, a non-urgent stream might have its associated channels active only during the night.

- To allow different queue attributes (such as maximum message length) to be assigned for publications on different streams.

Broker networks

You can link brokers together to form a network of brokers. A broker network must be arranged as a hierarchy. The broker at the top of the hierarchy is called the *root broker*. The root broker can have one or more *child brokers*, and is known as the *parent broker* to these brokers. The child brokers can also have child brokers, and so on, as illustrated in Figure 4.

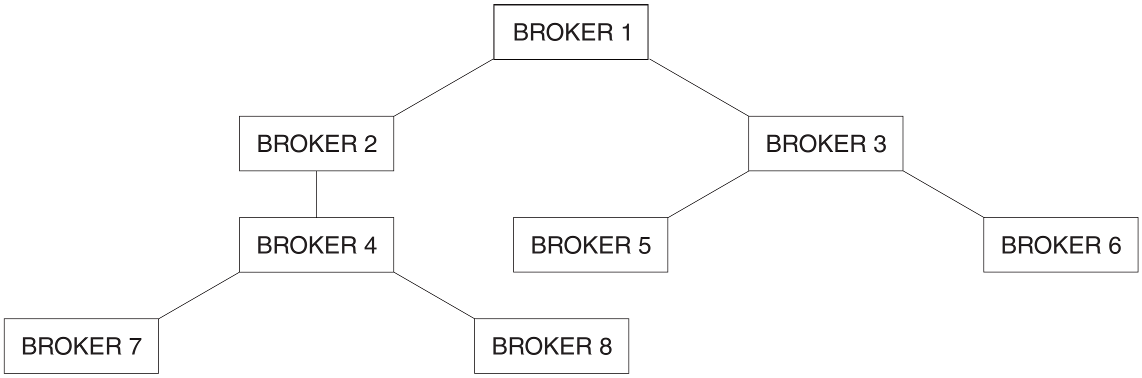


Figure 4. Simple broker hierarchy. Broker 1 is the root broker and brokers 2 and 3 are its children. Broker 4 is the child of broker 2 and the parent of brokers 7 and 8.

Using a hierarchy reduces the number of channels that need to be defined because each broker does not need to be connected to every other broker. Both publication and subscription traffic take a hierarchic route to their destinations.

Each broker maintains administrative information about its parent broker. When a broker first starts, it communicates with its parent. In this way, each broker knows the identities of its immediate children as well as its parent. These are known as the broker's *neighbors*.

The hierarchy should be defined from the root down and, if it is necessary to delete brokers, this should be done from the bottom up. This usually means that to change the root broker you have to delete the whole network and start again (in exceptional cases you can use the **clrmqbrk** command described on 126).

Passing subscription information between brokers

Subscriptions flow to all nodes in the network that support the stream in question. This is shown in Figure 5.

A broker consolidates all of the subscriptions that are registered with it, whether from applications directly or from other brokers. In turn, it registers subscriptions for these topics with its neighbors, unless a subscription already exists. This is shown in Figure 6 on page 27.

When an application publishes information, the receiving broker forwards it (possibly through one or more other brokers) to any applications that have valid subscriptions for it, including applications registered at other brokers supporting this stream (for global publications). This is shown in Figure 7 on page 27.

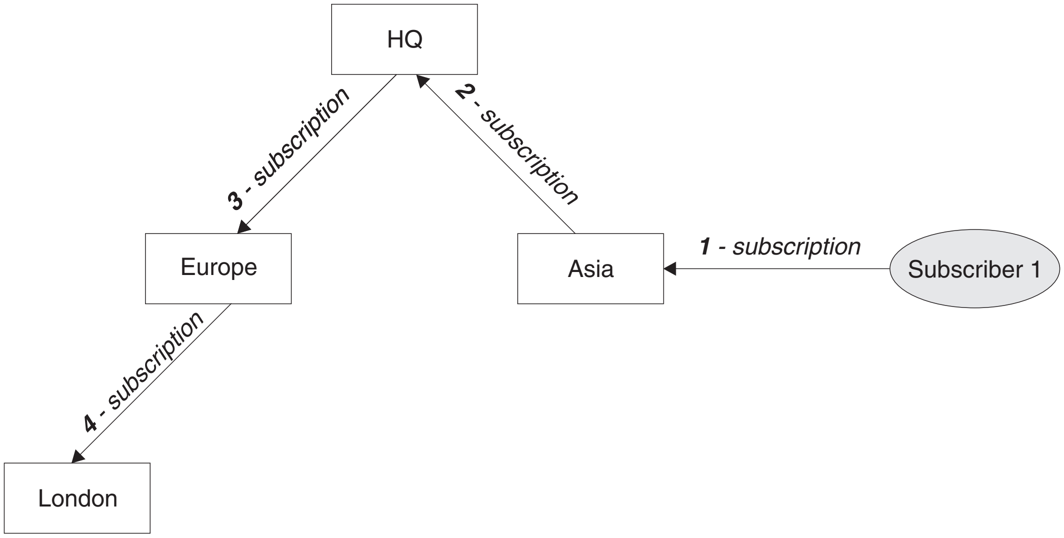


Figure 5. Propagation of subscriptions through a broker network. Subscriber 1 registers a subscription for a particular topic and stream on the Asia broker (1). The subscription for this topic is forwarded to all other brokers in the network that support the stream (2,3,4).

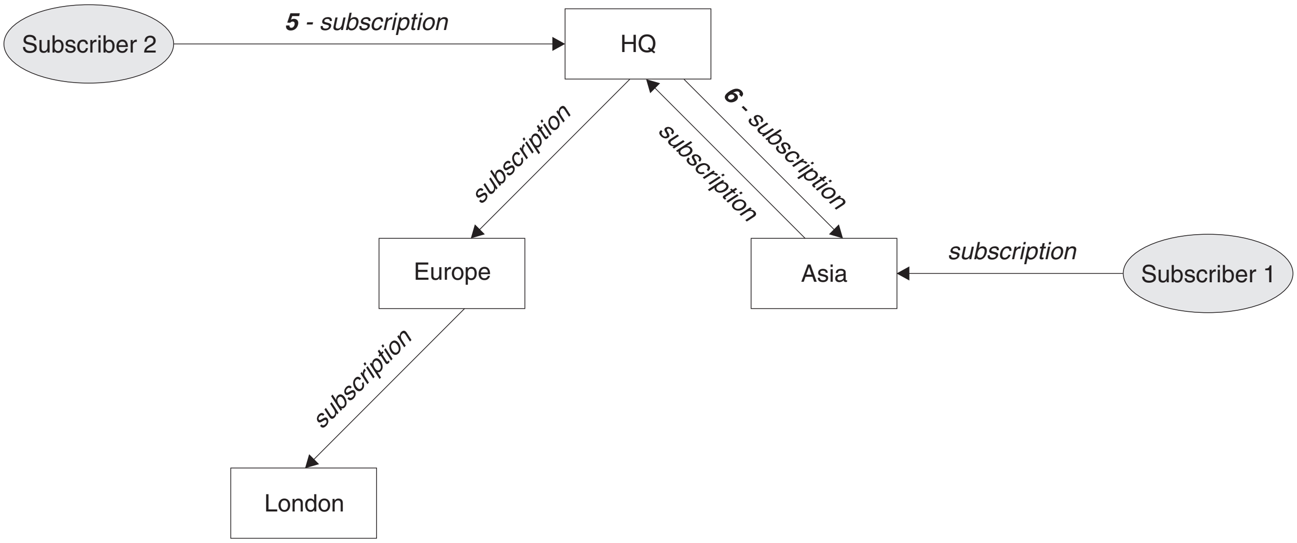

Subscriber 2
5 - subscription
HQ
subscription
6 - subscription
subscription
Europe
Asia
subscription
Subscriber 1
subscription
London

Figure 6. Multiple subscriptions. Subscriber 2 registers a subscription, with the same topic and stream as in Figure 5, on the HQ broker (5). The subscription for this topic is forwarded to the Asia broker, so that it is aware that subscriptions exist elsewhere on the network (6). The subscription does not have to be forwarded to the Europe broker, because a subscription for this topic has already been registered (step 3 in Figure 5).

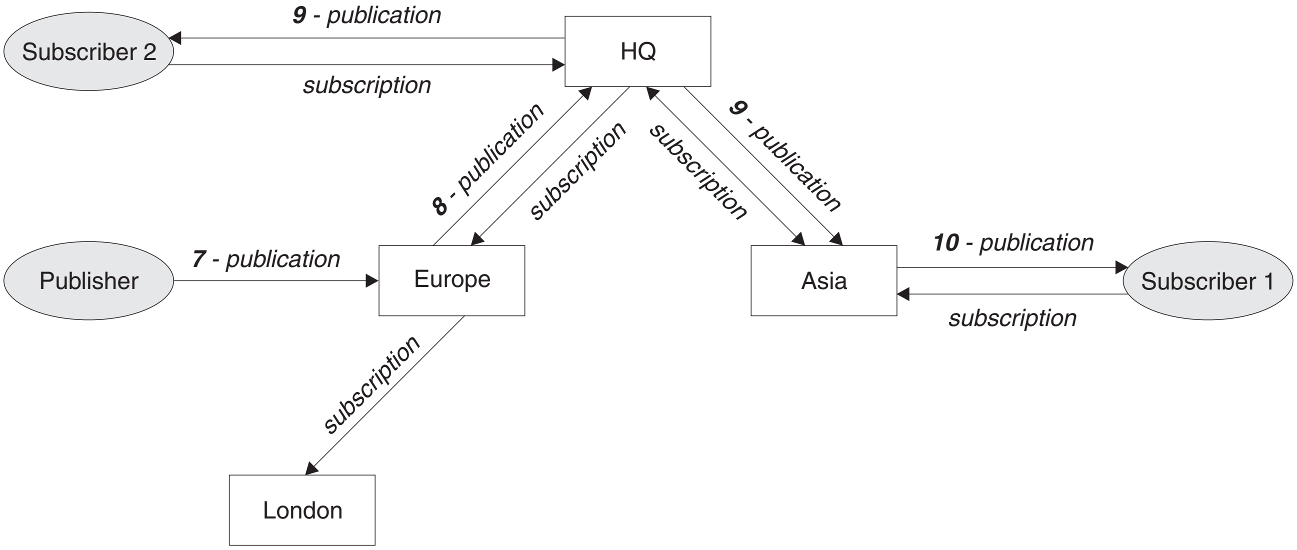

9 - publication
Subscriber 2
HQ
subscription
8 - publication
subscription
subscription
9 - publication
Publisher
7 - publication
Europe
Asia
10 - publication
Subscriber 1
subscription
subscription
London

Figure 7. Propagation of publications through a broker network. A publisher sends a publication, on the same topic and stream as in Figure 6, to the Europe broker (7). A subscription for this topic exists from HQ to Europe, so the publication is forwarded to the HQ broker (8). However, no subscription exists from London to Europe (only from Europe to London), so the publication is not forwarded to the London broker. The HQ broker sends the publication directly to subscriber 2 and to the Asia broker (9), from where it is forwarded to subscriber 1 (10).

When a broker sends any publish or subscribe message to another broker, it sets its own user ID in the message, and uses its own authority to put the message. This means that the broker must have the authority to put messages onto other brokers' queues (unless the channel is set up to put incoming messages with the message channel agent's authority). This also means that all authorization checks are performed at the publisher's or subscriber's local broker.

For more information about brokers, see Part 3, “Managing the broker” on page 107.

Different types of publication

The broker can handle publications it receives in different ways, depending on the type of information contained in the publication.

Local and global publications

A publication that is made available through all the brokers on a network is called a *global publication*. If required, access to publications can be restricted to subscribers that use the same broker as the publisher. This is called a *local publication*, and it can be specified when the publisher registers with the broker, or each time it sends publications to the broker. Local publications are not forwarded to other brokers.

Subscribers can specify whether they want to receive local publications or global publications (but not both) when they register with the broker. Subscribers subscribing to global publications do not receive local publications, even if they are published at the same broker that their subscription was registered at.

State and event information

Publications can be categorized as follows:

State publications
State publications contain information about the current *state* of something, such as the price of stock or the current score in a soccer match. When something happens (for example, the stock price changes or the soccer score changes), the previous state information is no longer required as it is superseded by the new information.

A subscriber will want to receive the current version of the state information when it starts up, and be sent new information whenever the state changes.

Event publications
Event publications contain information about individual *events* that occur, such as a trade in some stock or the scoring of a particular goal. Each of these events is independent of other events.

A subscriber will want to receive information about events as they happen.

Retained publications

By default, a broker deletes a publication when it has sent that publication to all the interested subscribers. This type of processing is suitable for event information, but is not always suitable for state information. A publisher can specify that it wants the broker to keep a copy of a publication, which is then called a *retained publication*. The copy can be sent to subsequent subscribers who register an interest in the topic. This means that new subscribers don't have to wait for information to be published again before they receive it. For example, a subscriber registering a subscription to a stock price would receive the current price straightaway, without waiting for the stock price to change (and hence be re-published).

The broker retains only one publication for each topic, so the old publication is deleted when a new one arrives. It is recommended that you do not have more than one publisher sending retained publications on the same topic.

Subscribers can specify that they do *not* want to receive retained publications, and existing subscribers can ask for duplicate copies of retained publications to be sent to them.

When deciding whether to used retained publications, you need to consider several factors.

- Will your publications contain state information or event information?

  Event publications do not usually have to be retained. For state information, if all the subscriptions to a topic are in place *before* any publications are made on that topic (and no new ones are expected), there is no need to retain publications because they will be delivered to all subscribers when they are published.

  Another reason why publications might not need to be retained is if they are very frequent (for example, every second), because a new subscriber (or a subscriber recovering from a failure) will receive the current state almost immediately after it subscribes.

- How will the subscriber application recover from a failure?

  If the publisher does not use retained publications, the subscriber application might need to store its current state locally. If the publisher does use retained publications, the subscriber application can use the **Request Update** message to refresh its state information after a restart.

  Note that the broker will continue to send publications to a registered subscriber even if that subscriber is not running. This could lead to a build-up of messages on the subscriber queue, which can be avoided if the subscriber registers with the ‘Publish on Request Only’ option. The subscriber must then refresh its state periodically using the **Request Update** command message. Note that in this case the subscriber will *not* receive any non-retained publications.

- What are the performance implications of retaining publications?

  The broker needs to write retained publications to disk during the **Publish** request, which will reduce throughput. If the publications are very large, a considerable amount of queue space (and hence disk space) will be needed to store the retained publication of each topic. In a multi-broker environment, retained publications will also be stored by all other brokers in the network that have a matching subscription.

Sample application

The sample application (see Chapter 9, “Sample programs” on page 101) simulates a results gathering service that reports the latest score in a sports event such as a soccer match. It receives information from one or more instances of a soccer match simulator that scores goals at random for the two teams. This is illustrated in Figure 8 on page 31.

The match simulator does not keep track of the score. It merely indicates when a match starts or finishes, and when a goal is scored. These events are published to three different topics on the SAMPLE.BROKER.RESULTS.STREAM stream. (The sample program sets up its own stream to avoid any possible conflict with customer applications on the default stream).

- When a match starts, the names of the teams are published on the `Sport/Soccer/Event/MatchStarted` topic.
- When a goal is scored, the name of the team scoring the goal is published on the `Sport/Soccer/Event/ScoreUpdate` topic.
- When a match ends, the names of the teams are published on the `Sport/Soccer/Event/MatchEnded` topic.

The publications on these topics are *not* retained, as they contain event information and not state information.

The results service subscribes to the topic `Sport/Soccer/Event/*` to receive publications from any matches that are in progress. It keeps track of the current score in each match, and whenever there is a change it publishes the score as a *retained* publication on the topic `Sport/Soccer/State/LatestScore/Team1 Team2`, where Team1 and Team2 are the names of the teams in the match.

A subscriber wanting to receive all the latest scores could register a wildcard subscription to topic `Sport/Soccer/State/LatestScore/*` . If it was interested in one particular team only, it could register a different wildcard subscription to topic `Sport/Soccer/State/LatestScore/*MyTeam*` .

Note that the results service must be started before the match simulators, otherwise it might miss some events and hence not be able to ascertain the current state in each match. This is usually the case with event publications, in which subscriptions are static and need to be in place before publications arrive.

If it stops while matches are still in progress the results service can find out the state of play when it restarts. This is done by subscribing to its own retained publications using the the `Sport/Soccer/State/LatestScore/*` topic, with the ‘Publish on Request Only’ option. A **Request Update** command is then issued to receive any retained publications which contain latest scores. (In fact, this is done using a different *`CorrelId`* as explained in “Publisher and subscriber identity” on page 43.)

These publications enable the results service to reconstruct its state as it was when it stopped. It can then process all events that occurred while it was stopped by processing the subscription queue for the `Sport/Soccer/Events/*` topic. Since the subscription will still be registered (no **Deregister Subscriber** message has been sent) it will include any event publications that arrived while the results service was inactive.

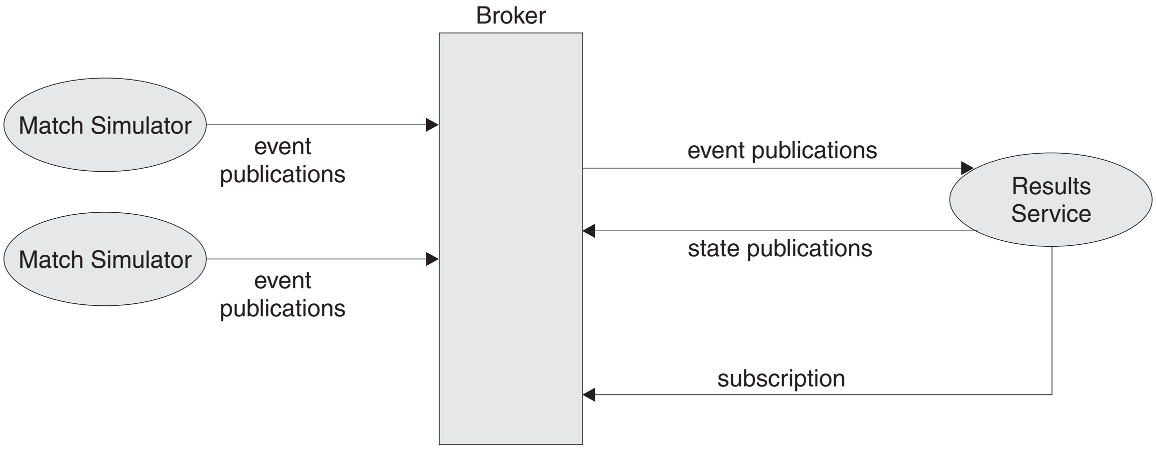

Broker
Match Simulator
event publications
Match Simulator
event publications
event publications
Results Service
state publications
subscription

Figure 8. The results service application. The match simulators publish events when a match starts or finishes, or a goal is scored. The results service subscribes to these events, and publishes the latest scores as state publications.

This sample program illustrates the following aspects of a Publish/Subscribe application:

- The use of streams other than the default stream.
- Event publications (not retained).
- State publications (retained).
- Wildcard matching of topic strings.
- Multiple publishers on the same topics (event publications only).
- The need to subscribe to a topic *before* it is published on (event publications).
- A subscriber continuing to be sent publications when that subscriber (not its subscription) is interrupted.
- The use of retained publications to recover state after a subscriber failure.

Further details of the messages sent between the publisher, subscriber and broker, and the results service sample program, are given in Part 2, “Writing applications” on page 33.

Part 2. Writing applications

Chapter 3. Introduction to writing applications 37
Message flows 38
  Simplified message flow 39
Message ordering 42
  Ensuring that messages are retrieved in the correct order 42
Publisher and subscriber identity 43
The message descriptor 44
  Messages sent to the broker 44
  Publications forwarded by the broker 45
Persistence and units of work 46
Limitations 47
  Group messages 47
  Segmented messages 47
  Cluster queues 47
  Data conversion of MQRFH structure 47
Using the Application Messaging Interface 47
  AMI publish/subscribe functions 48
    Publish command 48
    Register Subscriber command 48
    Deregister Subscriber command 48
    Receive a publication 48

Chapter 4. Writing publisher applications 49
Registering with the broker 49
  Choosing not to register 50
  Options you can specify when registering as a publisher 50
    Queue name 50
    Selecting a stream 50
    Publisher identity 50
    Registration scope 51
    Registration expiry 51
  Broker restart 51
  Changing an application’s registration 51
Publishing information 51
  Publication data 51
    Including data in the message 51
    Referring to data in the message 52
  Retained publications 52
    Expiry of retained publications 52
  Publishing locally and globally 52
Deleting information 53
Deregistering with the broker 53

Chapter 5. Writing subscriber applications 55
Registering as a subscriber 55
  Subscriber queues 56
  Options you can specify when registering as a subscriber 56
    Queue name 56
    Selecting a stream 56
    Subscriber identity 57

© Copyright IBM Corp. 1998, 2000

    Subscription scope . . . 57
    Subscription expiry . . . 57
  Broker restart . . . 57
  Changing an application’s registration . . . 57
Requesting information . . . 58
  Requesting information from the broker . . . 58
  Requesting information from a publisher . . . 58
Deregistering as a subscriber . . . 59

Chapter 6. Format of command messages . . . 61
MQRFH – Rules and formatting header . . . 62
  Fields . . . 62
  Structure definition in C . . . 65
Publish/Subscribe name/value strings . . . 65
    Options using string constants . . . 66
    Options using integer constants . . . 67
    Sending a command message with the RFH structure . . . 67
Publication data . . . 68
    Double-byte character sets . . . 68

Chapter 7. Publish/Subscribe command messages . . . 71
Delete Publication . . . 72
  Required parameters . . . 72
  Optional parameters . . . 72
  Example . . . 73
  Error codes . . . 73
Deregister Publisher . . . 74
  Required parameters . . . 74
  Optional parameters . . . 74
  Example . . . 75
  Error codes . . . 75
Deregister Subscriber . . . 76
  Required parameters . . . 76
  Optional parameters . . . 76
  Example . . . 77
  Error codes . . . 77
Publish . . . 78
  Required parameters . . . 78
  Optional parameters . . . 78
  Example . . . 82
  Error codes . . . 82
Register Publisher . . . 83
  Required parameters . . . 83
  Optional parameters . . . 83
  Example . . . 84
  Error codes . . . 85
Register Subscriber . . . 86
  Required parameters . . . 86
  Optional parameters . . . 86
  Example . . . 88
  Error codes . . . 88
Request Update . . . 89
  Required parameters . . . 89
  Optional parameters . . . 89

  Example 90
  Error codes 90

Chapter 8. Error handling and response messages 91
Error handling by the broker 91
Response messages 92
  Message descriptor for response messages 92
  Types of error response 93
    OK response 93
    Warning response 93
    Error response 93
Broker responses 94
  Standard parameters 94
  Optional parameters 95
  Examples 96
  Error codes applicable to all commands 97
Problem determination 98

Chapter 9. Sample programs 101
Sample application 102
  Running the application 102
  Possible extensions 104
Application Messaging Interface samples 105

Chapter 3. Introduction to writing applications

Applications use command messages to communicate with the broker when they want to publish or subscribe to information. These messages use the MQSeries Rules and Formatting Header (RF Header), which is described in Chapter 6, "Format of command messages" on page 61. The content of each command message starts with an MQRFH structure. This structure contains a name/value string, which defines the type of command the message represents and any parameters associated with the command. In the case of a **Publish** command message, the name/value string is usually followed by the data to be published, in any format specified by the user. Broker responses to command messages also use the MQRFH structure.

The normal Message Queue Interface (MQI) calls (such as MQPUT and MQGET) can be used to put RF Header command messages to the broker queue, and to retrieve response messages and publications from their respective queues. The MQI is described in the *MQSeries Application Programming Guide*. Most command messages are sent to the broker's control queue (SYSTEM.BROKER.CONTROL.QUEUE), but **Publish** and **Delete Publication** command messages are sent to the appropriate stream queue at the broker (for example, SYSTEM.BROKER.DEFAULT.STREAM).

Alternatively, you can use the MQSeries Application Messaging Interface (AMI) to send messages to and receive them from the broker. The AMI constructs and interprets the fields in the RF Header, so you don't need to understand its structure. In addition, the application programmer is not concerned with details of how MQSeries sends the message. These details (for instance, the queue name and fields in the message descriptor) are contained in AMI services and policies set up by a system administrator.

Like MQSeries Publish/Subscribe, the AMI is available as a SupportPac.

This chapter describes the things that you need to know before you start writing a publisher or subscriber application. It discusses the following topics:

- "Message flows" on page 38
- "Message ordering" on page 42
- "Publisher and subscriber identity" on page 43
- "The message descriptor" on page 44
- "Persistence and units of work" on page 46
- "Limitations" on page 47
- "Using the Application Messaging Interface" on page 47

You can find more information about writing applications in:

- Chapter 4, "Writing publisher applications" on page 49
- Chapter 5, "Writing subscriber applications" on page 55
- Chapter 6, "Format of command messages" on page 61
- Chapter 7, "Publish/Subscribe command messages" on page 71
- Chapter 8, "Error handling and response messages" on page 91
- Chapter 15, "Finding out about other publishers and subscribers" on page 159

Sample programs to illustrate the techniques used are described in Chapter 9, "Sample programs" on page 101.

© Copyright IBM Corp. 1998, 2000

Message flows

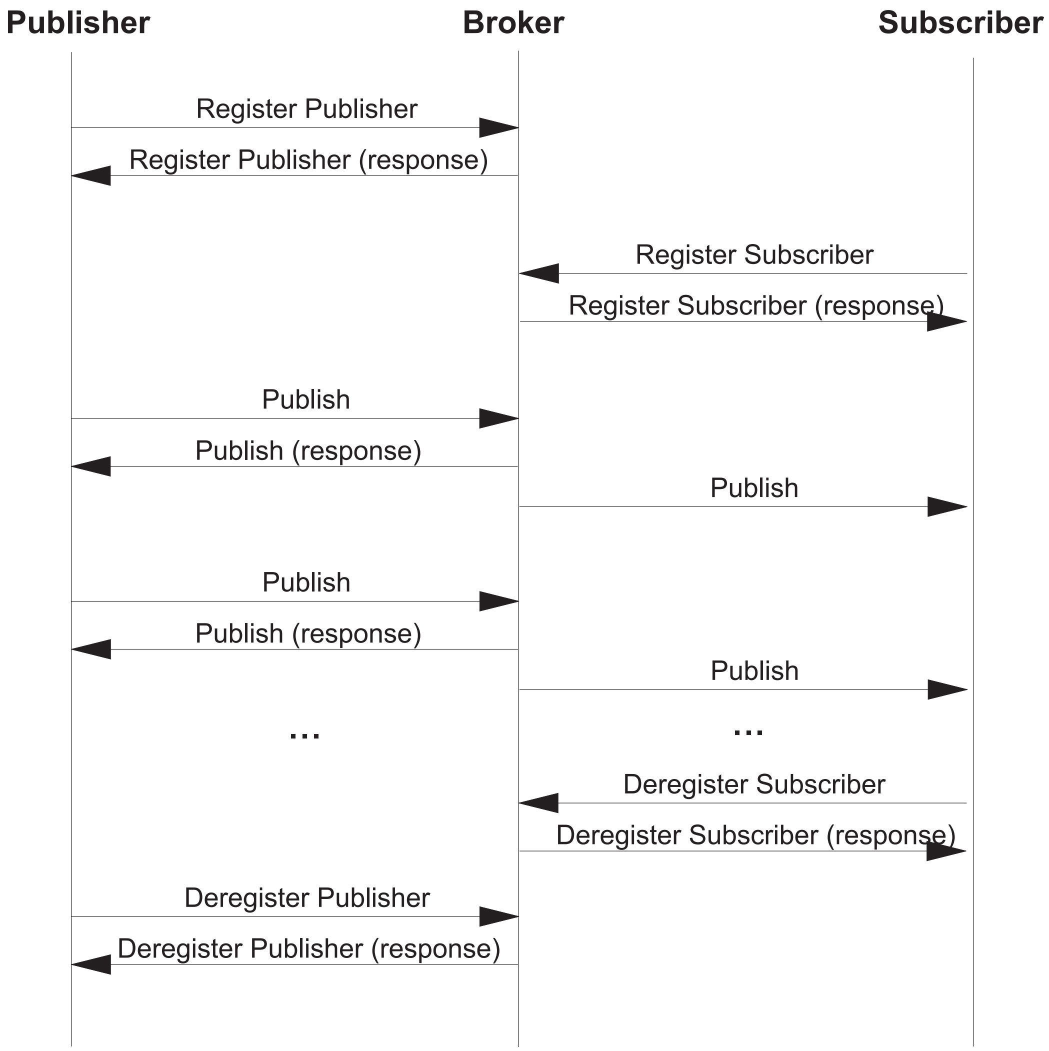

Publisher
Broker
Subscriber
Register Publisher
Register Publisher (response)
Register Subscriber
Register Subscriber (response)
Publish
Publish (response)
Publish
Publish
Publish (response)
Publish
...
...
Deregister Subscriber
Deregister Subscriber (response)
Deregister Publisher
Deregister Publisher (response)

Figure 9. Basic flow of messages

Figure 9 shows the basic flow of messages using the **Register Publisher**, **Deregister Publisher**, **Register Subscriber**, **Deregister Subscriber** and **Publish** command message and responses. This flow applies to all event publications, and to state information where the subscriber wants to get the latest published state of a topic.

The responses are optional, and the **Register Publisher** and **Deregister Publisher** command messages can be omitted (publishers can choose not to register, or to register on their first publish command). So the flow diagram can be simplified as shown in Figure 10 on page 39.

Simplified message flow

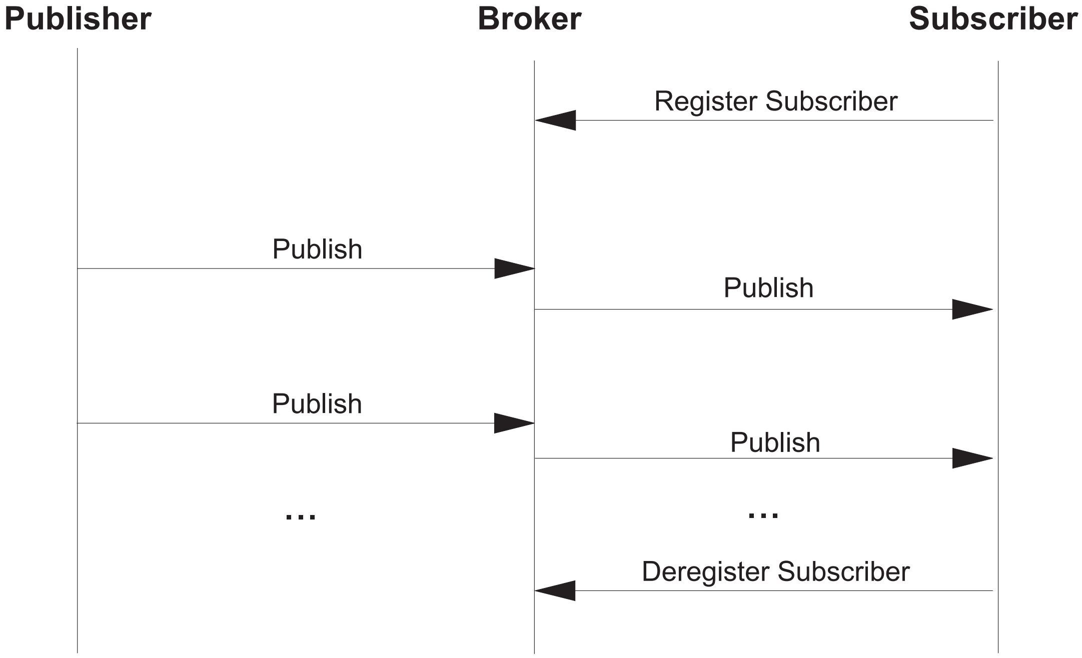

Publisher
Broker
Subscriber
Register Subscriber
Publish
Publish
Publish
Publish
...
...
Deregister Subscriber

Figure 10. Simplified flow of messages

Figure 10 is a simplified version of Figure 9 on page 38 with the optional messages and responses omitted.

Figure 11 on page 40 shows how publish and subscribe messages flow between the publisher, the subscriber, and the broker queues. In Figure 12 on page 40 this is extended to a two-broker system.

The flow of messages when retained publications are used is shown in Figure 13 on page 41. In this case, the subscriber receives the current retained publication as soon as it registers a subscription. In Figure 14 on page 41, the subscriber registers with the 'Publish on Request Only' option, so it doesn't receive the publication until it sends a **Request Update** command message. (Note that the first publication is not delivered to the subscriber, because it is updated by the second publication before the update request is received).

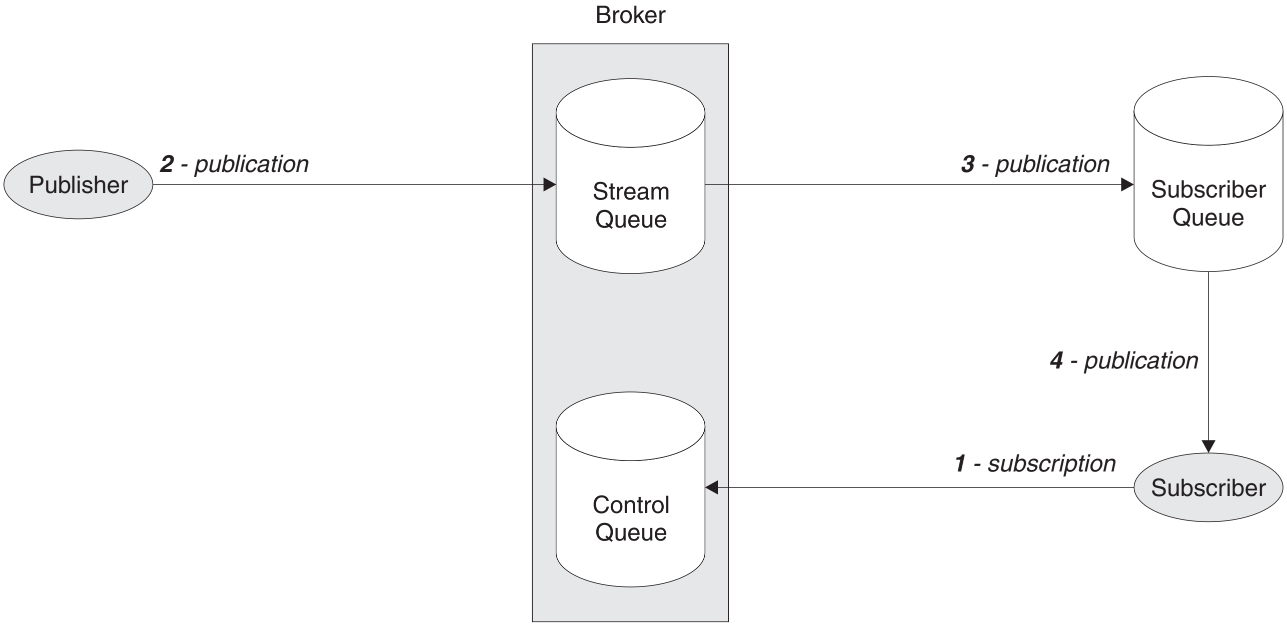

Broker
Publisher
2 - publication
Stream Queue
3 - publication
Subscriber Queue
4 - publication
1 - subscription
Subscriber
Control Queue

Figure 11. Flow of messages in a single-broker system. The subscriber registers a subscription by putting a message on the broker's control queue (1). Subsequently, a publisher puts a publication message, for the same topic, on the corresponding stream queue in the broker (2). The broker forwards the publication by putting the same message on the subscriber queue (3), from where the subscriber application can get it (4).

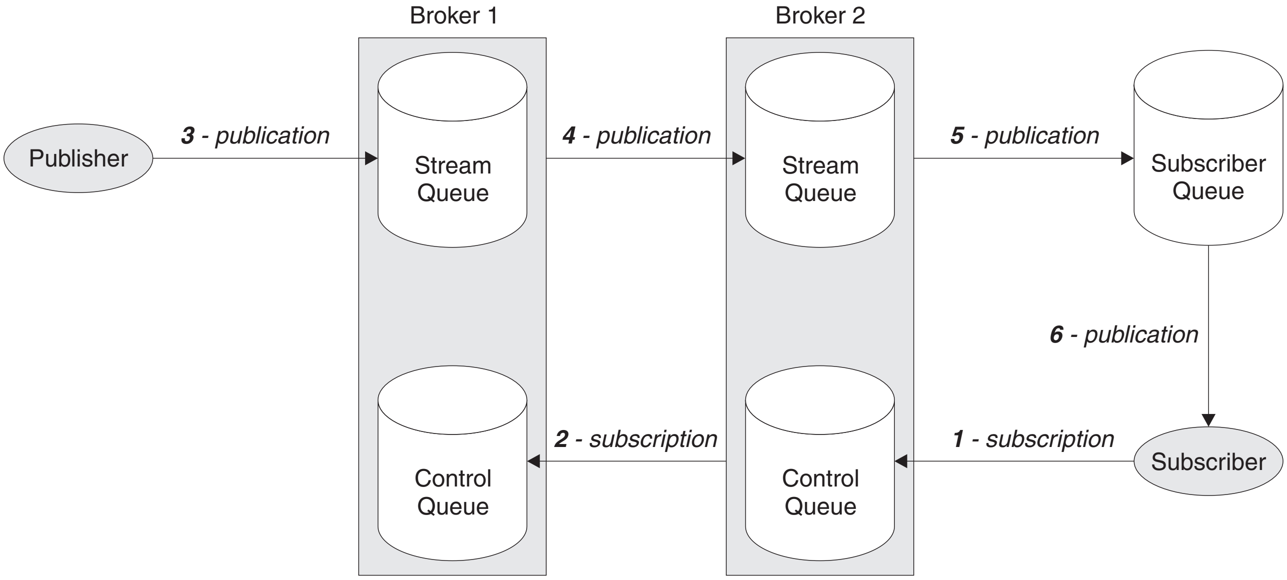

Broker 1
Broker 2
Publisher
3 - publication
Stream Queue
4 - publication
Stream Queue
5 - publication
Subscriber Queue
6 - publication
Control Queue
2 - subscription
Control Queue
1 - subscription
Subscriber

Figure 12. Flow of messages in a multi-broker system. The subscriber registers a subscription as in Figure 11(1). Broker 2 forwards the subscription by putting a message on the control queue of Broker 1 (2). Subsequently, a publisher puts a publication message, for the same topic, on the corresponding stream queue in Broker 1 (3). The publication is forwarded to Broker 2 (4), and then to the subscriber queue (5), from where the subscriber application can get it (6).

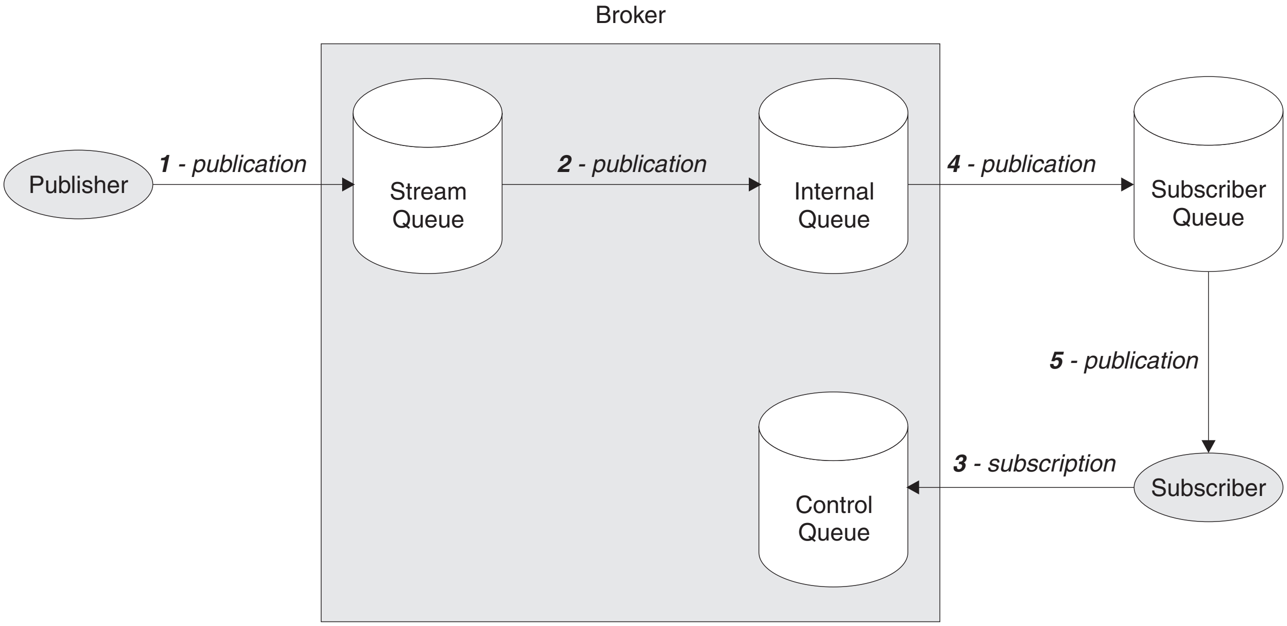

Broker
Publisher
1 - publication
Stream Queue
2 - publication
Internal Queue
4 - publication
Subscriber Queue
5 - publication
Control Queue
3 - subscription
Subscriber

Figure 13. Flow of messages using retained publications. A publisher sends a retained publication by putting a message on the appropriate stream queue in the broker (1). The broker stores the publication on an internal queue (2). Subsequently, a subscriber registers a subscription, to the same topic and stream, by putting a message on the broker's control queue (3). The broker sends the current retained publication for this topic by putting a message on the subscriber queue (4), from where the subscriber application can get it (5).

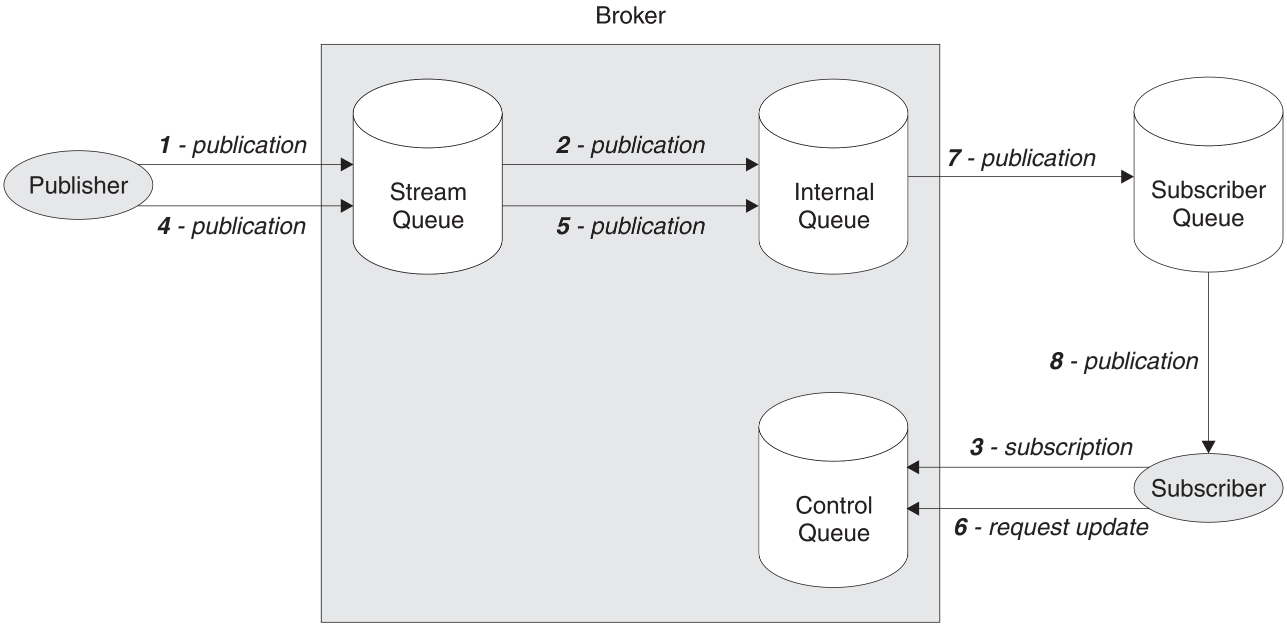

Broker
Publisher
1 - publication
4 - publication
Stream Queue
2 - publication
5 - publication
Internal Queue
7 - publication
Subscriber Queue
8 - publication
Control Queue
3 - subscription
6 - request update
Subscriber

Figure 14. Flow of messages using publish on request only. A publisher sends a retained publication to a stream queue in the broker (1). The broker stores it on an internal queue (2). A subscriber registers a subscription, to the same topic and stream, by putting a message on the broker's control queue (3), but it uses the 'Publish on Request Only' option so the broker takes no action. Subsequently, the publisher sends a second retained publication to the broker (4), which replaces the first one on the internal queue (5). The subscriber then sends a request update message to the broker's control queue (6). This causes the broker to send the current retained publication to the subscriber queue (7), from where the subscriber application can get it (8).

Message ordering

For a given stream, messages are published by brokers in the same order as they are received from publishers (subject to reordering based on message priority). This normally means that each subscriber receives messages from a particular broker, on a particular topic and stream, from a particular publisher in the order that they are published by that publisher.

However, as with all MQSeries messages, it is possible for messages, occasionally, to be delivered out of order. This could happen:

- If a link in the network goes down and subsequent messages are rerouted along another link
- If a queue becomes temporarily full, or put-inhibited, so that a message is put to a dead-letter queue and therefore delayed, while subsequent messages pass straight through.
- If the administrator deletes a broker or uses the **clrmqbrk** command when publishers and subscribers are still operating, causing queued messages to be put to the dead-letter queue and subscriptions to be interrupted.

If these circumstances cannot occur, publications will always be delivered in order.

Ensuring that messages are retrieved in the correct order

If you need to ensure that your messages are delivered in the correct order in all circumstances, you can use one of the following strategies:

- A *SequenceNumber* parameter is supported on the **Publish** message. A publisher can include this with each message, increasing the value by one for each successive message that it publishes for the same stream and topic. The broker does not check or set this parameter; the responsibility for it lies with the publisher. The number can be checked by the subscriber, which needs to remember the last sequence number it received for each stream and topic combination.

  If a subscriber receives a publication message that is out of order, it can react in various ways:

  - If it needs only the latest information (for example, a stock price) and the sequence number is greater than it should be (that is, one or more previous publications have not yet been received), this publication message is accepted. If the sequence number is less than it should be (that is, this is a previous publication), the publication message is ignored.
  - If it needs to keep track of all information, it must record this information and its sequence number.
- A *PublishTimestamp* parameter, in Universal time, is provided on the **Publish** message. A publisher can include this with each message (with or without the *SequenceNumber* parameter). This is particularly useful if subscribers are only interested in the latest information; they can check whether the timestamp is greater than that of the last **Publish** message that they processed.

In both of the above solutions, the publisher and subscriber need to remember information about the last message they processed for a particular stream and topic. In the first solution this is the *SequenceNumber* for the **Publish** message, and in the second solution it is the *PublishTimestamp*. This information might need to be remembered atomically with issuing or receiving a publication. This can be accomplished by saving the information on a queue, using the same unit-of-work as the one in which the publication is put or retrieved.

Publisher and subscriber identity

A publisher's or subscriber's identity consists of the following:

- Their queue name
- Their queue manager name (this can be blank to indicate the local queue manager).
- Correlation identifier (this is optional).

The correlation identifier can be used to distinguish between different publishers or subscribers using the same queue. If different subscribers are using the same queue, all publications sent by the broker to a subscriber specify the correlation identifier in the *CorrelId* field of the message descriptor (MQMD).

Note: For responses, MQRO_xx_CORREL_ID report options determine the correlation identifier used. Applications using a correlation identifier for identification will typically specify the *CorrelId* and the MQRO_PASS_CORREL_ID option.

The recipient can then use **MQGET** with the *CorrelId* to retrieve the messages.

This allows several applications to share a queue (this might be desirable if there are many clients). It also allows one application to distinguish between publications arising from different subscriptions. An example of this is in the sample program described on page 30. When the results service restarts, it subscribes to the topic `Sport/Soccer/State/LatestScore/*`, with the 'Publish on Request Only' option. It uses a different *CorrelId* from that used to subscribe to the `Sport/Soccer/Event/*` publications. This allows it to retrieve from the same queue all of the retained 'LatestScore' publications before it starts processing the event publications again.

An identity that includes the correlation identifier in the message descriptor is established by including MQPS_CORREL_ID_AS_IDENTITY in the *RegistrationOptions* parameter of the **Register Publisher** or **Register Subscriber** message (or of the **Publish** message for implicit registration). The correlation identifier to be used as part of the identity must not be zero.

If MQPS_CORREL_ID_AS_IDENTITY is not set, the identity does not include the correlation identifier and the broker uses a correlation identifier of its own choosing when sending messages to that publisher or subscriber. When a broker selects the correlation identifier itself, this will not conflict with other message identifiers or correlation identifiers generated by queue managers.

A single publisher or subscriber queue can therefore support multiple identities, each with a specific correlation identifier value, plus one further identity for which the correlation identifier is not specified (MQPS_CORREL_ID_AS_IDENTITY was not set for registration). Each of these identities is treated by the broker as being independent of the others. (Usually, however, a queue will either have a number of identities each with its own specific correlation identifier, or it will have only one identity with no specific correlation identifier).

MQPS_CORREL_ID_AS_IDENTITY should be set by a publisher whose identity includes a correlation identifier when sending a **Publish** message to the broker, so that the broker can identify the publisher using the *CorrelId* field in the MQMD. If such a message is received by the broker when there is no registration in effect for the publisher's queue and the correlation identifier specified, an implicit registration is performed (unless MQPS_NO_REGISTRATION is specified).

When a **Publish message** is sent by a broker to a subscriber whose identity includes a correlation identifier, the *CorrelId* field in the MQMD is set to the required correlation identifier. The correlation identifier sent to the subscriber depends only upon what the subscriber set when it registered. The correlation identifier used by the publisher is independent of the correlation identifier sent to the subscriber.

MQPS_CORREL_ID_AS_IDENTITY is valid for the **Deregister Publisher** and **Deregister Subscriber** message, to delete a registration for an identity that includes a correlation identifier.

The value used for a correlation identifier that is part of a publisher's or subscriber's identity needs to be unique only between the other users of the same queue. The MQPMO_NEW_CORREL_ID option can be used to cause the queue manager to generate a unique value.

The message descriptor

This section gives information about the values you should set in the message descriptor (MQMD) for messages that you send to the broker. It also explains the values that the broker sets in the message descriptor for publication messages it forwards to subscribers.

Messages sent to the broker

This section shows the values set for fields in the MQMD for messages sent to the broker.

Report

See *MsgType* (below), and “Error handling by the broker” on page 91.

MsgType

Can be set to MQMT_REQUEST for a command message if a response is always required. The MQRO_PAN and MQRO_NAN flags in the *Report* field are not significant in this case.

Can be set to MQMT_DATAGRAM, in which case responses depend on the setting of the MQRO_PAN and MQRO_NAN flags in the *Report* field:

- MQRO_PAN alone means that the broker is to send a response only if the command succeeds.
- MQRO_NAN alone means that the broker is to send a response only if the command fails.
- If a command succeeds partially a response is sent if either MQRO_PAN or MQRO_NAN is set.
- MQRO_PAN + MQRO_NAN means that the broker is to send a response whether the command succeeds or fails. This has the same

effect from the broker's perspective as setting *MsgType* to MQMT_REQUEST.

- If neither MQRO_PAN nor MQRO_NAN is set, no response will ever be sent.

Format

Set to MQFMT_RF_HEADER.

MsgId

Normally set to MQMI_NONE, so that the queue manager generates a unique value.

CorrelId

Specifies the *CorrelId* which can optionally be included as part of the subscriber's identity. When used with the MQRO_PASS_CORREL_ID option in the *Report* field, it will also be set in all response messages sent by the broker to the sender.

ReplyToQ

This is the queue to which responses, if any, are to be sent. This can be the sender's publisher or subscriber queue which has the advantage that the *QName* parameter can be omitted from the message text. If, however, responses are to be sent to a different queue, the *QName* parameter will be needed.

ReplyToQMgr

Queue manager for responses.

Note that a putting application can leave this field blank (the default value), in which case the local queue manager puts its own name in this field.

Expiry

Expiry of the subscription or publication.

Publications forwarded by the broker

This section shows the values set for fields in the MQMD for publications sent by the broker to subscribers.

The fields are set to default values, except for the following:

Report

Set to MQRO_NONE.

MsgType

Set to MQMT_DATAGRAM.

Expiry

Set to the value in the **Publish** message received from the publisher. In the case of a retained message, the time outstanding is reduced by the approximate time the message has been at the broker.

Format

Set to MQFMT_RF_HEADER.

MsgId

Set to MQMI_NONE, so that the queue manager generates a unique value.

CorrelId
: If *CorrelId* is part of the subscriber's identity, this is the value specified by the subscriber when registering. Otherwise, it is a non-zero value chosen by the broker.

Priority
: Set by the publisher or as a resolved value if the publisher specified MQPRI_PRIORITY_AS_Q_DEF.

Persistence
: Set by the publisher or as a resolved value if the publisher specified MQPER_PERSISTENCE_AS_Q_DEF.

ReplyToQ
: Set to blanks.

ReplyToQMgr
: Broker's queue manager name.

UserIdentifier
: Subscriber's user identifier (as set when the subscriber registered).

AccountingToken
: Subscriber's accounting token (as set when the subscriber registered).

ApplIdentityData
: Subscriber's application identity data (as set when the subscriber registered).

PutApplType
: Set to MQAT_BROKER.

PutApplName
: Set to the first 28 characters of the broker's queue manager name.

PutDate
: Timestamp when the broker puts the message.

PutTime
: Timestamp when the broker puts the message.

ApplOriginData
: Set to blanks.

Persistence and units of work

Subscriber and publisher registration messages should normally be sent as persistent messages (registrations themselves are always persistent, regardless of the persistence of the messages that caused them). Publication messages can be either persistent or non-persistent. Brokers maintain the persistence and priority of publications as set by the publisher.

When reading messages from stream queues, brokers always read persistent messages within a unit-of-work, so that they are not lost if the broker or system crashes. Non-persistent messages might or might not be read within a unit-of-work, depending on the options set in the queue manager configuration file, qm.ini. This is described in "Broker configuration stanza" on page 112.

Publication messages are treated so that publication to subscribers is once and once only for persistent messages. For non-persistent messages, delivery to subscribers is also once only unless *SyncPointIfPersistent* was specified in the queue manager configuration file and the broker or queue manager stops abruptly. In this case, the message might be lost for one or more subscribers. Regardless of its persistence, however, a **Publish** message is never sent more than once to a subscriber, for a given subscription (unless **Request Update** is used).

Publishers and subscribers can choose whether or not to use a unit-of-work when publishing or receiving messages. However, if the *SequenceNumber* technique described previously is used for maintaining ordering, both publisher and subscriber must retain sequencing information atomically with putting or getting a message if the application is to be re-startable.

Limitations

This section describes some limitations of MQSeries Publish/Subscribe.

Group messages

Group messages are not supported by MQSeries Publish/Subscribe. If a group message is sent to the broker it will not cause an error, but the group message flags in the message descriptor will not be forwarded by the broker.

Segmented messages

Segmented messages are not supported by MQSeries Publish/Subscribe. If a segmented message is sent to the broker, it will be rejected as not valid.

If you want to distribute a segmented message to subscribers, you can publish a short notification that the message is available, offering to accept ‘direct requests’ for the full message (see “Publish” on page 78).

Cluster queues

Stream queues must not be cluster queues.

Data conversion of MQRFH structure

You might have a client application (publisher or subscriber) running on a version of MQSeries that does not support data conversion of the MQRFH structure. The application is able to pass publish/subscribe messages to other queue managers provided that CONVERT(NO) is specified on the sending channel.

Using the Application Messaging Interface

The MQSeries Application Messaging Interface (AMI) provides a simple interface that application programmers can use without needing to understand all the options available in the MQSeries Message Queue Interface (MQI). The options that are needed in a particular installation are defined by a system administrator, using services and policies.

The AMI has functions to generate the most commonly used publish/subscribe command messages, and to receive a publication from the broker. It is available for the C, C++, and Java™ programming languages. The name of the function (or method) depends on the programming language being used. In the case of C, there are two sets of functions: the high-level interface and the object interface.

AMI publish/subscribe functions

Publish command

C high-level	amPublish
C object-level	amPubPublish
C++	AmPublisher->publish
Java	AmPublisher.publish

Register Subscriber command

C high-level	amSubscribe
C object-level	amSubSubscribe
C++	AmSubscriber->subscribe
Java	AmSubscriber.subscribe

Deregister Subscriber command

C high-level	amUnsubscribe
C object-level	amSubUnsubscribe
C++	AmSubscriber->unsubscribe
Java	AmSubscriber.unsubscribe

Receive a publication

C high-level	amReceivePublication
C object-level	amSubReceive
C++	AmSubscriber->receive
Java	AmSubscriber.receive

These functions have parameters that enable you to specify some of the parameters in the command message, such as the *topic*. Other parameters in the command message are specified by the AMI *service* that you use to send the message (the service is set up by the system administrator). You can modify these parameters by changing the appropriate name/value elements before sending the command message; helper functions are provided for this purpose. Details of these name/value elements and the options that are available for each command are given in Chapter 7, “Publish/Subscribe command messages” on page 71.

There are no AMI functions to generate **Delete Publication**, **Deregister Publisher**, **Register Publisher**, or **Request Update** command messages directly. You have to construct a message containing the appropriate name/value elements using the helper functions provided, and then send the message to the broker.

Please refer to the *MQSeries Application Messaging Interface* book for details of how to use the functions mentioned above (including the name/value element helper functions).

Chapter 4. Writing publisher applications

Publisher applications communicate with the broker using command messages in the RF Header format (or the equivalent functions in the Application Messaging Interface). Publishers can register with the broker before they start publishing information, they can register implicitly with their first publication, or they can choose not to register. When they have finished publishing information, they can deregister with the broker. They can also delete retained publications. This chapter discusses the following:

- "Registering with the broker"
- "Publishing information" on page 51
- "Deleting information" on page 53
- "Deregistering with the broker" on page 53

You can see an example of a publisher application in Chapter 9, "Sample programs" on page 101.

The only configuration the administrator has to perform before you can define an application as a potential publisher is to set up the necessary security authorization to enable the application to put messages to the required stream queues, and, if explicit registration is required, to send messages to the broker's control queue (see "Broker queues" on page 109).

Registering with the broker

Publisher applications can register their intention to publish with a broker.

There are two ways for a publisher to register with a broker:

- The publisher can send a **Register Publisher** command message to the broker's control queue (SYSTEM.BROKER.CONTROL.QUEUE) to indicate that a publisher will be, or is capable of, publishing data on one or more specified topics. This message can also be sent by another application on a publisher's behalf. This command is described in "Register Publisher" on page 83.
- The publisher can register with the broker implicitly when it sends its first **Publish** command message to a stream queue at the broker (such as SYSTEM.BROKER.DEFAULT.STREAM or SAMPLE.BROKER.RESULTS.STREAM). This command is described in "Publish" on page 78. However, if the broker is not currently aware of the stream specified, a **Register Publisher** command message will be necessary to cause the broker to recognize the stream queue.

A publishing application might not know if a stream is supported by a particular broker. In this case it is recommended that the publisher issues the **Register Publisher** command message and waits for a response that indicates that the stream is known to the broker, before sending the first **Publish** command message.

An application can register with the same broker more than once, and can also register with many different brokers. An application that is already registered as a subscriber can also register as a publisher. This is the case in the sample application (see "Sample application" on page 30). The results service registers as

© Copyright IBM Corp. 1998, 2000

a *subscriber* to the events published by the match simulators, and as a *publisher* of the latest scores.

Choosing not to register

Publishers do not have to register with a broker. This saves the programming overhead of performing the registration, and of deregistering when the publisher has finished. However, other applications cannot find out about unregistered publishers because they do not appear in metatopics (see “Metatopics” on page 160 for information about these). Unregistered publishers can send **Publish** command messages to the broker, specifying that they do not want the broker to perform an implicit registration, provided that:

- The publisher does not need to be listed in the metatopics
- The publisher’s stream is already known to the broker

Options you can specify when registering as a publisher

When a publisher registers with a broker, it must specify the topics that it is going to publish information about. It can specify the name of more than one topic, but it cannot use wildcards to specify a range of topics.

Queue name

A publisher is required to specify a queue when it registers and also when it issues **Publish** command messages (unless it specifies the MQPS_NO_REGISTRATION option). This is the queue to which any **Request Update** command messages sent directly by a subscriber to this publisher are normally sent. The publisher specifies the queue to which any responses from the broker are to be sent using the *ReplyToQ* and *ReplyToQMgr* parameters; this queue can also be the publisher’s queue.

Selecting a stream

You can specify the name of the stream to which the specified topics apply. If you do not specify this, the SYSTEM.BROKER.DEFAULT.STREAM is assumed.

Publisher identity

The identity of the publisher consists of the name of the queue and queue manager that it uses, as described in “Publisher and subscriber identity” on page 43. You can specify these names when you register as a publisher. If you do not specify these names, the names of the reply-to queue and reply-to queue manager specified in the message descriptor (MQMD) of the command message are used for this instead.

You can also specify that you want to use the correlation identifier in the message descriptor as part of the publisher's identity.

A publisher can register anonymously. In this case its identity will not be divulged by the broker, except to subscribers to metatopics which have additional authority (see “Authorized metatopics” on page 162).

Registration scope

If the broker is part of a network, the publisher can specify whether it wants its publications (a) sent to subscribers who have registered local subscriptions on that broker only (a local publication), or (b) distributed to other brokers in the network and sent to all subscribers, including those on that broker, who have registered global subscriptions (a global publication).

Registration expiry

Publisher registrations do not expire, even if you specify a value for *Expiry* field of the message descriptor. The value you set for *Expiry* might however cause the command message to expire before it is processed by the broker.

Broker restart

Publisher registrations and retained publications are maintained across broker restarts.

Changing an application’s registration

If a publisher has registered, it can use the **Register Publisher** command message again to increase the range of topics it wants to publish for, or to change the options for topics that it has already registered for. This command should be sent to the broker’s control queue.

Publishing information

When an application wants to publish some information, it sends a **Publish** command message to the stream queue at the broker. This command is described in “Publish” on page 78.

The publisher must specify the topic to which the publication applies. If a publication matches several subscriptions for which a subscriber is registered, only one copy of the publication is sent to the subscriber for all matching subscriptions. The publisher can also specify the name of a stream; however, this is not necessary if the message is put to the correct stream queue at the broker.

If the publisher is not registered with the broker for those topics, the broker will automatically register the publisher when it receives this message, unless you tell it not to (see “Choosing not to register” on page 50).

If an application is registered as both a publisher and a subscriber for a topic, it can use an option when publishing to say that it does not want to receive a copy of this publication.

Publication data

Publishers can include the publication data in the message, or they can refer to it.

Including data in the message

Publication data is usually appended to the **Publish** command message, following the *NameValueString* of the MQRFH header, as shown in “Publication data” on page 68. The characteristics of the data are defined in the *Encoding*, *CodedCharSetId* and *Format* fields of the MQRFH header. Alternatively, string data can be contained within the *NameValueString*.

Referring to data in the message

Publishers can make information available to subscribers directly, without going through the broker. The publisher needs to advertise the fact that it is publishing information about a topic, and that it is willing to receive direct requests for this information from subscribers.

There are two ways that a subscriber can find out about this information:

- From a publication received in a normal way.

  The publisher can use a normal publication to advertise the fact that it has more information about a topic (for example, a large file in several different formats). The publisher should also specify the topic name to be used (which could be the same, or different) and where the subscriber will find the information.

- From a subscription to the metatopics.

  The publisher can register with the broker specifying that it will accept direct requests for information about a topic. Subscribers that request information about publishers (metatopics) will discover the names of publishers who publish on this topic. (See “Metatopics” on page 160 for information about metatopics.)

Retained publications

When a publication specifies that it is to be retained, any previously retained publication for this stream and topic combination is replaced, so that the information is always at the latest level. See “Retained publications” on page 28 for information about retained publications.

Mixing retained and non-retained publications on the same topic in a stream is not recommended. If an application does this and publishes a non-retained publication, any previously retained publication is still retained.

It is not recommended for two or more applications to publish retained publications to the same topic and stream. If two applications do publish a retained publication about the same topic on the same stream simultaneously, it is difficult to determine which publication is retained. If these publishers use two different brokers, it is possible that different retained publications could be active at different brokers for the same topic and stream.

Expiry of retained publications

Use the *Expiry* field of the message descriptor (MQMD) of the publish message to set an expiry interval for a retained message.

Publishing locally and globally

Publishers can specify that they want a publication to be published locally. If they do not specify this, the publication is made available globally through all the brokers in the network. Local publications can only be received by subscribers who register local subscriptions at the same broker as the publisher. Local retained publications are only retained at this broker.

Applications can publish and subscribe locally to the same topic and stream at different brokers. Each broker will deal with the publications and subscriptions in isolation from the other brokers.

Mixing local and global publications and subscriptions to the same topic and stream is not recommended. A local publication will not be delivered to a subscriber registered globally, even if they are at the same broker.

Deleting information

Publishers can request that the broker delete retained publications for specified topics. To do this, send the **Delete Publication** command message to the stream queue at the broker to tell it to delete its copy of any data for the specified topics. This command is described in “Delete Publication” on page 72.

The application needs the same authority to delete publications as it needs to publish messages for the specified stream. You do not have to be a registered publisher to be able to delete publications.

If you want to delete some of the information that was originally published in a message that covered more than one topic, the broker deletes the publication only for the topics you specify, and retains the rest.

If different publishers publish data on the same stream and topics, the data that is deleted might have originated from a different publisher.

You can also specify if you want to delete retained publications published locally at the broker, or those published globally.

Deregistering with the broker

When a publisher that is registered with a broker no longer wishes to publish information on a topic, it can use the **Deregister Publisher** command message to deregister with the broker. This message should be sent to the SYSTEM.BROKER.CONTROL.QUEUE. This command is described in “Deregister Publisher” on page 74.

This command can be used if the publisher registered with the broker explicitly using **Register Publisher**, or implicitly using **Publish**. A publisher cannot deregister if it chose not to register in the first place.

The application must specify one of the following:

- Deregister for all topics for which it was registered.
- Deregister for a subset of the topics for which it is registered if it wants to continue publishing on other topics. It must specify one or more topics, and it can use wildcards.

You must specify the stream name for these topics, unless it is the default (SYSTEM.BROKER.DEFAULT.STREAM).

You must also specify the name of the publisher’s queue and queue manager.

The publisher registration must be deregistered by the same user that registered it originally, unless the deregistering application is allowed to put the message as the appropriate user (for example using alternate user authority to open the SYSTEM.BROKER.CONTROL.QUEUE for that user).

Chapter 5. Writing subscriber applications

Subscriber applications communicate with the broker using command messages in the RF Header format (or the equivalent functions in the Application Messaging Interface). Subscribers need to register with a broker before they can start receiving publications. They can also request certain types of publication from the broker or directly from the publisher.

This chapter discusses the following:

- "Registering as a subscriber"
- "Requesting information" on page 58
- "Deregistering as a subscriber" on page 59

You can see an example of a subscriber application in Chapter 9, "Sample programs" on page 101.

Registering as a subscriber

Subscriber applications need to register their interest in receiving publications with a broker. Before you can define an application as a potential subscriber, you must set up the necessary security authorization to enable the application to:

- put a message to the broker's control queue
- browse the required stream queues
- put a message to the subscriber queue that will be used to receive publications

Send the **Register Subscriber** command message to the SYSTEM.BROKER.CONTROL.QUEUE to register as a subscriber. This command is described in "Register Subscriber" on page 86.

Your application should send this message to a broker's control queue (see "Broker queues" on page 109). to indicate that it wishes to subscribe to the topics specified in the message. Alternatively, an application can send this message to register on behalf of another application that wishes to subscribe. If an application subscribes on behalf of another application, the user ID of the subscribing application is used. The application will need alternate user authority if a different user ID is to be used. An application that has already registered as a publisher can also register as a subscriber.

An application can register with the same broker more than once, and can also register with many different brokers.

When a subscriber has registered with a broker, the subscription is persistent and survives broker and queue manager restarts, regardless of the persistence of the **Register Subscriber** command message.

When a subscriber registers with the broker, it must specify the topics that it is interested in. It can specify the name of more than one topic, and it can also use wildcards to specify a range of topics (described in "Topics" on page 23). If a subscriber has many (different) registrations that match the topic of a publication, only one copy of the publication is sent to it.

© Copyright IBM Corp. 1998, 2000

Subscriber queues

A subscriber queue is the queue where publications for that subscriber will be sent. The subscriber specifies the name of the queue when it registers a subscription. If the subscriber is at the same queue manager as the broker, the subscriber's queue name must not be the same as that of the stream. Such a subscription will be rejected. Even if the subscriber's and broker's queue managers are different, it is strongly recommended that you use different names for the queues.

If a subscribing application registers multiple subscriptions (for the same or different streams), it can choose whether all **Publish** command messages are sent to the same queue, or whether **Publish** command messages for different subscriptions go to different queues.

The queue name, queue manager name and correlation identifier (if one is specified) of a subscriber's queue are used by the broker to identify the subscriber (as described in “Publisher and subscriber identity” on page 43). When the broker publishes information about subscribers, if a subscriber has registered several subscriptions for the same stream that are all to be sent to the same queue (and the subscriptions are not distinguished with different correlation identifiers, the subscriber appears as a single application.

If publications for different subscriptions are sent to different queues, or use a different *CorrelId*, the broker regards these as being from multiple subscribers (even though the subscriber might be a single application).

Options you can specify when registering as a subscriber

The options that a subscriber specifies when registering determine which publications (if any) are sent to it by the broker. Any previously retained publications for the topics specified are sent immediately after registration (unless the subscriber specifies new publications only, which are those published *after* the subscriber registered with the broker).

Alternatively, the subscriber can request that it is not sent any publications about a topic unless it asks for them using the **Request Update** command message. This method is applicable where publications have been retained, and an application might want to know the latest information about a topic.

Queue name

The queue where messages for a subscriber should be sent is called the subscriber queue. This queue should not be a temporary dynamic queue. The subscriber specifies the name of the queue when it registers a subscription.

Selecting a stream

You can specify the name of the stream to which the specified topics apply. If you do not specify this, the SYSTEM.BROKER.DEFAULT.STREAM is used.

You can also request that publication messages that will be sent to the subscriber include the name of the stream to which the publication applies, even if the publisher did not include the name in the publication.

Subscriber identity

The identity of the subscriber consists of the name of the queue and queue manager that it uses, as described in “Publisher and subscriber identity” on page 43. You can specify these names when you register as a subscriber. If you do not specify these names, the names of the reply-to queue and reply-to queue manager specified in the message descriptor (MQMD) of the command message are used for this instead.

You can also use the correlation identifier in the message descriptor as part of the subscriber’s identity. You might need to do this if, for example, the broker publishes information about subscribers, and a subscriber has registered several subscriptions for the same stream that are all to be sent to the same queue. If the subscriptions are not distinguished with different correlation identifiers, the subscriber appears as a single application.

If the different subscriptions are to be sent to different queues, the broker believes that these are from multiple subscribers even though the subscriber might be a single application.

If required, you can tell the broker that the identity of the subscriber should not be divulged by the broker when the broker publishes information about subscribers (unless the request comes from a subscriber with additional authority).

Subscription scope

If the broker is part of a network, the subscriber can specify whether it wants to subscribe to local publications sent to the local broker only, or whether it wants its subscription distributed to other brokers in the network.

Subscription expiry

The values you set for the *Expiry* attribute in the message descriptor (MQMD) of the **Register Subscriber** command message determines when the subscription will expire. This is measured from the time the subscription request is put. This means that the message could expire before the subscriber is registered with the broker. If this is set to MQEI_UNLIMITED, the subscription does not expire, and the subscriber will continue to receive publications until it explicitly deregisters.

Broker restart

Subscriber registrations are maintained across broker restarts. Any subsequent publications for the specified topics will be forwarded to the subscriber, including any that arrived while the broker was inactive.

Changing an application’s registration

When a subscriber has registered, it can use the **Register Subscriber** command message again to increase the range of topics that it wants to receive information for, or to change the options for topics that it has already registered for.

When a subscription is re-registered, the values you set for the *Expiry* attribute in the message descriptor (MQMD) of the **Register Subscriber** command message determines when the subscription will expire. This is measured from the time the subscription request is put. Thus the **Register Subscriber** command message can be used to refresh a subscription before it expires.

Requesting information

A subscriber can request information from the broker, or directly from a publisher.

Requesting information from the broker

A subscriber can request a retained publication on a specified topic from the broker. To do this, it uses the **Request Update** command message, which is described in “Request Update” on page 89. Applications usually do this if, when they registered with the broker, they asked to be sent publications on request only. If the broker has a retained publication for the topic specified, it is sent to the subscriber.

This command message can also be sent by a subscriber that did not register in this way, to request that the latest copy of a publication be sent to it. This might be necessary if a subscriber has already seen a publication, but has failed without saving it, and on restart wants to see it again.

This command message can be satisfied only by a retained publication at the broker (see “State and event information” on page 28). If the broker to which this message is sent has no retained publication for the topic specified, the request fails.

Requesting information from a publisher

Under some circumstances, subscribers can request information directly from a publisher without involving the broker.

A publisher can specify that it is willing to receive direct requests for information from other applications. In this case, the publisher must make its queue and queue manager names (and possibly correlation identifier) known to subscribers by including them in a publication that advertises the availability of other publications on direct request.

Alternatively, subscribers can subscribe to information about publishers (called metatopics). They can discover the names of publishers who are willing to accept direct requests for publications on this topic. (See “Metatopics” on page 160 for information about metatopics.)

The subscriber can use this information to send a normal MQSeries message (using the MQI) directly to the publisher. The publisher can then use the MQI to send the publication directly to the subscriber.

Deregistering as a subscriber

When a subscriber no longer wishes to receive publications on a topic, send the **Deregister Subscriber** command message to the broker's control queue. This command is described in "Deregister Subscriber" on page 76.

This tells the broker that it should stop sending publications, about the topics specified, to the subscriber.

An application must specify one of the following:

- Deregister for all topics for which it was registered.
- Deregister for a subset of the topics for which it is registered if it still wants to receive publications on other topics. It must specify one or more topics. If the original subscription used wildcards, it must be deregistered using the same wildcard topic.

You must specify the stream name for these topics, unless it was the default (SYSTEM.BROKER.DEFAULT.STREAM).

You must also specify the name of the subscriber's queue and queue manager, unless they are the same as the reply-to queue and reply-to queue manager in the message descriptor of the command message. The subscription must be deregistered by the same user that registered it originally, unless the deregistering application is allowed to put the **Deregister Subscriber** message as the appropriate user (for example using alternate user authority to open the SYSTEM.BROKER.CONTROL.QUEUE for that user and *CorrelId*).

Chapter 6. Format of command messages

Applications use command messages to communicate with the broker when they want to publish or subscribe to information. These messages use the MQSeries Rules and Formatting Header (RF Header). Each message or response starts with an MQRFH structure, which includes a *NameValueString*. This consists of a succession of tag names and values (name/value pairs), which define the type of command the message represents and any options that apply to it. In the case of a **Publish** command message, the MQRFH header is usually followed by the data being published, in a format defined in the MQRFH structure. Alternatively, string publication data can be included within the *NameValueString*, using appropriate tag names and values defined by the publisher.

This chapter discusses the following topics:

- "MQRFH – Rules and formatting header" on page 62
- "Publish/Subscribe name/value strings" on page 65
- "Publication data" on page 68

The name/value pairs that define the parameters needed for the command messages are detailed in Chapter 7, "Publish/Subscribe command messages" on page 71.

If you are using the MQSeries Application Messaging Interface (AMI) to communicate with the broker, you don't need to understand all the information in this chapter. The AMI constructs and interprets the RF Header and its name/value pairs (see "Using the Application Messaging Interface" on page 47). However, you might find it useful to read this chapter, in particular the section on publication data.

© Copyright IBM Corp. 1998, 2000

MQRFH – Rules and formatting header

The following table summarizes the fields in the structure.

Table 2. Fields in MQRFH

Field	Description	Page
StrucId	Structure identifier	62
Version	Structure version number	62
StrucLength	Total length of MQRFH including string containing name/value pairs	63
Encoding	Numeric encoding	63
CodedCharSetId	Coded character set identifier	63
Format	Format name	63
Flags	Flags	64
NameValueString	String containing name/value pairs	64

The MQRFH structure defines the format of the rules and formatting header. This header can be used to send string data in the form of name/value pairs.

The format name of an MQRFH structure is MQFMT_RF_HEADER. The fields in the MQRFH structure and the name/value pairs 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.

Character data in the MQRFH (including the *NameValueString* field) must belong to a single-byte character set (SBCS). The user data that follows *NameValueString* can belong to any supported character set (SBCS or DBCS).

This structure is supported in the following environments: AIX, Compaq Tru64 UNIX, DOS client, HP-UX, Linux, OS/2, OS/390, Sun Solaris, Windows client, Windows NT, Windows 2000.

Fields

StrucId (MQCHAR4)

Structure identifier.

The value must be:

MQRFH_STRUC_ID

Identifier for rules and formatting header structure.

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

The initial value of this field is MQRFH_STRUC_ID.

Version (MQLONG)

Structure version number.

The value must be:

MQRFH_VERSION_1
Version-1 rules and formatting header structure.

The initial value of this field is MQRFH_VERSION_1.

StrucLength (MQLONG)
Total length of MQRFH including string containing name/value pairs.

This is the length in bytes of the MQRFH structure, including the *NameValueString* field at the end of the structure. The length does *not* include any user data that follows the *NameValueString* field.

To avoid problems with data conversion of the user data in some environments, it is recommended that *StrucLength* should be a multiple of four.

The following constant gives the length of the *fixed* part of the structure, that is, the length excluding the *NameValueString* field:

MQRFH_STRUC_LENGTH_FIXED
Length of fixed part of MQRFH structure.

The initial value of this field is MQRFH_STRUC_LENGTH_FIXED.

Encoding (MQLONG)
Numeric encoding.

This specifies the representation used for numeric values in the user data (if any) that follows the string containing the name/value pairs. This applies to binary integer data, packed-decimal integer data, and floating-point data.

The initial value of this field is MQENC_NATIVE.

CodedCharSetId (MQLONG)
Coded character set identifier.

This specifies the coded character set identifier of character strings in the user data (if any) that follows the string containing the name/value pairs.

Note: When a message is put, this field must be set to the nonzero value that specifies the character set of the user data. If this is not done, it will not be possible to convert the message using the MQGMO_CONVERT option when the message is retrieved.

The initial value of this field is `0`.

Format (MQCHAR8)
Format name.

This specifies the format name of the user data (if any) that follows the string containing the name/value pairs.

The name should be padded with blanks to the length of the field. Do not use a null character to terminate the name before the end of the field, as the queue manager does not change the null and subsequent characters to blanks in the MQRFH structure. Do not specify a name with leading or embedded blanks.

The initial value of this field is MQFMT_NONE.

Flags (MQLONG)
Flags.

The following can be specified:

MQRFH_NONE
No flags.

The initial value of this field is MQRFH_NONE.

NameValueString (MQCHARn)
String containing name/value pairs.

This is a variable-length character string containing name/value pairs in the form:

```
name1 value1 name2 value2 name3 value3 ...
```

Each name or value must be separated from the adjacent name or value by one or more blank characters; these blanks are not significant. A name or value can contain significant blanks by prefixing and suffixing the name or value with the double-quote character; all characters between the open double-quote and the matching close double-quote are treated as significant. In the following example, the name is `FAMOUS_WORDS`, and the value is `Hello World`:

```
FAMOUS_WORDS "Hello World"
```

A name or value can contain any characters other than the null character (which acts as a delimiter for *NameValueString* – see below). However, to assist interoperability, an application may prefer to restrict names to the following characters:

- First character: upper or lower case alphabetic (`A` through `Z`, or `a` through `z`), or underscore.
- Second character: upper or lower case 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, lower-case letters are not considered to be the same as upper-case letters. For example, `FAMOUS_WORDS` and `Famous_Words` are two different names.

The length in bytes of *NameValueString* is equal to *StrucLength* minus MQRFH_STRUC_LENGTH_FIXED. To avoid problems with data conversion of the user data in some environments, it is recommended that this length should be a multiple of four. *NameValueString* must be padded with blanks to this length, or terminated earlier by placing a null character following the last value in the string. The null and bytes following it, up to the specified length of *NameValueString*, are ignored.

Note: Because the contents and length of the *NameValueString* field are not fixed, no initial value is given for this field, and it is omitted from the “Structure definition in C” on page 65.

Table 3. Initial values of fields in MQRFH

Field name	Name of constant	Value of constant
StrucId	MQRFH_STRUC_ID	`'RFHƀ'` (See note 1)
Version	MQRFH_VERSION_1	`1`
StrucLength	MQRFH_STRUC_LENGTH_FIXED	`32`
Encoding	MQENC_NATIVE	See note 2
CodedCharSetId	None	`0`
Format	MQFMT_NONE	`'ƀƀƀƀƀƀƀƀ'`
Flags	MQRFH_NONE	`0`

Notes:

1. The symbol ‘ƀ’ represents a single blank character.
2. The value of this constant is environment-specific.
3. 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};`

Structure definition in C

```
typedef struct tagMQRFH {
  MQCHAR4  StrucId;         /* Structure identifier */
  MQLONG   Version;         /* Structure version number */
  MQLONG   StrucLength;     /* Total length of MQRFH including string
                               containing name/value pairs */
  MQLONG   Encoding;        /* Numeric encoding */
  MQLONG   CodedCharSetId;  /* Coded character set identifier */
  MQCHAR8  Format;          /* Format name */
  MQLONG   Flags;           /* Flags */
 } MQRFH;
```

Publish/Subscribe name/value strings

The MQRFH format is used to encode command messages that are sent to the MQSeries Publish/Subscribe broker. The *NameValueString* field within the RF header contains name/value pairs that describe the command to be carried out by the broker. If the command being issued is a **Publish** command, publication data (in a format defined by the publisher) can follow the *NameValueString* field.

The *NameValueString* can contain any number of name/value pairs, but only those in which the tag-name begins with the characters ‘MQPS’ will be recognized by the broker. Other name/value pairs (which can be defined by the publisher to encode publication data, for instance) will be ignored by the broker.

The first occurrence of an ‘MQPS’ tag-name must be `MQPSCommand`, followed by a tag-value which identifies the command to be carried out. Subsequent ‘MQPS’ tag-names and their values identify any options for that command (if they occur *before* the `MQPSCommand` tag-name, the command will fail).

Each name or value must be separated from the adjacent name or value by one or more blank characters. The C header file **cmqpsc.h** defines tag-names and values that can be used by publisher and subscriber applications when building command messages to be sent to the broker. Blank enclosed versions of the constants are provided to simplify construction of a *NameValueString*. For example, topics are specified using a tag-name of `MQPSTopic`, and the following three constants are provided in the **cmqpsc.h** header file:

```
#define MQPS_TOPIC    "MQPSTopic"
#define MQPS_TOPIC_B  " MQPSTopic "
#define MQPS_TOPIC_A  ' ','M','Q','P','S','T','o','p','i','c',' '
```

The MQPS_TOPIC constant is not enclosed by blanks. If it is used to build a *NameValueString*, the application must add blanks between tag-names and values. The version of the constant with the '_B' suffix includes the necessary blanks. The version with the '_A' suffix also includes the blanks, but is in character array form. These constants are most suited for initialization of a C structure which is being used to define a fixed layout of a *NameValueString*.

For example, the **Delete Publication** command can be issued to delete retained publications throughout the broker network. A topic of `'*'` matches all topics within the stream which the command is sent to, so using this will delete all retained publications. A *NameValueString* to perform such a command can be constructed as follows.

If the constants without blanks are used the blanks must be inserted:

```
MQCHAR DeleteCmd[] =
  MQPS_COMMAND " " MQPS_DELETE_PUBLICATION " " MQPS_TOPIC " *";
```

This can be simplified by using the constants with blanks:

```
MQCHAR DeleteCmd[] =
  MQPS_COMMAND_B MQPS_DELETE_PUBLICATION_B MQPS_TOPIC_B "*";
```

A subscribing application might need to analyze a *NameValueString*, for instance to determine the topic associated with each publication it receives. One approach is to break down the entire *NameValueString* into its constituent parts. An illustration of this approach is given in the results service sample application (see Chapter 9, "Sample programs" on page 101). A simpler approach is to use the `sscanf` in the C runtime library to determine the position of the `MQPSTopic` tag-name in the string. Since `sscanf` automatically strips away whitespace, the MQPS_TOPIC constant (without the blanks) is needed here.

Options using string constants

Some commands have options associated with them, which are also specified to the broker by name/value pairs. They are defined in the C header file **cmqpsc.h**. Multiple registration options, publication options and delete options are allowed, so the MQPSRegOpts, MQPSPubOpts and MQPSDelOpts tag-names can be repeated with different values. The effect is cumulative.

For example, to register an anonymous local publisher on topic 'News', the following *NameValueString* is needed:

```
MQPSCommand  RegPub
MQPSRegOpts  Anon
MQPSRegOpts  Local
MQPSTopic    News
```

Options using integer constants

Alternatively, an application can specify all of its options using a single name/value pair. This might be useful when the presence or absence of an option is conditional upon program logic. In this case, the combined set of options can be specified as a single decimal numeric value. The C header file **cmqcfc.h** provides corresponding integer constants for all of the options. In the previous example, the constants MQREGO_ANONYMOUS and MQREGO_LOCAL are relevant. The anonymous option has a decimal value of 2, and the local option has a decimal value of 4, so the following *NameValueString* is equivalent:

```
MQPSCommand  RegPub
MQPSRegOpts  6
MQPSTopic    News
```

Sending a command message with the RFH structure

Figure 15 shows how the RFH structure (including the *NameValueString*) is appended to the Message Descriptor in order to send a message to a broker. In this case, the message is to register a subscriber to the topic "IBM Stock Price". Part of the message descriptor is shown, together with the message data which consists of the RFH structure. The NameValueString should be padded to a multiple of four bytes.

Details of the name/value pairs for all the command messages are given in Chapter 7, “Publish/Subscribe command messages” on page 71.

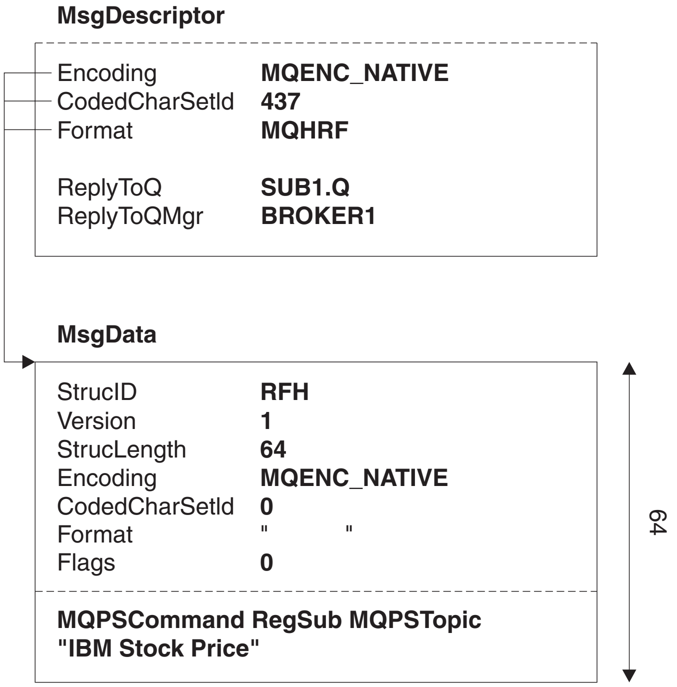

MsgDescriptor
Encoding MQENC_NATIVE
CodedCharSetId 437
Format MQHRF
ReplyToQ SUB1.Q
ReplyToQMgr BROKER1
MsgData
StrucID RFH
Version 1
StrucLength 64
Encoding MQENC_NATIVE
CodedCharSetId 0
Format "   "
Flags 0
MQPSCommand RegSub MQPSTopic
"IBM Stock Price"
64

Figure 15. Message descriptor and RFH structure. The message descriptor indicates that the subscriber has nominated its subscriber queue to be the same as its reply queue. It also defines the encoding and CCSID of the RFH structure, which follows as the message data. The encoding and CCSID fields in the RFH structure are not set, as there is no data following the RFH structure (compare with Figure 16 on page 68). Note that the length of the RFH structure includes the NameValueString (which contains the name/value pairs defining the Register Subscriber command). The topic string is quoted as it contains significant blanks.

Publication data

Publication data, or *UserData*, can be appended to a **Publish** command message after the *NameValueString*. The format of the data is defined in the *Encoding*, *CodedCharSetId* and *Format* fields of the MQRFH header. Alternatively, publication data can be included within the *NameValueString*, by means of user defined name/value pairs (which must not begin with the characters ‘MQ’), or the system provided *StringData* and *IntegerData* tags. More details are given in “Publish” on page 78.

Figure 16 shows how publication data can be appended to the RFH structure. Note how the encoding, CCSID and format of the publication data are defined in the RFH structure. In Figure 17 on page 69 the publication data is included within the *NameValueString*, and in Figure 18 on page 69, the format of the publication data is defined by the user.

Double-byte character sets

Publication data can use a single-byte character set (SBCS) or a double-byte character set (DBCS) codepage. However, if a publishing application publishes information in SBCS, then a subscribing application receiving that information must not request the data to be converted to DBCS (because the MQRFH header would be converted as well, and the header must be SBCS).

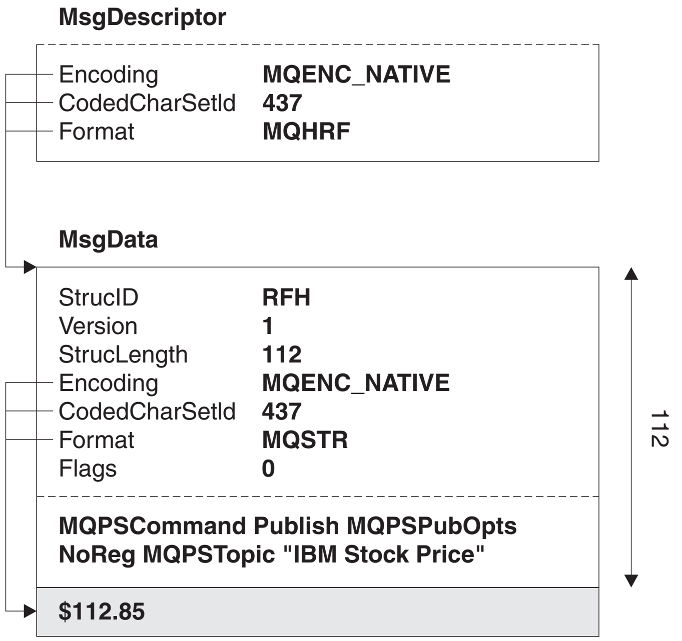

MsgDescriptor
Encoding
MQENC_NATIVE
CodedCharSetId
437
Format
MQHRF
MsgData
StrucID
RFH
Version
1
StrucLength
112
Encoding
MQENC_NATIVE
CodedCharSetId
437
Format
MQSTR
Flags
0
MQPSCommand Publish MQPSPubOpts NoReg MQPSTopic "IBM Stock Price"
112
$112.85

Figure 16. Publication data after the RFH structure. In this example, the publication data ($112.85) which is being published as string data in MQSTR format, is appended to the message after the NameValueString. Note that the RFH StrucLength includes the NameValueString, but not the publication data. The message descriptor defines the encoding, CCSID and format of the RFH structure, which in turn defines the encoding, CCSID and format of the publication data.

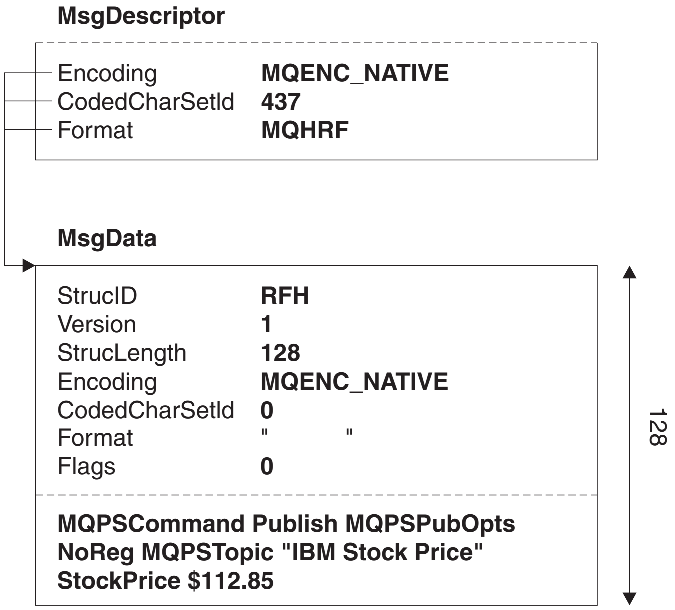
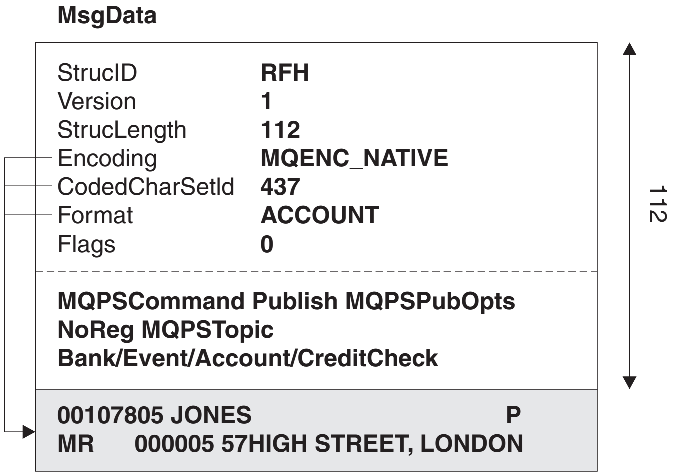

MsgDescriptor
Encoding MQENC_NATIVE
CodedCharSetId 437
Format MQHRF
MsgData
StrucID RFH
Version 1
StrucLength 128
Encoding MQENC_NATIVE
CodedCharSetId 0
Format "   "
Flags 0
MQPSCommand Publish MQPSPubOpts NoReg MQPSTopic "IBM Stock Price" StockPrice $112.85
128

Figure 17. Publishing data within the NameValueString. Publication data can be included within the NameValueString, by means of one or more user-defined name/value pairs, as shown in this example. The encoding and CCSID fields in the RFH structure are not set, as there is no following data. The receiving application must parse the RFH structure to extract the publication data.


MsgData
StrucID RFH
Version 1
StrucLength 112
Encoding MQENC_NATIVE
CodedCharSetId 437
Format ACCOUNT
Flags 0
MQPSCommand Publish MQPSPubOpts NoReg MQPSTopic Bank/Event/Account/CreditCheck
00107805 JONES P
MR 000005 57HIGH STREET, LONDON
112

```
struct {   MQLOMG         AccountNo;
           MQCHAR         Customer[32];
           MQLONG         CreditRating;
           MQCHAR         Address[24] }
```

Figure 18. User-defined publication data. In this example, the format of the publication data is set to a user-defined format, ACCOUNT, which contains character and numeric data. When the broker processes Publish messages, it converts the RFH header (but not the publication data) to its own CCSID and encoding. The user must write a data conversion routine if the publication is sent to subscribing applications which use a different CCSID or encoding.

In the previous examples, it is assumed that the subscribing or publishing application is running in an explicit codepage of 437. However, for reasons of portability, applications can use the special CCSID value MQCCSI_Q_MGR in the message descriptor if they are using the same codepage as the queue manager they are communicating with. In addition, the special value MQCCSI_INHERIT can be set in the CCSID field of the RF header to indicate that the publication data is in the same CCSID as the character data in the header.

Figure 19 shows how the CCSID for the RF header and the publication data can be inherited from the message descriptor.

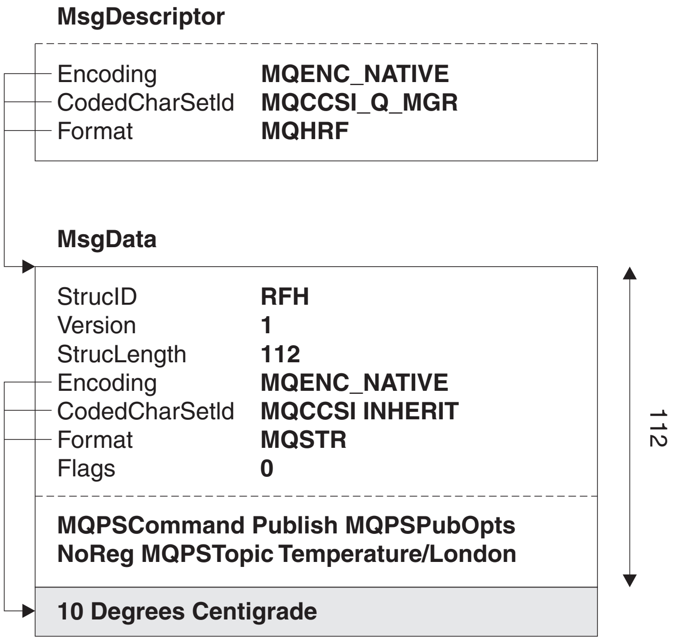

MsgDescriptor
Encoding MQENC_NATIVE
CodedCharSetId MQCCSI_Q_MGR
Format MQHRF
MsgData
StrucID RFH
Version 1
StrucLength 112
Encoding MQENC_NATIVE
CodedCharSetId MQCCSI INHERIT
Format MQSTR
Flags 0
MQPSCommand Publish MQPSPubOpts
NoReg MQPSTopic Temperature/London
10 Degrees Centigrade
112

Figure 19. Inheriting the CCSID. The message descriptor uses the special value MQCCSI_Q_MGR to indicate that data within the RFH structure is in the same CCSID as the queue manager. The value of MQCCSI_INHERIT in the RFH structure indicates that the same CCSID will be used for the publication data.

Chapter 7. Publish/Subscribe command messages

This chapter describes the name/value pairs that define the parameters needed for the following command messages:

- "Delete Publication" on page 72
- "Deregister Publisher" on page 74
- "Deregister Subscriber" on page 76
- "Publish" on page 78
- "Register Publisher" on page 83
- "Register Subscriber" on page 86
- "Request Update" on page 89

Chapter 6, "Format of command messages" on page 61 describes how to send these command messages using the Rules and Formatting header.

If you are using the MQSeries Application Messaging Interface (AMI) to communicate with the broker, you don't need to understand all the information in this chapter. The AMI constructs and interprets the RF Header and its name/value pairs (see "Using the Application Messaging Interface" on page 47). However, you might find it useful to read this chapter to see what options are available in each command message. Some of the options are directly accessible through parameters in an AMI function such as **amPublish**. Others can be accessed using an AMI name/value element helper function such as **amMsgGetElement**, or a macro such as **AmMsgGetStreamName**.

© Copyright IBM Corp. 1998, 2000

Delete Publication

The **Delete Publication** command message is sent from a publisher (or another broker) to a broker's stream queue to tell it to delete its copy of any retained publications for the specified topics within that stream.

Required parameters:
Command, Topic

Optional parameters:
DeleteOptions, StreamName

Required parameters

Command

name: "MQPSCommand" (string constant: MQPS_COMMAND)
value: "DeletePub" (string constant: MQPS_DELETE_PUBLICATION)

The command tag must be the first one in the *NameValueString*.

Topic

name: "MQPSTopic" (string constant: MQPS_TOPIC)
value: The topic for which published information is to be deleted. Wildcards can be used to delete several topics.

The topic tag can be repeated for as many topics as required.

Optional parameters

DeleteOptions

name: "MQPSDelOpts" (string constant: MQPS_DELETE_OPTIONS)
value: The following delete options can be specified:

"Local" (string constant: MQPS_LOCAL, integer constant: MQDELO_LOCAL).

Retained publications published locally at this broker (that is, with RetainPub and Local specified) will be deleted. Those published globally (that is, with RetainPub but not Local specified) will not be deleted, even if they were published at this broker.

The default if this tag is omitted is that global retained publications will be deleted at all brokers in the network, but local retained publications will not be deleted. Mixing local and global publications to the same topic and stream is not recommended. See “Publish” on page 78 for more information about retained local publications.

StreamName

name: "MQPSStreamName" (string constant: MQPS_STREAM_NAME)
value: The name of the publication stream for the specified *Topic(s)*.

The default value is the name of the stream queue to which the message is sent.

Example

Here is an example of a *NameValueString* for a **Delete Publication** command message. This is used by the sample application to delete the retained publication that contains the latest score in the match between Team1 and Team2.

```
MQPSCommand     DeletePub
MQPSStreamName  SAMPLE.BROKER.RESULTS.STREAM
MQPSTopic       "Sport/Soccer/State/LatestScore/Team1 Team2"
```

Error codes

The following reason codes might be returned in the *NameValueString* of the broker response message to this command, in addition to those shown on page 97.

Reason	Reason text	Explanation
3071	MQRCCF_STREAM_ERROR	Stream name too long or contains invalid characters.
3072	MQRCCF_TOPIC_ERROR	Topic name has an invalid length or contains invalid characters.
3075	MQRCCF_INCORRECT_STREAM	Stream name does not match stream queue.
3087	MQRCCF_DEL_OPTIONS_ERROR	Invalid delete options supplied.

Deregister Publisher

The **Deregister Publisher** command message is sent from a publisher, or another application on a publisher's behalf, to a broker's control queue to indicate that a publisher will no longer be publishing data on the topics contained in the message.

Required parameters:
Command

Optional parameters:
RegistrationOptions, StreamName, Topic, QMgrName, QName

Required parameters

Command

name: "MQPSCommand" (string constant: MQPS_COMMAND)
value: "DeregPub" (string constant: MQPS_DEREGISTER_PUBLISHER)

The command tag must be the first one in the *NameValueString*.

Optional parameters

RegistrationOptions

name: "MQPSRegOpts" (string constant: MQPS_REGISTRATION_OPTIONS)
value: The following registration options can be specified:

"DeregAll" (string constant: MQPS_DEREGISTER_ALL, integer constant: MQREGO_DEREGISTER_ALL)

All topics registered for this publisher are to be deregistered. If this option is set, the *Topic* parameter must be omitted.

"CorrelAsId" (string constant: MQPS_CORREL_ID_AS_IDENTITY, integer constant: MQREGO_CORREL_ID_AS_IDENTITY).

The *CorrelId* in the MQMD (which must not be zero) is part of the publisher's identity.

The default if this tag is omitted is that no options are set. In this case, the *Topic* parameter is required.

StreamName

name: "MQPSStreamName" (string constant: MQPS_STREAM_NAME)
value: The name of the publication stream for the specified *Topic(s)*.

The default value is SYSTEM.BROKER.DEFAULT.STREAM.

Topic

name: "MQPSTopic" (string constant: MQPS_TOPIC)
value: The topic being deregistered. Wildcards are allowed.

If DeregAll is specified in *RegistrationOptions*, the *Topic* tag must be omitted. Otherwise, it is required, and can optionally be repeated for as many topics as needed.

QueueManagerName

name: `"MQPSQMgrName"` (string constant: MQPS_Q_MGR_NAME)
value: The publisher's queue manager name.

For a message sent by a publisher, if this tag is not present it defaults to the *ReplyToQMgr* name in the message descriptor (MQMD). If the resulting name is blank, it matches a publisher that registered with a blank queue manager name.

For a message sent by a broker, this tag is omitted.

QueueName

name: `"MQPSQName"` (string constant: MQPS_Q_NAME)
value: The publisher's queue name.

For a message sent by a publisher, if this tag is not present, it defaults to the *ReplyToQ* name in the message descriptor (MQMD), which must not be blank in this case.

For a message sent by a broker, this tag is omitted.

Example

Here is an example of a *NameValueString* for a **Deregister Publisher** command message. This will deregister a publisher for all topics it has registered that match `Stock/*`. The publisher's identity, including the *CorrelId*, is taken from the defaults in the MQMD.

```
MQPSCommand  DeregPub
MQPSRegOpts  CorrelAsId
MQPSTopic    Stock/*
```

Error codes

The following reason codes might be returned in the *NameValueString* of the broker response message to this command, in addition to those shown on page 97.

Reason	Reason text	Explanation
3071	MQRCCF_STREAM_ERROR	Stream name too long or contains invalid characters.
3072	MQRCCF_TOPIC_ERROR	Topic name has an invalid length or contains invalid characters.
3073	MQRCCF_NOT_REGISTERED	Publisher or subscriber not registered.
3074	MQRCCF_Q_MGR_NAME_ERROR	Queue manager name invalid.
3076	MQRCCF_Q_NAME_ERROR	Queue name invalid.
3078	MQRCCF_DUPLICATE_IDENTITY	Publisher or subscriber identity already assigned to another user ID.
3080	MQRCCF_CORREL_ID_ERROR	Correlation identifier used as part of identity but is all binary zero.
3082	MQRCCF_UNKNOWN_STREAM	Stream not defined to broker and cannot be created.
3083	MQRCCF_REG_OPTIONS_ERROR	Invalid registration options supplied.

Deregister Subscriber

The **Deregister Subscriber** command message is sent from a subscriber, another application on a subscriber's behalf, or another broker, to a broker's control queue to indicate that it no longer wishes to subscribe to the topics specified.

Required parameters:
Command

Optional parameters:
RegistrationOptions, StreamName, Topic, QMgrName, QName

Required parameters

Command

name: `"MQPSCommand"` (string constant: MQPS_COMMAND)
value: `"DeregSub"` (string constant: MQPS_DEREGISTER_SUBSCRIBER)

The command tag must be the first one in the *NameValueString*.

Optional parameters

RegistrationOptions

name: `"MQPSRegOpts"` (string constant: MQPS_REGISTRATION_OPTIONS)
value: The following registration options can be specified:

`"DeregAll"` (string constant: MQPS_DEREGISTER_ALL, integer constant: MQREGO_DEREGISTER_ALL).

All topics registered for this subscriber are to be deregistered. If this option is set, the *Topic* parameter must be omitted.

`"CorrelAsId"` (string constant: MQPS_CORREL_ID_AS_IDENTITY, integer constant: MQREGO_CORREL_ID_AS_IDENTITY).

The *CorrelId* in the MQMD (which must not be zero) is part of the subscriber's identity.

The default if this tag is omitted is that no options are set. In this case, the *Topic* parameter is required.

StreamName

name: `"MQPSStreamName"` (string constant: MQPS_STREAM_NAME)
value: The name of the publication stream for the specified *Topic(s)*.

The default value is SYSTEM.BROKER.DEFAULT.STREAM.

Topic

name: `"MQPSTopic"` (string constant: MQPS_TOPIC)
value: The topic being deregistered. Wildcards are allowed, but a specified topic string must match exactly the corresponding string that was originally specified in the **Register Subscriber** command.

If `DeregAll` is specified in *RegistrationOptions*, the *Topic* tag must be omitted. Otherwise, it is required, and can optionally be repeated for as many topics as needed. Topics specified can be a subset of those for which the subscriber is registered if it wishes to retain subscriptions to the other topics.

QueueManagerName

name: `"MQPSQMgrName"` (string constant: MQPS_Q_MGR_NAME)
value: The subscriber's queue manager name.

If this tag is not present it defaults to the *ReplyToQMgr* name in the message descriptor (MQMD). If the resulting name is blank, it matches a subscriber that registered with a blank queue manager name.

QueueName

name: `"MQPSQName"` (string constant: MQPS_Q_NAME)
value: The subscriber's queue name.

If this tag is not present, it defaults to the *ReplyToQ* name in the message descriptor (MQMD), which must not be blank in this case.

Example

Here is an example of a *NameValueString* for a **Deregister Subscriber** command message. In this case the sample application is deregistering its subscription to the topics which contain the latest score for all matches. The subscriber’s identity, including the *CorrelId*, is taken from the defaults in the MQMD.

```
MQPSCommand     DeregSub
MQPSRegOpts     CorrelAsId
MQPSStreamName  SAMPLE.BROKER.RESULTS.STREAM
MQPSTopic       Sport/Soccer/State/LatestScore/*
```

Error codes

The following reason codes might be returned in the *NameValueString* of the broker response message to this command, in addition to those shown on page 97.

Reason	Reason text	Explanation
3071	MQRCCF_STREAM_ERROR	Stream name too long or contains invalid characters.
3072	MQRCCF_TOPIC_ERROR	Topic name has an invalid length or contains invalid characters.
3073	MQRCCF_NOT_REGISTERED	Publisher or subscriber not registered.
3074	MQRCCF_Q_MGR_NAME_ERROR	Queue manager name invalid.
3076	MQRCCF_Q_NAME_ERROR	Queue name invalid.
3078	MQRCCF_DUPLICATE_IDENTITY	Publisher or subscriber identity already assigned to another user ID.
3080	MQRCCF_CORREL_ID_ERROR	Correlation identifier used as part of identity but is all binary zero.
3082	MQRCCF_UNKNOWN_STREAM	Stream not defined to broker and cannot be created.
3083	MQRCCF_REG_OPTIONS_ERROR	Invalid registration options supplied.

Publish

The **Publish** command message is sent:

- From a publisher (or another broker) to a broker's stream queue
- From a broker to a subscriber's stream queue

to publish information on specific topics.

Publication data can be appended to the message, after the *NameValueString*, in a format defined by the *Encoding*, *CodedCharSetId* and *Format* fields in the MQRFH header.

Alternatively, publication data can be included within the *NameValueString*, using name/value pairs such as the *StringData* and *IntegerData* parameters defined below, or any other name/value pairs defined by the publisher (provided the tag-name does not begin with the characters 'MQ').

Required parameters:
Command, *Topic*

Optional parameters:
RegistrationOptions, *PublicationOptions*, *StreamName*, *QMgrName*, *QName*, *PublishTimestamp*, *SequenceNumber*, *StringData*, *IntegerData*

Required parameters

Command

name: `"MQPSCommand"` (string constant: MQPS_COMMAND)
value: `"Publish"` (string constant: MQPS_PUBLISH)

The command tag must be the first one in the *NameValueString*.

Topic

name: `"MQPSTopic"` (string constant: MQPS_TOPIC)
value: The topic that categorizes this publication. No wildcards are allowed.

This tag can be repeated for as many topics as required. For example, an application might publish information under topic 'Topic 1', which is then enhanced to publish extra information. The new publications might use topics 'Topic 1' and 'Topic 1 enhanced', so that subscribers to 'Topic 1 enhanced' would be sure to get the additional information, while existing subscribers to 'Topic 1' could still access the basic information in the same publication.

Optional parameters

RegistrationOptions

name: `"MQPSRegOpts"` (string constant: MQPS_REGISTRATION_OPTIONS)
value: The registration options listed below can be specified, subject to the following conditions:

If `NoReg` is not specified in *PublicationOptions*:

- If the publisher is already registered, the registration options are changed to the values specified, if this tag is present. If it is not present the registration options are unchanged.

- If the publisher is not already registered, an implicit registration is performed. The registration options are those specified by this tag, if it is present. If it is not present, no options are set.

If `NoReg` is specified in *PublicationOptions*, any current registration has no effect and it is not changed. *RegistrationOptions* can be specified. If `Local` is specified in *RegistrationOptions*, the publication is restricted to local subscribers and any other valid options are not acted on by the broker.

The following can be set:

`"Anon"` (string constant: MQPS_ANONYMOUS, integer constant: MQREGO_ANONYMOUS).

(Valid only if the recipient is a broker.) This option tells the broker that the identity of the publisher is not to be divulged, except to subscribers with additional authority.

This option (or the lack of it) overrides the option setting for any previous publication on the same topics (or publisher registration).

`"Local"` (string constant: MQPS_LOCAL, integer constant: MQREGO_LOCAL).

(Valid only if the recipient is a broker.) This option tells the broker that publications published by this publisher should only be sent to subscribers that registered at this broker specifying `Local`.

`"DirectReq"` (string constant: MQPS_DIRECT_REQUESTS, integer constant: MQREGO_DIRECT_REQUESTS).

This option tells the recipient that the publisher is willing to receive direct requests for publication information from other applications (not just from the broker).

The publisher's queue and queue manager names can be included in a **Publish** message sent by a publisher, so that the names are visible to the subscriber.

This option (or the lack of it) overrides the option setting for any previous publication on the same topics (or registration in the case of a publisher to a broker, or the value returned in the response to a subscriber registration).

This option must not be set if `Anon` is also set.

`"CorrelAsId"` (string constant: MQPS_CORREL_ID_AS_IDENTITY, integer constant: MQREGO_CORREL_ID_AS_IDENTITY).

The *CorrelId* in the MQMD (which must not be zero) is part of the publisher's identity. This option is assumed if `CorrelAsId` is set in the *PublicationOptions*.

PublicationOptions

name: `"MQPSPubOpts"` (string constant: MQPS_PUBLICATION_OPTIONS)
value: The following publication options can be specified:

`"NoReg"` (string constant: MQPS_NO_REGISTRATION, integer constant: MQPUBO_NO_REGISTRATION).

(Valid only if the recipient is a broker.) If the publisher is not already registered with the broker as a publisher for this stream and topic, this option stops the broker from performing an implicit registration. If the publisher is already registered, the registration is unchanged, and has no effect on this publication.

`"RetainPub"` (string constant: MQPS_RETAIN_PUBLICATION, integer constant: MQPUBO_RETAIN_PUBLICATION).

(Valid only if the recipient is a broker.) The broker is to retain a copy of the publication. If this option is not set, the publication is deleted as soon as the broker has sent the publication to all of its current subscribers.

`"IsRetainedPub"` (string constant: MQPS_IS_RETAINED_PUBLICATION, integer constant: MQPUBO_IS_RETAINED_PUBLICATION).

(Can only be set by a broker.) This publication has been retained by the broker. The broker sets this option to notify a subscriber that this publication was published earlier and has been retained. A subscriber can receive such a publication immediately after registering (or later if a publication has been retained at another broker that is temporarily inaccessible). It can also be received in response to a **Request Update** command.

The broker sets this option only if the subscriber registered with the `InformIfRet` option.

`"OtherSubsOnly"` (string constant: MQPS_OTHER_SUBSCRIBERS_ONLY, integer constant: MQPUBO_OTHER_SUBSCRIBERS_ONLY).

(Valid only if the recipient is a broker.) This option allows simpler processing of conference-type applications. It tells the broker not to send the publication to the publisher even if he has subscribed. For example, a group of applications can all subscribe to the same topic (for example, “Conference”). Using this option, each application can publish information into the conference without themselves receiving the information.

`"CorrelAsId"` (string constant: MQPS_CORREL_ID_AS_IDENTITY, integer constant: MQPUBO_CORREL_ID_AS_IDENTITY).

The *CorrelId* in the MQMD (which must not be zero) is part of the publisher’s identity (for messages sent by a publisher to a broker). For messages sent from a broker to a subscriber, this option is not changed by the broker.

The default is that no publication options are set.

StreamName

name: `"MQPSStreamName"` (string constant: MQPS_STREAM_NAME)
value: The name of the publication stream for the specified *Topic(s)*.

This defaults to the name of the stream queue to which the message is sent if sent to a broker, or an unspecified stream name if the message is sent to a subscriber (note that a subscriber can request that the broker always include *StreamName* in **Publish** messages by specifying `"InclStreamName"` when it registers).

QMgrName

name: `"MQPSQMgrName"` (string constant: MQPS_Q_MGR_NAME)
value: The publisher's queue manager name.

For a message sent by a publisher, the default is the *ReplyToQMgr* name in the message descriptor (MQMD). If the resulting name is blank, it represents a publisher that can be reached by resolving *QName* at the broker.

For a message sent by a broker, this tag is present if and only if it was explicitly included by the publisher. (Note that it is not removed by the broker if the publisher has registered with `Anon`)

QName

name: `"MQPSQName"` (string constant: MQPS_Q_NAME)
value: The publisher's queue name.

For a message sent by a publisher, the default is the *ReplyToQ* name in the message descriptor (MQMD), which must not be blank in this case (unless *PublicationOptions* specifies `NoReg` and not `OtherSubsOnly`).

For a message sent by a broker, this tag is present if and only if it was explicitly included by the publisher. (Note that it is not removed by the broker if the publisher has registered with `Anon`)

PublishTimestamp

name: `"MQPSQName"` (string constant: MQPS_Q_NAME)
value: Optional publication timestamp set by the publisher.

This is of length 16 characters in the format:

```
YYYYMMDDHHMMSSTH
```

using Universal Time. However, this is not checked by the broker, which merely transmits this information to subscribers if it is present.

SequenceNumber

name: `"MQPSSeqNum"` (string constant: MQPS_SEQUENCE_NUMBER)
value: Optional sequence number set by the publisher.

This should increase by 1 with each publication. However, this is not checked by the broker, which merely transmits this information to subscribers if it is present. If publications on the same stream and topic are published to different interconnected brokers, it is the responsibility of the publisher to ensure that sequence numbers, if used, are meaningful.

StringData

name: `"MQPSStringData"` (string constant: MQPS_STRING_DATA)
value: Optional publication data as a character string.

The meaning and format are as defined by the publisher. This tag can be repeated, interspersed with *IntegerData* tags if required, to send publication data in any manner defined by the publisher.

IntegerData

name: `"MQPSIntData"` (string constant: MQPS_INTEGER_DATA)
value: Optional publication data as an integer.

The meaning is as defined by the publisher. This tag can be repeated, interspersed with *StringData* tags if required, to send publication data in any manner defined by the publisher.

Example

Here are some examples of a *NameValueString* for a **Publish** command message. The first example is for an Event Publication sent by the match simulator in the sample application to indicate that a match has started, with 'No Registration' specified for the publisher:

```
MQPSCommand     Publish
MQPSPubOpts     NoReg
MQPSStreamName  SAMPLE.BROKER.RESULTS.STREAM
MQPSTopic       Sport/Soccer/Event/MatchStarted
```

The second example is for a State Publication, so 'Retain Publication' is specified as well. In this case the results service is publishing the latest score in the match between Team1 and Team2.

```
MQPSCommand     Publish
MQPSPubOpts     RetainPub
MQPSPubOpts     NoReg
MQPSStreamName  SAMPLE.BROKER.RESULTS.STREAM
MQPSTopic       "Sport/Soccer/State/LatestScore/Team1 Team2"
```

In both examples the publication data (the names of the teams, or the latest score) follows the *NameValueString*, as string data in MQSTR format.

Error codes

The following reason codes might be returned in the *NameValueString* of the broker response message to this command, in addition to those shown on page 97.

Reason	Reason text	Explanation
3071	MQRCCF_STREAM_ERROR	Stream name too long or contains invalid characters.
3072	MQRCCF_TOPIC_ERROR	Topic name has an invalid length or contains invalid characters.
3074	MQRCCF_Q_MGR_NAME_ERROR	Queue manager name invalid.
3075	MQRCCF_INCORRECT_STREAM	Stream not defined to broker and cannot be created.
3076	MQRCCF_Q_NAME_ERROR	Queue name invalid.
3078	MQRCCF_DUPLICATE_IDENTITY	Publisher or subscriber identity already assigned to another user ID.
3080	MQRCCF_CORREL_ID_ERROR	Correlation identifier used as part of identity but is all binary zero.
3083	MQRCCF_REG_OPTIONS_ERROR	Invalid registration options supplied.
3084	MQRCCF_PUB_OPTIONS_ERROR	Invalid publication options supplied.

Register Publisher

The **Register Publisher** command message is sent from a publisher (or another application on a publisher's behalf) to a broker's control queue to indicate that a publisher will be, or is capable of, publishing data on one or more specified topics.

Required parameters:
Command, *Topic*

Optional parameters:
RegistrationOptions, *StreamName*, *QMgrName*, *QName*

Required parameters

Command

name: `"MQPSCommand"` (string constant: MQPS_COMMAND)
value: `"RegPub"` (string constant: MQPS_REGISTER_PUBLISHER)

The command tag must be the first one in the *NameValueString*.

Topic

Topic

name: `"MQPSTopic"` (string constant: MQPS_TOPIC)
value: The topic for which the publisher will be providing publications. Wildcards are not allowed.

This tag can be repeated for as many topics as required.

Optional parameters

RegistrationOptions

name: `"MQPSRegOpts"` (string constant: MQPS_REGISTRATION_OPTIONS)
value: The following registration options can be specified:

`"Anon"` (string constant: MQPS_ANONYMOUS, integer constant: MQREGO_ANONYMOUS)

This option tells the broker that the identity of the publisher is not to be divulged, except to subscribers with additional authority.

`"Local"` (string constant: MQPS_LOCAL, integer constant: MQREGO_LOCAL)

This option tells the broker that publications published by this publisher should only be sent to subscribers that registered on this broker specifying `Local`.

`"DirectReq"` (string constant: MQPS_DIRECT_REQUESTS, integer constant: MQREGO_DIRECT_REQUEST)

This option tells the recipient that the publisher is willing to receive direct requests for publication information from other applications (that is, not just from the broker).

This option must not be set if `Anon` is also set.

`"CorrelAsId"` (string constant: MQPS_CORREL_ID_AS_IDENTITY, integer constant: MQREGO_CORREL_ID_AS_IDENTITY)

The *CorrelId* in the message descriptor, MQMD, (which must not be zero) is part of the publisher's identity.

If the *RegistrationOptions* tag is omitted and the publisher is already registered, its registration options are unchanged. If the publisher is not already registered, the default is that no registration options are set.

StreamName

name: `"MQPSStreamName"` (string constant: MQPS_STREAM_NAME)
value: The name of the publication stream for the specified *Topic(s)*.

The default value is SYSTEM.BROKER.DEFAULT.STREAM.

QMgrName

name: `"MQPSQMgrName"` (string constant: MQPS_Q_MGR_NAME)
value: The publisher's queue manager name.

For a message sent by a publisher, the default is the *ReplyToQMgr* name in the message descriptor (MQMD). If the resulting name is blank, it represents a publisher that can be reached by resolving *QName* at the broker.

For a message sent by a broker, this tag is present only if `DirectReq` is set in the *RegistrationOptions* tag.

QName

name: `"MQPSQName"` (string constant: MQPS_Q_NAME)
value: The publisher's queue name.

For a message sent by a publisher, the default is the *ReplyToQ* name in the message descriptor (MQMD), which must not be blank in this case.

For a message sent by a broker, this tag is present only if `DirectReq` is set in the *RegistrationOptions* tag.

Example

Here is an example of a *NameValueString* for a **Register Publisher** command message. The publisher is registering with the 'Direct Requests' option, for the `Stock/IBM` topic on the default stream. The queue name and queue manager name are specified so that subscribers can respond directly to the publisher.

```
MQPSCommand   RegPub
MQPSRegOpts   DirectReq
MQPSQMgrName  Broker1
MQPSQName     STOCK.IBM.PUBLISHER.QUEUE
MQPSTopic     Stock/IBM
```

Error codes

The following reason codes might be returned in the *NameValueString* of the broker response message to this command, in addition to those shown on page 97.

Reason	Reason text	Explanation
3071	MQRCCF_STREAM_ERROR	Stream name too long or contains invalid characters.
3072	MQRCCF_TOPIC_ERROR	Topic name has an invalid length or contains invalid characters.
3074	MQRCCF_Q_MGR_NAME_ERROR	Queue manager name invalid.
3076	MQRCCF_Q_NAME_ERROR	Queue name invalid.
3078	MQRCCF_DUPLICATE_IDENTITY	Publisher or subscriber identity already assigned to another user ID.
3080	MQRCCF_CORREL_ID_ERROR	Correlation identifier used as part of identity but is all binary zero.
3082	MQRCCF_UNKNOWN_STREAM	Stream not defined to broker and cannot be created.
3083	MQRCCF_REG_OPTIONS_ERROR	Invalid registration options supplied.

Register Subscriber

The **Register Subscriber** command message is sent from a subscriber (or another application on its behalf), or a broker, to a broker's control queue to indicate that it wishes to subscribe to the topics specified.

Required parameters:
Command, *Topic*

Optional parameters:
RegistrationOptions, *StreamName*, *QMgrName*, *QName*

Required parameters

Command

name: `"MQPSCommand"` (string constant: MQPS_COMMAND)
value: `"RegSub"` (string constant: MQPS_REGISTER_SUBSCRIBER)

The command tag must be the first one in the *NameValueString*.

Topic

name: `"MQPSTopic"` (string constant: MQPS_TOPIC)
value: The topic for which the subscriber wants to receive publications. Wildcards are allowed.

This tag can be repeated for as many topics as required.

Optional parameters

RegistrationOptions

name: `"MQPSRegOpts"` (string constant: MQPS_REGISTRATION_OPTIONS)
value: The following registration options can be specified:

`"Anon"` (string constant: MQPS_ANONYMOUS, integer constant: MQREGO_ANONYMOUS)

This option tells the broker that the identity of the publisher is not to be divulged, except to subscribers with additional authority.

`"Local"` (string constant: MQPS_LOCAL, integer constant: MQREGO_LOCAL)

This option tells the broker that the subscription is local and should not be distributed to other brokers in the network. Only publications published at this node by a publisher specifying `Local` will be sent to this subscriber.

`"NewPubsOnly"` (string constant: MQPS_NEW_PUBLICATIONS_ONLY, integer constant: MQREGO_NEW_PUBLICATIONS_ONLY)

This option tells the broker that no currently retained publications are to be sent, only new publications. If a subscriber re-registers and changes this option so that it is not set, it is possible that a publication that has already been sent to it will be sent to it again.

`"PubOnReqOnly"` (string constant: MQPS_PUBLISH_ON_REQUEST_ONLY, integer constant: MQREGO_PUBLISH_ON_REQUEST_ONLY)

This option indicates that the subscriber will only poll for information with **Request Update**. The broker is not to send unsolicited messages to the subscriber.

This option is not propagated if the broker sends this subscription to other brokers in the network. Publications will be sent to it in the normal way, and these publications should specify `RetainPub` in order to be eligible for return in response to a **Request Update** message.

`"CorrelAsId"` (string constant: MQPS_CORREL_ID_AS_IDENTITY, integer constant: MQREGO_CORREL_ID_AS_IDENTITY)

The *CorrelId* in the message descriptor, MQMD, (which must not be zero) is part of the subscriber's identity.

`"InclStreamName"` (string constant: MQPS_INCLUDE_STREAM_NAME, integer constant: MQREGO_INCLUDE_STREAM_NAME)

Each **Publish** message that is sent must include the *StreamName* parameter. The broker does this by adding the appropriate name/value pair to the *NameValueString* of the message. The *NameValueString* will be extended if necessary.

If this option is not set, *StreamName* is included only if it was specified explicitly by the publisher.

`"InformIfRet"` (string constant: MQPS_INFORM_IF_RETAINED, integer constant: MQREGO_INFORM_IF_RETAINED)

The broker will inform the subscriber if a publication is retained when a **Publish** message is sent. It does this by adding the name/value pair `"MQPSPubOpts IsRetainedPub"` to the *NameValueString* of the message (after the *StreamName* if that has been added in accordance with the `InclStreamName` option).

Use this option if a subscriber needs to distinguish between new publications and old publications that were retained by the broker prior to the subscription being made. If this option is specified, the broker will always add the name/value pair to a publication sent in response to a **Request Update** command.

If the registration options tag is omitted and the subscriber is already registered, its registration options are unchanged. If the subscriber is not already registered, the default is that no registration options are set.

StreamName

name: `"MQPSStreamName"` (string constant: MQPS_STREAM_NAME)
value: The name of the publication stream for the specified *Topic(s)*.

The default value is SYSTEM.BROKER.DEFAULT.STREAM.

QMgrName

name: `"MQPSQMgrName"` (string constant: MQPS_Q_MGR_NAME)
value: The subscriber's queue manager name.

The default is the *ReplyToQMgr* name in the message descriptor (MQMD). If the resulting name is blank, it represents a publisher that can be reached by resolving *QName* at the broker.

QName

name: `"MQPSQName"` (string constant: MQPS_Q_NAME)
value: The subscriber's queue name.

The default is the *ReplyToQ* name in the message descriptor (MQMD), which must not be blank in this case.

Example

Here is an example of a *NameValueString* for a **Register Subscriber** command message. In the sample application, the results service uses this message to register a subscription to the topics containing the latest scores in all matches, with the 'Publish on Request Only' option set. The subscriber's identity, including the *CorrelId*, is taken from the defaults in the MQMD.

```
MQPSCommand     RegSub
MQPSRegOpts     PubOnReqOnly
MQPSRegOpts     CorrelAsId
MQPSStreamName  SAMPLE.BROKER.RESULTS.STREAM
MQPSTopic       Sport/Soccer/State/LatestScore/*
```

Here is the same message using the equivalent decimal registration options:

```
MQPSCommand     RegSub
MQPSRegOpts     33
MQPSStreamName  SAMPLE.BROKER.RESULTS.STREAM
MQPSTopic       Sport/Soccer/State/LatestScore/*
```

Error codes

The following reason codes might be returned in the *NameValueString* of the broker response message to this command, in addition to those shown on page 97.

Reason	Reason text	Explanation
3071	MQRCCF_STREAM_ERROR	Stream name too long or contains invalid characters.
3072	MQRCCF_TOPIC_ERROR	Topic name has an invalid length or contains invalid characters.
3074	MQRCCF_Q_MGR_NAME_ERROR	Queue manager name invalid.
3076	MQRCCF_Q_NAME_ERROR	Queue name invalid.
3078	MQRCCF_DUPLICATE_IDENTITY	Publisher or subscriber identity already assigned to another user ID.
3080	MQRCCF_CORREL_ID_ERROR	Correlation identifier used as part of identity but is all binary zero.
3081	MQRCCF_NOT_AUTHORIZED	Publisher or subscriber not registered.
3082	MQRCCF_UNKNOWN_STREAM	Stream not defined to broker and cannot be created.
3083	MQRCCF_REG_OPTIONS_ERROR	Invalid registration options supplied.

Request Update

The **Request Update** command message is sent from a subscriber to a broker to request an update publication for the topic specified. This is normally used if the subscriber specified the option `"PubOnReqOnly"` (publish on request only) when it registered. If the broker has a retained publication for the topic, this is sent to the subscriber. If not, the request fails.

Required parameters:
Command, *Topic*

Optional parameters:
RegistrationOptions, *StreamName*, *QMgrName*, *QName*

Required parameters

Command

name: `"MQPSCommand"` (string constant: MQPS_COMMAND)
value: `"ReqUpdate"` (string constant: MQPS_REQUEST_UPDATE)

The command tag must be the first one in the *NameValueString*.

Topic

name: `"MQPSTopic"` (string constant: MQPS_TOPIC)
value: The topic the subscriber is requesting. Wildcards are allowed, in which case the subscriber might receive multiple retained publications.

Only one occurrence of this tag is allowed in this message.

Optional parameters

RegistrationOptions

name: `"MQPSRegOpts"` (string constant: MQPS_REGISTRATION_OPTIONS)
value: The following registration options can be specified:

`"CorrelAsId"` (string constant: MQPS_CORREL_ID_AS_IDENTITY, integer constant: MQREGO_CORREL_ID_AS_IDENTITY)

The *CorrelId* in the message descriptor (MQMD), which must not be zero, is part of the subscriber's identity.

StreamName

name: `"MQPSStreamName"` (string constant: MQPS_STREAM_NAME)
value: The name of the publication stream for the specified *Topic(s)*.

The default value is SYSTEM.BROKER.DEFAULT.STREAM.

QMgrName

name: `"MQPSQMgrName"` (string constant: MQPS_Q_MGR_NAME)
value: The subscriber's queue manager name.

The default is the *ReplyToQMgr* name in the message descriptor (MQMD). If the resulting name is blank, it matches a publisher with a blank queue manager name (that is, local to the broker).

QName

name: `"MQPSQName"` (string constant: MQPS_Q_NAME)
value: The subscriber's queue name.

The default is the *ReplyToQ* name in the message descriptor (MQMD), which must not be blank in this case.

Example

Here is an example of a *NameValueString* for a **Request Update** command message. In the sample application, the results service uses this message to request retained publications containing the latest scores for all teams. The subscriber's identity, including the *CorrelId*, is taken from the defaults in the MQMD.

```
MQPSCommand     ReqUpdate
MQPSRegOpts     CorrelAsId
MQPSStreamName  SAMPLE.BROKER.RESULTS.STREAM
MQPSTopic       Sport/Soccer/State/LatestScore/*
```

Error codes

The following reason codes might be returned in the *NameValueString* of the broker response message to this command, in addition to those shown on page 97.

Reason	Reason text	Explanation
3071	MQRCCF_STREAM_ERROR	Stream name too long or contains invalid characters.
3072	MQRCCF_TOPIC_ERROR	Topic name has an invalid length or contains invalid characters.
3073	MQRCCF_NOT_REGISTERED	Publisher or subscriber not registered.
3074	MQRCCF_Q_MGR_NAME_ERROR	Queue manager name invalid.
3076	MQRCCF_Q_NAME_ERROR	Queue name invalid.
3077	MQRCCF_NO_RETAINED_MSG	No retained message exists for this topic.
3078	MQRCCF_DUPLICATE_IDENTITY	Publisher or subscriber identity already assigned to another user ID.
3080	MQRCCF_CORREL_ID_ERROR	Correlation identifier used as part of identity but is all binary zero.
3081	MQRCCF_NOT_AUTHORIZED	Subscriber not authorized to browse broker's stream queue or subscriber queue.
3082	MQRCCF_UNKNOWN_STREAM	Stream not defined to broker and cannot be created.
3083	MQRCCF_REG_OPTIONS_ERROR	Invalid registration options supplied.

Chapter 8. Error handling and response messages

Messages sent to and by a broker are subject to exception processing, report generation and dead-letter queue processing in the same way as other MQSeries messages. A message can indicate that a response is not required, is required only if there is an error, only if the command succeeds, or always required.

Response messages can be generated by the broker to each command message issued by a publisher or subscriber. Response messages indicate the success or failure of a request and also the reason for the failure. Responses are given only by the broker to which the messages are initially sent.

The following topics are discussed in this chapter:

- "Error handling by the broker"
- "Response messages" on page 92
- "Broker responses" on page 94
- "Problem determination" on page 98

Error handling by the broker

Any message received by a broker that is not of *Format* MQFMT_RF_HEADER (or MQFMT_PCF in the case of the system management messages described in Part 4, "System programming" on page 147) is treated as an error. It is written to the dead-letter queue (or discarded, depending on the report options), and an exception report generated, if requested. If a message is of the correct format but has some other error (for example, a syntax error), or if the broker is unable to process it correctly (for example, it is unable to retain a message), the following happens:

- If a response has been requested, one is generated.
  - If the response cannot be enqueued at the broker, the response is put to the dead-letter queue (responses are always generated with MQRO_NONE).
  - If the response cannot be put to the dead-letter queue, the response is discarded if this is allowed (this depends on the type of response message), depending on the broker configuration parameters.
  - If the response could not be discarded or put to the reply-to queue or the dead-letter queue, the command is backed out and the input message is put to the dead-letter queue with a *Reason* of MQRCCF_BROKER_COMMAND_FAILED, or discarded, as indicated by the report options. An exception report message is generated if requested.
  - If the input message or response cannot be put to the dead-letter queue or discarded, the command is backed out and the input message is restored to the input queue if the message is within syncpoint. The input message is retried periodically, and (less frequently) a message is written to the queue manager log to alert the administrator.
- If a response has not been requested, one is not sent, and no further action is appropriate for this message.

© Copyright IBM Corp. 1998, 2000

If an input message is put to the dead-letter queue, no response and publication messages will have been sent. It might be appropriate for the input message to be restored and reprocessed when the error has been resolved.

If the message is a **Publish** command message, and there is a problem sending an outgoing message on to a subscriber, the processing is as follows:

- The outgoing message is put to the dead-letter queue, if this is permitted by the broker and queue manager configuration. If the outgoing message cannot be put to the dead-letter queue because of a failure or because it is not permitted by the broker and queue manager configuration, it is discarded if this is permitted by the broker and queue manager configuration.
- If the outgoing message cannot be put to the dead-letter queue or discarded, the input message is restored. The input message is retried after suitable time interval, and (less frequently) a message is written to the log to alert the administrator.

Note: If the broker cannot put a publication message onto a destination queue or the dead-letter queue and cannot discard the message, the broker will continue trying to put the publication message onto the destination queue (at suitable intervals) and will not continue processing subsequent messages.

The dead-letter queue and discard options for nonpersistent messages are specified in queue manager configuration file (qm.ini or equivalent). These options are described in Chapter 10, “Setting up a broker” on page 109.

Response messages

Each command message that the broker processes can generate a response message. A response message has a similar format to a command message; the *NameValueString* in the MQRFH header contains the response to the command. Response messages are sent to the queue identified by the *ReplyToQ* and *ReplyToQMgr* fields in the message descriptor of the original message.

The *MsgType* and *Report* options specified in the message descriptor of the command message, together with the success or failure of the command, determine whether response messages are sent or not. If no responses are requested, and the command message contains an error, it will be discarded.

Notes:

1. If there are multiple errors in a command message, a single response message will be generated.
2. Brokers do not request publishers or subscribers to generate responses.

Message descriptor for response messages

When the broker sends a response message, all the fields of the message descriptor are set to their default values, except for the following:

Report	Set to zeroes.
MsgType	Set to MQMT_REPLY.
Format	Set to MQFMT_RF_HEADER.

MsgId	Set according to the *Report* options in the original command message. By default, this means that it is set to MQMI_NONE, so that the queue manager generates a unique value.
CorrelId	Set according to the *Report* options in the original command message. By default, this means that the *CorrelId* is set to the same value as the *MsgId* of the command message. This can be used to correlate commands with their responses.
Priority	The same value as in the original command message.
Persistence	The same value as in the original command message.
Expiry	The same value as in the original command message received by the broker.
PutApplType	Set to MQAT_QMGR.
PutApplName	Set to the first 28 characters of the queue manager name.

Other context fields are set as if generated with MQPMO_PASS_IDENTITY_CONTEXT.

Types of error response

The broker generates three types of response.

OK response

This indicates that the command completed successfully. The response consists of a message that contains an MQRFH format header with the *CompCode* tag name in the *NameValueString* set to the value of MQCC_OK.

An OK response is sent by the broker if the command message was sent with a *MsgType* of MQMT_REQUEST, or if it was sent with a *MsgType* of MQMT_DATAGRAM and the MQRO_PAN *Report* option was set.

Warning response

This indicates that the command was only partially successful. The response consists of a message that contains an MQRFH format header with the *CompCode* tag name in the *NameValueString* set to the value of MQCC_WARNING. The *Reason* and the *ReasonText* tag names and values identify the nature of the warning.

A warning response is sent by the broker if the command message was sent with a *MsgType* of MQMT_REQUEST, or if it was sent with a *MsgType* of MQMT_DATAGRAM and either the MQRO_PAN or MQRO_NAN *Report* options were set.

Error response

This indicates that the command has failed. The response consists of a message that contains an MQRFH format header with the *CompCode* tag name in the *NameValueString* set to the value of MQCC_FAILED. The *Reason* and the *ReasonText* tag names and values identify the nature of the failure, and additional tags may be used to give more information.

Error responses are sent by the broker if the command message was sent with a *MsgType* of MQMT_REQUEST, or if it was sent with a *MsgType* of MQMT_DATAGRAM and the MQRO_NAN *Report* option was set.

Broker responses

A **Broker Response** message is sent from a broker to the *ReplyToQ* of a publisher or a subscriber, to indicate the success or failure of a command message received by the broker.

The standard parameters listed below will always be returned in the order shown. In the case where an error is being reported, they may be followed by an optional parameter (depending on the command message that failed) which gives more information about the error.

With multiple errors, the group of standard and optional parameters will be repeated as necessary.

The *NameValueString* of the command message that caused an error will usually be appended to the broker response message following the MQRFH structure, to assist in diagnosis of the error. However, in the case of an MQRC_RFH_ERROR or MQRCCF_MSG_LENGTH_ERROR, the *NameValueString* of the command message that caused the error will not be appended to the broker response message.

Standard parameters:
CompCode, Reason, ReasonText

Optional parameters:
DeleteOptions, ErrorId, ErrorPos, ParameterId, PublicationOptions, QMgrName, QName, RegistrationOptions, StreamName, Topic, UserId

Standard parameters

CompCode

name: "MQPSCompCode" (string constant: MQPS_COMPCODE)
value: The completion code is returned in decimal form, and takes one of three values:

MQCC_OK	Command completed successfully
MQCC_WARNING	Command completed with warning
MQCC_FAILED	Command failed

Reason

name: "MQPSReason" (string constant: MQPS_REASON)
value: A decimal value corresponding to the error code. It is set to the value of MQRC_NONE if *CompCode* is set to MQCC_OK.

Error codes are listed on page 97, and in the sections describing individual command messages.

ReasonText

name: "MQPSReasonText" (string constant: MQPS_REASON_TEXT)
value: A string corresponding to the error code. It is set to MQRC_NONE if *CompCode* is set to MQCC_OK.

Error codes are listed on page 97, and in the sections describing individual command messages.

Optional parameters

Command

name: `"MQPSCommand"` (string constant: MQPS_COMMAND)
value: The incorrect command that was specified when a command fails with MQRC_RFH_COMMAND_ERROR.

DeleteOptions

name: `"MQPSDelOpts"` (string constant: MQPS_DELETE_OPTIONS)
value: The incorrect delete options that were specified when a command fails with MQRCCF_DEL_OPTIONS_ERROR.

ErrorId

name: `"MQPSErrorId"` (string constant: MQPS_ERROR_ID)
value: An additional reason code (decimal value) when a command fails with MQRCCF_Q_MGR_NAME_ERROR, MQRCCF_Q_NAME_ERROR or MQRCCF_NOT_AUTHORIZED. For example, the value might be MQRC_UNKNOWN_ENTITY indicating that the subscriber is not authorized because it is unknown to the broker.

ErrorPos

name: `"MQPSErrorPos"` (string constant: MQPS_ERROR_POS)
value: A decimal value indicating the position in the *NameValueString* of the command message sent to the broker at which an error was found. An error at the first character is reported with an error position of zero.

If the first ‘MQPS’ tag isn't `MQPSCommand`, the command will fail with an MQRC_RFH_COMMAND_ERROR, and the `MQPSErrorPos` tag will indicate the position of the offending tag.

If no ‘MQPS’ tags were encountered, the command will fail with an MQRC_RFH_COMMAND_ERROR, and the `MQPSErrorPos` tag will be set to the last character in the string.

If an ‘MQPS’ tag doesn't have a matching value, or a quoted name or value doesn't have a matching end quote, the command will fail with an MQRC_RFH_STRING_ERROR, and the `MQPSErrorPos` tag will indicate the position in the string where the error was detected.

ParameterId

name: `"MQPSParmId"` (string constant: MQPS_PARAMETER_ID)
value: The incorrect parameter that was specified, or the parameter that was missing, when a command fails with MQRC_RFH_PARM_ERROR, MQRC_RFH_DUPLICATE_PARM or MQRC_RFH_PARM_MISSING.

PublicationOptions

name: `"MQPSPubOpts"` (string constant: MQPS_PUBLICATION_OPTIONS)
value: The incorrect publication options that were specified when a command fails with MQRCCF_PUB_OPTIONS_ERROR.

QMgrName

name: `"MQPSQMgrName"` (string constant: MQPS_Q_MGR_NAME)
value: The invalid queue manager name that was specified when a command fails with MQRCCF_Q_MGR_NAME_ERROR.

QName

name: `"MQPSQName"` (string constant: MQPS_Q_NAME)
value: The invalid queue name that was specified when a command fails with MQRCCF_Q_NAME_ERROR.

RegistrationOptions

name: `"MQPSRegOpts"` (string constant: MQPS_REGISTRATION_OPTIONS)
value: The incorrect registration options that were specified when a command fails with MQRCCF_REG_OPTIONS_ERROR.

StreamName

name: `"MQPSStreamName"` (string constant: MQPS_STREAM_NAME)
value: The unknown or incorrect stream name that was specified when a command fails with MQRCCF_UNKNOWN_STREAM or MQRCCF_STREAM_ERROR.

Topic

name: `"MQPSTopic"` (string constant: MQPS_TOPIC)
value: Up to 256 characters of the incorrect topic name that was specified when a command fails with MQRCCF_TOPIC_ERROR.

UserId

name: `"MQPSUserId"` (string constant: MQPS_USER_ID)
value: The user ID to which the publisher or subscriber is currently assigned when a command fails with MQRCCF_DUPLICATE_IDENTITY.

Examples

Here are some examples of the *NameValueString* in a **Broker Response** message. A successful response will be as follows:

```
MQPSCompCode    0
MQPSReason      0
MQPSReasonText  MQRC_NONE
```

Examples of failure responses are:

```
MQPSCompCode    2
MQPSReason      2102
MQPSReasonText  MQRC_RESOURCE_PROBLEM

MQPSCompCode    2
MQPSReason      3082
MQPSReasonText  MQRCCF_REG_OPTIONS_ERROR
MQPSRegOpts     DeregAll
```

Error codes applicable to all commands

The following reason codes might be returned in the *NameValueString* of the response message for any of the commands, in addition to the codes listed for each command message. See Appendix A, “Reason codes” on page 173 for detailed descriptions of these codes.

Reason	Reason text	Explanation
2334	MQRC_RFH_ERROR	MQRFH structure not valid.
2335	MQRC_RFH_STRING_ERROR	"NameValueString" field not valid.
2336	MQRC_RFH_COMMAND_ERROR	Command not valid.
2337	MQRC_RFH_PARM_ERROR	Parameter not valid.
2338	MQRC_RFH_DUPLICATE_PARM	Duplicate parameter.
2339	MQRC_RFH_PARM_MISSING	Parameter missing.
3016	MQRCCF_MSG_LENGTH_ERROR	Message length not valid.
3023	MQRCCF_MD_FORMAT_ERROR	Format not valid.
3050	MQRCCF_ENCODING_ERROR	Encoding error.
3079	MQRCCF_INCORRECT_Q	Command sent to wrong broker queue.

Problem determination

Check that you are not using MQSeries facilities that are not supported by MQSeries Publish/Subscribe (see "Limitations" on page 47).

Problems with brokers are reported as AMQ58xx messages, which are described in Appendix B, "Error messages" on page 179.

Problems with the command messages sent to brokers by publisher and subscriber applications are reported in broker response messages (described in "Broker responses" on page 94). You should remember to set the *MsgType* and *Report* options in the message descriptor of the command message so that the broker will send a response message (see "The message descriptor" on page 44).

Even if there are no problems with the brokers and command messages, you might find that subscribers do not receive the publications they expect. Here is a list of possible causes:

- One or more of the brokers in the network isn't running.
- The subscription has expired, or failed to be made in the first place.

  Use the `amqspsd` sample to check that the broker has knowledge of the subscribing application's subscription.
- If the publishing application is running at a different broker, a channel might be down.

  Check that all channels between the publishing and subscribing brokers have been started. If not then the subscriber's publication might be sitting on a transmission queue.
- If the publishing application is running at a different broker, the subscription might not have been propagated to that broker yet.

  Even though a subscribing application has received a positive reply to its **Register Subscriber** command message, the subscription might not have propagated to the publishing broker. Check all channels between the subscribing and publishing brokers. Also check the SYSTEM.BROKER.CONTROL.QUEUE at each of these brokers, since an intermediate broker might not have processed the propagated subscription yet.

  Note that brokers process publish messages in batches. This is controlled by the *PublishBatchSize* parameter (see "Broker configuration parameters" on page 113). The effect of this is that, in general, publish messages are processed more rapidly than subscriptions. If you are loading your system with a large number of new subscriptions there might be a delay before they are propagated to all brokers in the network.
- The publishing application might not have published successfully.

  Don't always assume that the problem is with the subscribing application. Make sure that the publishing application received a positive response message from its broker. If it is publishing using MQMT_DATAGRAM messages and doesn't specify either the MQRO_NAN or MQRO_PAN report options, then the broker won't send it a reply message even if the **Publish** command messages fails. If such a publishing application doesn't use the `NoReg` publication option, then it must set up a valid *`ReplyToQ`* in the message descriptor.

- The broker might be putting the subscriber's publications to the dead-letter queue.

  There might be a problem with the subscriber's queue. For example, it might be put-inhibited or the publications might be too large for the queue. In this case the broker will, by default, put these messages to the dead-letter queue (DLQ). Check the DLQ at the subscriber's broker. The broker will also issue message AMQ5882 if it has had to put a message to the DLQ.

- The stream might not be supported by all necessary brokers.

  If the publication is not being published on the default stream, all brokers in the network between the publishing and subscribing brokers must support the stream you are using. Use the `amqspsd` sample to check that the stream is supported by all necessary brokers.

Chapter 9. Sample programs

Table 4 shows the techniques demonstrated by the sample programs supplied with MQSeries Publish/Subscribe.

Table 4. Sample programs

Technique	C source	Executable	MQSC script
Results service	amqsresa.c	amqsres	-
Match simulator	amqsgama.c	amqsgam	-
Administration application	amqspsda.c	amqspsd	-
Routing exit	amqspsra.c	-	-
Create definitions for sample application	-	-	amqsresa.tst
Create stream on another broker	-	-	amqsgama.tst
Create administration app. reply queue	-	-	amqspsda.tst
Create SYSTEM.BROKER.MODEL.STREAM	-	-	amqsfmda.tst

You can find the samples in the following directories.

AIX

source files	`/usr/lpp/mqm/samp/pubsub`
`amqspsda.*`	`/usr/lpp/mqm/samp/pubsub/admin`
executables	`/usr/lpp/mqm/samp/bin`

| **Compaq Tru64 UNIX**, **HP-UX**, **Linux**, and **Sun Solaris**

source files	`/opt/mqm/samp/pubsub`
`amqspsda.*`	`/opt/mqm/samp/pubsub/admin`
executables	`/opt/mqm/samp/bin`

| **Windows NT and Windows 2000**

source files	`<drive:directory>\MQM\TOOLS\C\SAMPLES\PUBSUB`
`amqspsda.*`	`<drive:directory>\MQM\TOOLS\C\SAMPLES\PUBSUB\ADMIN`
executables	`<drive:directory>\MQM\TOOLS\C\SAMPLES\BIN`

The sample programs are described in the following sections:

- `amqsres` and `amqsgam` in “Sample application” on page 102
- `amqspsd` in “Sample program for administration information” on page 166
- `amqspsr` in “Sample routing exit” on page 146

You must start the queue manager before running the MQSC scripts. In addition, before running the executables, you must start the broker (see Chapter 11, “Controlling the broker” on page 119).

Instructions for compiling the samples can be found in the *MQSeries Application Programming Guide*.

© Copyright IBM Corp. 1998, 2000

Sample application

The following aspects of the results service application are described in “Sample application” on page 30:

- The use of streams other than the default stream.
- Event publications (not retained).
- State publications (retained).
- Wildcard matching of topic strings.
- Multiple publishers on the same topics (event publications only).
- The need to subscribe to a topic *before* it is published on (event publications).
- A subscriber continuing to be sent publications when that subscriber (not its subscription) is interrupted.
- The use of retained publications to recover state after a subscriber failure.

The application’s use of multiple subscription identities on the same subscriber queue is covered in “Publisher and subscriber identity” on page 43, and the following aspects are described in Chapter 6, “Format of command messages” on page 61:

- MQRFH format messages.
- MQRFH *NameValueString* parsing.
- MQRFH broker response message checking.
- **Publish**, **Register Subscriber**, **Request Update**, **Delete Publication** and **Deregister Subscriber** command messages.
- Separate user data in **Publish** messages.

Running the application

To run the application on a single queue manager, first start the queue manager and then enter the following command:

```
runmqsc QMgrName < amqsresa.tst
```

where `QMgrName` is the queue manager that the results service will use (if `QMgrName` is omitted, the default queue manager will be assumed). This will create the appropriate queues on the queue manager. Then start the broker (see Chapter 11, “Controlling the broker” on page 119).

The results service program is started by entering the following:

```
amqsres QMgrName
```

`QMgrName` is optional, and defaults to the default queue manager. The results service will produce the following output:

```
Results Service is ready for match input,
instances of amqsgam can now be started.
```

You can now start one or more match simulators by entering the following:

```
amqsgam Team1 Team2 QMgrName
```

`QMgrName` is optional, as before.

Typical output from a match simulator is:

```
Match between Team1 and Team2
GOAL! Team2 scores after 20 minutes
GOAL! Team1 scores after 25 minutes
GOAL! Team1 scores after 38 minutes
GOAL! Team2 scores after 73 minutes
Full time
```

This would produce corresponding output from the results service:

```
LATEST: Team1 0, Team2 0
LATEST: Team1 0, Team2 1
LATEST: Team1 1, Team2 1
LATEST: Team1 2, Team2 1
LATEST: Team1 2, Team2 2
FULLTIME: Team1 2, Team2 2
```

A match simulator can be run on a different queue manager in the broker hierarchy if required. In this case, you need to enter the following command:

```
runmqsc QMgrName < amqsgama.tst
```

to create the appropriate stream queue on that queue manager. This must be done *before* starting the results service and the match simulator.

The team names must be 31 characters or less in length, and contain no blanks. The simulator runs for 30 seconds and scores goals at random for each side.

The simulator publishes event publications on the following topics:

```
Sport/Soccer/Event/MatchStarted
Sport/Soccer/Event/ScoreUpdate
Sport/Soccer/Event/MatchEnded
```

The *UserData* is contained in a formatted string following the *NameValueString* of the MQRFH header. In the case of ‘MatchStarted’ or ‘MatchEnded’ it consists of both team names in the following structure:

```
{
   MQCHAR32  Team1;
   MQCHAR32  Team2;
}
```

For a ‘ScoreUpdate’ the *UserData* consists of the name of the team that scored:

```
   MQCHAR32  TeamThatScored;
```

The team names are NULL padded to 32 characters.

The results service program subscribes to these three topics to monitor the state of play in the matches that are active. It publishes the latest score in the match between Team1 and Team2 on the following topic:

```
Sport/Soccer/State/LatestScore/Team1 Team2
```

In this case the *UserData* is a variable string containing the data in the format:

```
"Team1Score Team2Score"
```

For example `"0 0"` or `"2 1"`.

Figure 20 on page 104 illustrates the situation when four match simulators are running.

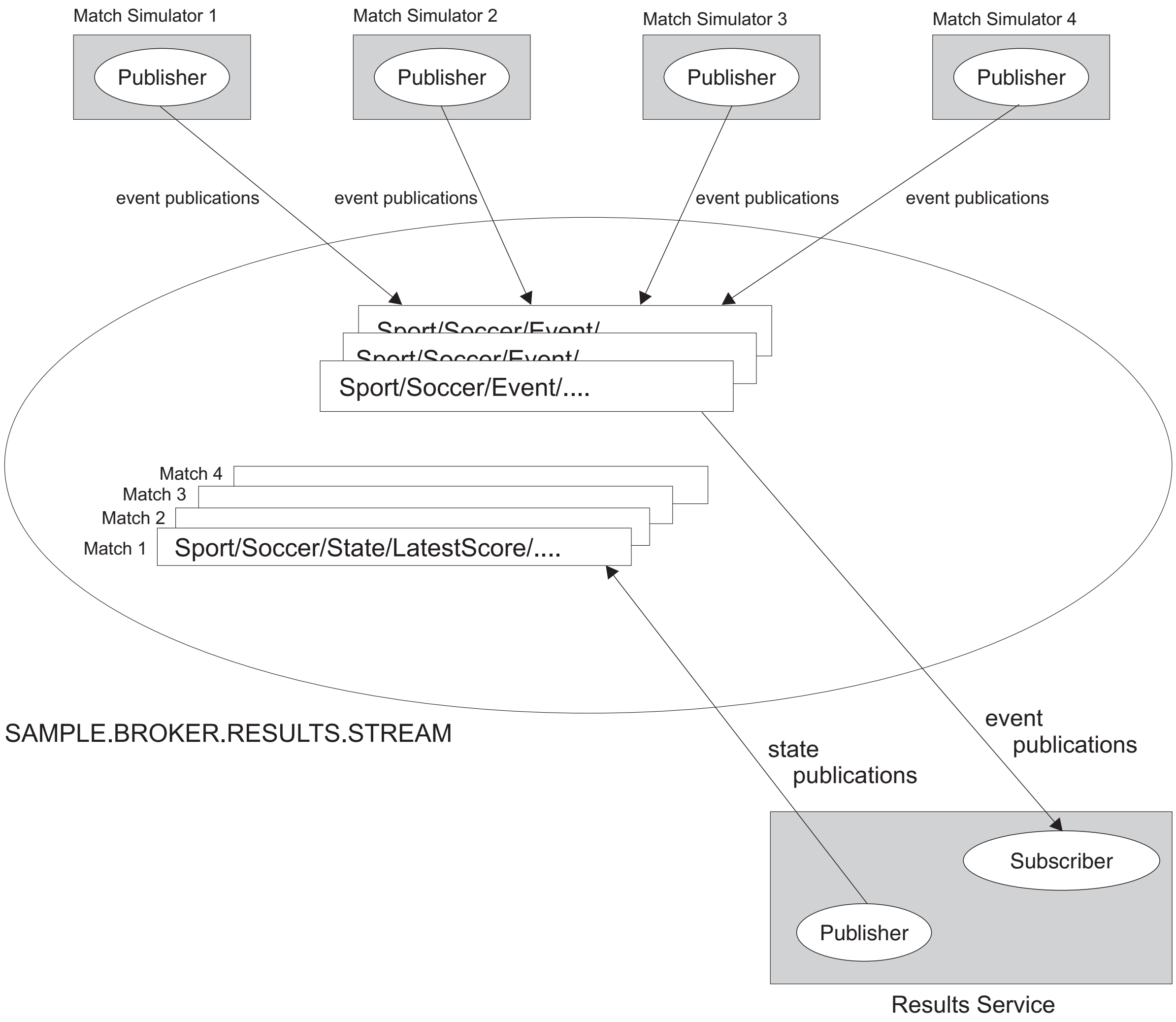

Match Simulator 1
Match Simulator 2
Match Simulator 3
Match Simulator 4
Publisher
Publisher
Publisher
Publisher
event publications
event publications
event publications
event publications
Sport/Soccer/Event/
Sport/Soccer/Event/
Sport/Soccer/Event/....
Match 4
Match 3
Match 2
Match 1
Sport/Soccer/State/LatestScore/....
SAMPLE.BROKER.RESULTS.STREAM
state publications
event publications
Subscriber
Publisher
Results Service

Figure 20. Results service running with four match simulators. The match simulators send event publications to three topics (MatchStarted, ScoreUpdate, MatchEnded). The results service subscribes to these, and sends state publications to four state topics (the LatestScore for each match).

Once a match has ended, the retained publication that contains its latest score is deleted. After a period of inactivity (45 seconds), the results service deregisters the subscription from the `Sport/Soccer/Event/*` topic and the program ends with the message:

```
Results Service has ended
```

If the results service program (`amqsres`) is stopped and restarted while the match simulators are still running, the results will be restored to their correct values and processing will continue as before.

Possible extensions

The sample application illustrates many aspects of an MQSeries Publish/Subscribe system. Possible extensions which might be implemented by the user include:

- Distribute the results service and match simulators across multiple connected brokers.
- Extend the application to handle more than one sport, and have a results service running for each sport.

- Extend the results service to publish the final score when a match ends, and add another application that subscribes to these publications to produce a table of results.
- Extend the match simulator to confirm that a results service is subscribing to `Sport/Soccer/Events/*` topics before it starts publishing. This can be done using metatopics.
- Change the format of the user data in the publications, create a user defined format, and write a data conversion exit to enable the passing of publications between different platforms or languages.

Application Messaging Interface samples

For sample publisher and subscriber programs that use the Application Messaging Interface in C, C++ and Java, see the *MQSeries Application Messaging Interface* manual.

Part 3. Managing the broker

Chapter 10. Setting up a broker 109
Broker queues 109
  System queues 109
  Other stream queues 110
    SYSTEM.BROKER.MODEL.STREAM 110
  Internal queues 111
  Dead-letter queue 111
Other considerations 111
  Access control 111
  Backup 112
Broker configuration stanza 112
  Broker configuration tool 112
  Broker configuration parameters 113

Chapter 11. Controlling the broker 119
Starting a broker 119
  Using triggering to start the broker 119
Stopping a broker 119
Displaying the status of a broker 119
Adding a stream 120
  Creating a stream queue 120
  Informing other brokers about the stream 120
Deleting a stream 120
  Deleting a stream on an isolated broker 121
  Deleting a stream on a broker that is part of a network 121
Adding a broker to a network 121
Deleting a broker from the network 122
  Problems when deleting brokers 122
  Deleting a broker that has a child broker 122
Sequence of commands for adding and deleting brokers 123

Chapter 12. Control commands 125
clrmqbrk (Clear broker's memory of a neighboring target broker) 126
dltmqbrk (Delete broker) 129
dspmqbrk (Display broker status) 131
endmqbrk (End broker function) 132
migmqbrk (Migrate broker to MQSeries Integrator) 133
strmqbrk (Start broker function) 135

Chapter 13. Message broker exit 137
Publish/subscribe routing exit 138
    C invocation 139
  Publish/subscribe routing exit parameter structure 139
    Fields 139
    C declaration 145
Writing a publish/subscribe routing exit program 145
  Limitations on MQSeries work done in the routing exit 146
  Security considerations 146
Compiling a publish/subscribe routing exit program 146
Sample routing exit 146

© Copyright IBM Corp. 1998, 2000

Chapter 10. Setting up a broker

Publishers, subscribers, and brokers communicate by using queues. Configuration and monitoring of these queues can be performed by whatever technique is currently in use for MQSeries, whether supplied by MQSeries or available from third parties.

Before you can use MQSeries Publish/Subscribe you need to do the following things to set up your broker:

- If necessary, define the queues that the broker needs to use
- Authorize applications to use these queues
- Review the default settings of the broker parameters in the queue manager initialization file (qm.ini)

For information about managing your brokers when they have been set up, see Chapter 11, “Controlling the broker” on page 119.

How to find out about publishers and subscribers registered with brokers, and how to write applications to manage a network of brokers is explained in Part 4, “System programming” on page 147.

Broker queues

Brokers are event-driven; they wait for messages to arrive on their queues. The broker needs several system queues, and can also have any number of *stream* queues; these are described below.

Note: If you are using MQSeries Version 5.2 or 5.1, please note that stream queues must not be cluster queues.

System queues

The broker uses three system queues. These queues all have names beginning with SYSTEM.BROKER, and are used for the purposes described below. These queues are created automatically when the broker starts if they do not already exist. You might want to alter access authority to these queues.

SYSTEM.BROKER.CONTROL.QUEUE

This is the broker’s control queue. Publisher and subscriber applications, and other brokers, send all command messages (except publications and requests to delete publications) to this queue.

SYSTEM.BROKER.CONTROL.QUEUE is created as a predefined queue based on the SYSTEM.DEFAULT.LOCAL.QUEUE.

SYSTEM.BROKER.DEFAULT.STREAM

This is the queue that receives all publication messages for the default stream. Applications can also send requests to delete publications on the default stream to this queue.

SYSTEM.BROKER.DEFAULT.STREAM is created using SYSTEM.BROKER.MODEL.STREAM if it exists, otherwise the broker predefines it based on the SYSTEM.DEFAULT.LOCAL.QUEUE.

© Copyright IBM Corp. 1998, 2000

Note: SYSTEM.BROKER.DEFAULT.STREAM is created with a default persistence of yes. This means that an application using the MQPER_AS_Q_DEF option in the message descriptor (the default) will be publishing persistent messages by default.

SYSTEM.BROKER.ADMIN.STREAM
This is the queue that the broker uses to publish its own broker configuration information (for example the identity of its parent). If you write your own administration applications, they can use the information published on this stream. You can also publish information on this stream (but not to topics with names beginning `MQ/`).

SYSTEM.BROKER.ADMIN.STREAM is created using SYSTEM.BROKER.MODEL.STREAM if it exists, otherwise the broker predefines it based on the SYSTEM.DEFAULT.LOCAL.QUEUE.

Other stream queues

Stream queues are used to process publications for all topics within a stream. Applications send publications (and requests to delete publications) to a stream queue. The stream queue must be a local queue at the broker, not an alias or remote queue. Applications can send messages to a stream queue through an alias or remote queue.

Publishing applications can register with the broker before they start sending publications. If the application specifies that it will be using a stream queue that does not yet exist, the broker might create a permanent dynamic queue with the same name as the stream specified, based on the SYSTEM.BROKER.MODEL.STREAM queue.

If the SYSTEM.BROKER.MODEL.STREAM queue does not exist, any message sent by an application that refers to a stream for which there is no stream queue, will be rejected. The broker keeps information about which streams are known to it so that, when it is restarted, it can recognize the stream queues.

Applications can also specify the stream name in a publication message. If a publication message specifies the name of a stream that is different from the name of the queue to which it was sent, the message is rejected. If the application does not specify a stream name, it defaults to the name of the stream queue to which it is sent.

If you are using a network of brokers, and you want to restrict a certain stream to a particular sub-tree of the hierarchy, brokers immediately outside the sub-tree must not have a SYSTEM.MODEL.STREAM.QUEUE defined. All stream queues for streams that these brokers support must, therefore, be defined by the administrator.

SYSTEM.BROKER.MODEL.STREAM

The SYSTEM.BROKER.MODEL.STREAM is a model queue definition that can be used by the broker to create dynamic queues to receive publications for streams other than the default stream. This is only used if the stream queue does not already exist. This definition must specify that the dynamic queue to be created is a permanent-dynamic queue. If this queue does not exist, all stream queues must be defined by the administrator. (The administrator can also define stream queues manually, even if this queue does exist.)

This queue is supplied as sample `amqsfmda.tst` (see page 101). To create the queue from the sample, use the following command:

```
runmqsc QMgrName < amqsfmda.tst
```

where `QMgrName` is the name of the queue manager.

Internal queues

The broker creates several other queues for its own internal use. These queues also have names beginning with SYSTEM.BROKER. The broker uses them to store its persistent state, such as subscriptions and retained publications.

Dead-letter queue

You are recommended to set up a dead-letter queue for each queue manager that has a broker running on it. This enables the broker to continue operating when problems are encountered, such as a subscriber's queue being full. In this case publications for that subscriber are put to the dead-letter queue, and the broker continues to process publish command messages.

Without a dead-letter queue you might also have problems if you want to delete that broker from the network (see "Deleting a broker from the network" on page 122).

Other considerations

Access control

Normal MQSeries access control techniques apply to applications and brokers opening queues for Publish/Subscribe messages. These authorization checks are carried out using standard MQSeries functions. The authority is tested before any message is sent to a particular identity after a broker restart, but not necessarily each subsequent time a message is put (see "Streams" on page 24).

Any application putting a message to the broker's SYSTEM.BROKER.CONTROL.QUEUE must have authorization to put messages to this queue.

A publisher must be authorized to put messages on the broker's appropriate stream queue.

Subscribers must be authorized to browse the broker's stream queue; this is checked by the broker because the subscriber does not try to open the broker's stream queue. In addition, a subscriber must have authority to put messages on the subscriber queue that the publications will be sent to.

There is no topic based security; the access check is for the stream and there are no further checks on topics within a particular stream.

Backup

Normal MQSeries backup and restore procedures apply, as described in the *MQSeries System Administration* book. When a queue manager is backed up, a broker installed on that queue manager will be backed up as well.

Broker configuration stanza

Broker parameters are controlled by the `Broker` stanza of the queue manager configuration file, qm.ini. Figure 21 shows an example of this stanza. The parameters are described in “Broker configuration parameters” on page 113.

```
Broker:
   MaxMsgRetryCount=5
   StreamsPerProcess=1
   OpenCacheSize=128
   OpenCacheExpiry=300
   PublishBatchSize=5
   PublishBatchInterval=0
   ChkPtMsgSize=100000
   ChkPtActiveCount=400
   ChkPtRestartCount=40
   RoutingExitPath=/opt/mqm/samp/bin/amqspsra(RoutingExit)
   RoutingExitConnectType=STANDARD
   RoutingExitAuthorityCheck=no
   RoutingExitData=My routing exit string data
   SyncPointIfPersistent=no
   DiscardNonPersistentInputMsg=no
   DLQNonPersistentResponse=yes
   DiscardNonPersistentResponse=no
   DLQNonPersistentPublication=yes
   DiscardNonPersistentPublication=no
   GroupId=nobody
```

Figure 21. Sample Broker stanza for qm.ini

Note: You do not need to list parameters if you are using their default values. Any parameters that you do list will be checked for validity. A blank entry is not valid.

Broker configuration tool

If you are running MQSeries for Windows NT and Windows 2000 version 5.2 or MQSeries for Windows NT version 5.1, you can use the broker configuration tool. This tool has a graphical user interface for setting up the broker configuration parameters. The tool is started by entering the following on the command line:

```
cfgmqbrk
```

The left-hand panel provides a list of queue managers. Click on one of them to display its broker configuration parameters in the right-hand panel.

Make any necessary changes to the parameters. You can do this by overtyping the values, or by using **Cut**, **Copy**, **Paste**, **Delete** and **Select** in the **Edit** menu or on the Toolbar. Then use **Save**, **Reload** or **Reset** in the **File** menu to save the changes, reload the stored parameters, or reset to the original values.

The **View** menu allows you to switch the toolbar and status bar on and off.

Broker configuration parameters

MaxMsgRetryCount=*number*

When the broker fails to process a command message under syncpoint (for example a publish message that cannot be delivered to a subscriber because the subscriber queue is full and it is not possible to put the publication to the dead-letter queue), the unit of work will be backed out and the command retried this number of times before the broker will attempt to process the command message according to its report options instead.

The default is *MaxMsgRetryCount*=5.

StreamsPerProcess=*number*

The broker consists of a broker main process (amqfcxaa) and a number of broker worker processes (amqfcxba). Each worker process is capable of supporting one or more streams. Depending upon the broker configuration (for example the operating system, number of streams, number of publishers, subscribers and retained messages, whether a non-fastpath routing exit is in use), varying this number can alter capacity or throughput (or both).

If you have a large number of lightly loaded streams you should consider increasing this value.

The defaults are:

AIX

StreamsPerProcess=10	(*RoutingExitConnectType*=Standard)
StreamsPerProcess=1	(*RoutingExitConnectType*=Fastpath)
StreamsPerProcess=1	(no routing exit)

| **Compaq Tru64 UNIX, HP-UX, Linux, and Sun Solaris**

StreamsPerProcess=10

| **Windows NT and Windows 2000**

StreamsPerProcess=10

OpenCacheSize=*number*

Each broker stream thread (2 threads per stream) keeps a cache of recently used open queues. This parameter specifies the maximum number of queues in the cache.

The default is *OpenCacheSize*=128.

OpenCacheExpiry=*number*

Each broker stream thread (2 threads per stream) keeps a cache of recently used open queues. If a queue in the cache is not used for approximately *OpenCacheExpiry* seconds, the queue is removed from the cache (closed).

The default is *OpenCacheExpiry*=300.

PublishBatchSize=*number*

The broker normally processes publish messages within syncpoint. It can be inefficient to commit each publication individually, and in some circumstances the broker can process multiple publish messages in a single unit of work. This parameter specifies the maximum number of publish messages that can be processed in a single unit of work.

The default is *PublishBatchSize*=5.

PublishBatchInterval=*number*

The broker normally processes publish messages within syncpoint. It can be inefficient to commit each publication individually, and in some circumstances the broker can process multiple publish messages in a single unit of work. This parameter specifies the maximum time (in milliseconds) between the first message in a batch and any subsequent publication included in the same batch. A batch interval of 0 indicates that up to *PublishBatchSize* messages can be processed, provided that the messages are available immediately.

The default is *PublishBatchInterval*=0.

ChkPtMsgSize=*number*

The broker stores individual publisher and subscriber registrations as messages on its internal queues. Periodically, it might consolidate a number of these registrations into a smaller number of larger messages called checkpoint messages. This action is called checkpointing and is performed to reduce the number of messages that need to be read to restore the publisher and subscriber registrations at broker and stream restart.

The *ChkPtMsgSize* parameter determines the default size of each checkpoint message in bytes, which in turn determines the number of registrations that each checkpoint message can contain.

The default is *ChkPtMsgSize*=100000.

ChkPtActiveCount=*number*

The broker stores individual publisher and subscriber registrations as messages on its internal queues. Periodically it might consolidate a number of these registrations into a smaller number of larger messages called checkpoint messages. This action is called checkpointing and is performed to reduce the number of messages that need to be read to restore the publisher and subscriber registrations at broker and stream restart.

The number of changes that need to be made to part of the registration state of an individual stream during normal broker operation before checkpointing is considered for that part depends on the *ChkPtActiveCount* parameter.

The default is *ChkPtActiveCount*=400. A lower value will make checkpointing occur more frequently. A higher value will make checkpointing occur less frequently. A value of 0 disables checkpointing completely during normal operation and would be applicable if checkpoint activity was having an adverse effect on broker throughput.

ChkPtRestartCount=*number*

The broker stores individual publisher and subscriber registrations as messages on its internal queues. Periodically it might consolidate a number of these registrations into a smaller number of larger messages called checkpoint messages. This action is called checkpointing and is performed to reduce the number of messages that need to be read to restore the publisher and subscriber registrations at broker and stream restart.

The number of changes that need to have been made to part of the registration state of an individual stream during broker or stream restart before checkpointing is considered for that part depends on the *ChkPtRestartCount* parameter.

The default is *ChkPtRestartCount*=40. This is lower than the *ChkPtActiveCount* on the assumption that stream or broker restart is a more suitable time for the registration state to be checkpointed. A value of 0 disables checkpointing completely during restart.

RoutingExitPath=*[path]module_name(function_name)*

Before the broker sends a publication to a subscriber, the broker invokes the exit identified by the *RoutingExitPath* (if any).

The default is no routing exit.

RoutingExitConnectType=*FASTPATH*|*STANDARD*

If the broker is configured to use a routing exit, the exit runs within a broker process. If the exit conforms to the requirements of a fastpath application (MQCNO_FASTPATH_BINDING), the broker process can use a fastpath connection to the queue manager. This attribute informs the broker if the exit meets the standards necessary for a fastpath application.

Note: This attribute is only relevant if a *RoutingExitPath* is specified. For performance reasons *RoutingExitConnectType*=FASTPATH is desirable.

The default is *RoutingExitConnectType*=STANDARD.

RoutingExitAuthorityCheck=*yes*|*no*

Before the broker sends a publication to a subscriber the broker must have validated the subscribers authority to write to the subscriber queue. If the routing exit changes the message destination, the authority check already performed by the broker is not valid. This attribute informs the broker if the authority check should be repeated for any changed destination.

Note: The performance implications of setting *RoutingExitAuthorityCheck*=yes are considerable if the routing exit frequently changes the destination.

The default is *RoutingExitAuthorityCheck*=no.

RoutingExitData=*string*

If the broker is using a routing exit, the broker invokes the routing exit passing an MQPXP structure as input. The data specified using this attribute is provided in the *ExitData* field. The string can be up to MQ_EXIT_DATA_LENGTH characters in length.

The default is 32 blank characters.

SyncPointIfPersistent=*yes***|***no*

If this attribute is specified, when the broker reads a publish or delete publication message from a stream queue during normal operation the broker specifies MQGMO_SYNCPOINT_IF_PERSISTENT. This makes the broker receive nonpersistent messages outside syncpoint. If the broker receives a publication outside syncpoint, the broker will forward that publication to subscribers known to the broker outside syncpoint.

When using *SyncPointIfPersistent*=yes it is possible that a nonpersistent publication might not be delivered to all subscribers known to a broker (for example, if an immediate broker shutdown occurred while a publish message was being processed). If *SyncPointIfPersistent*=yes is specified, the broker performance for publishing nonpersistent publications will improve.

The default is *SyncPointIfPersistent*=no.

DiscardNonPersistentInputMsg=*yes***|***no*

If the broker cannot process a nonpersistent input message, the broker might attempt to write the input message to the dead-letter queue (depending on the report options of the input message). If the attempt to write the input message to the dead-letter queue fails, and the MQRO_DISCARD_MSG report option was specified on the input message or *DiscardNonPersistentInputMsg*=yes, the broker will discard the input message. If *DiscardNonPersistentInputMsg*=no is specified, the broker will only discard the input message if the MQRO_DISCARD_MSG report option was set in the input message.

The defaults are:

DiscardNonPersistentInputMsg=no if *SyncPointIfPersistent*=no.
DiscardNonPersistentInputMsg=yes if *SyncPointIfPersistent*=yes.

Note: If *SyncPointIfPersistent*=yes is set, *DiscardNonPersistentInputMsg*=no must not be set.

DLQNonPersistentResponse=*yes***|***no*

If the broker attempts to generate a response message in response to a nonpersistent input message, and the response message cannot be delivered to the reply-to queue, this attribute indicates if the broker should attempt to write the undeliverable response message to the dead-letter queue.

The default is *DLQNonPersistentResponse*=yes.

DiscardNonPersistentResponse=*yes***|***no*

If the broker attempts to generate a response message in response to a nonpersistent input message, and the response message cannot be delivered to the reply-to queue or written to the dead-letter queue, this attribute indicates whether the broker can discard the undeliverable response message.

The default is:

DiscardNonPersistentResponse=no if *SyncPointIfPersistent*=no.
DiscardNonPersistentResponse=yes if *SyncPointIfPersistent*=yes.

Note: If *SyncPointIfPersistent*=yes is set *DiscardNonPersistentResponse*=no must not be set.

DLQNonPersistentPublication=*yes***|***no*

If the broker fails to send a nonpersistent publication to a subscriber, this attribute indicates whether the broker should attempt to put the publication to the dead-letter queue.

The default is *DLQNonPersistentPublication*=yes.

DiscardNonPersistentPublication=*yes***|***no*

If the broker fails to send a nonpersistent publication to a subscriber and is unable to write the publication to the dead-letter queue, this attribute indicates whether the broker can discard the publication.

The default is:

DiscardNonPersistentPublication=no if *SyncPointIfPersistent*=no.
DiscardNonPersistentPublication=yes if *SyncPointIfPersistent*=yes.

Note: If *SyncPointIfPersistent*=yes is set, *DiscardNonPersistentPublication*=no must not be set.

GroupId=*group_identifier*

Specifies the group that owns the stream queues created by the broker, except the admin stream (for example, SYSTEM.BROKER.DEFAULT.STREAM). Users in this group are able to access the stream queues. If this group does not exist, the broker will not be able to run.

If not specified, the following defaults are used (this normally means that all users can access the stream queues):

AIX, Compaq Tru64 UNIX, Linux, Sun Solaris

GroupId=nobody

HP-UX

GroupId=nogroup

Windows NT and Windows 2000

GroupId=Users

Note: For MQSeries for Windows NT and Windows 2000 Version 5.2, MQSeries for Windows NT Version 5.1 with CSD03 (PTF U200113), or later, the *GroupId* is set to ‘Users’ or the national language equivalent.

Chapter 11. Controlling the broker

This chapter describes the following broker operations:

- “Starting a broker”
- “Stopping a broker”
- “Displaying the status of a broker”
- “Adding a stream” on page 120
- “Deleting a stream” on page 120
- “Adding a broker to a network” on page 121
- “Deleting a broker from the network” on page 122

How to find out about publishers and subscribers registered with brokers, and how to write applications to manage a network of brokers is explained in Part 4, “System programming” on page 147.

Starting a broker

Use the **strmqbrk** command to start a broker.

This starts the broker on the specified queue manager, either initially, or as a restart after an **endmqbrk** command.

This command is described in “strmqbrk (Start broker function)” on page 135.

Using triggering to start the broker

It is also possible to start a broker by enabling triggering on any of the broker’s queues. Triggering on the first message should be specified. Note, however, that it might be unwise for a broker to be triggered on more than one of its stream queues because a trigger message will be generated for each queue at startup.

Stopping a broker

Use the **endmqbrk** command to stop a broker.

This stops the broker on the specified queue manager.

This command is described in “endmqbrk (End broker function)” on page 132.

Displaying the status of a broker

Use the **dspmqbrk** command to display the status of the broker for the specified queue manager.

This command is described in “dspmqbrk (Display broker status)” on page 131.

© Copyright IBM Corp. 1998, 2000

Adding a stream

The following things need to happen for a stream to be created:

- A queue must be created to hold publications for that stream.
- Information about the stream has to be passed to other brokers in the network that need to support the stream.

Creating a stream queue

The stream queue has the same name as the stream, and is usually created by the operator. There should be one instance of the stream queue at each broker that supports the stream. When defining the queue, you must specify the NOSHARE option.

Alternatively, you can let the broker create the stream queue dynamically when it is needed. The queue is based on the model queue definition SYSTEM.BROKER.MODEL.STREAM if this is available. If the model queue definition is not available, the broker will not create stream queues dynamically.

Note: If the queue is created dynamically, the operator will have to grant the required access authority to applications using the queue. Because of this, dynamic stream queue creation should be used only in a test environment.

Informing other brokers about the stream

When a stream is first referenced by a publisher or subscriber (for example, when a registration request is sent to the broker's control queue) the broker informs its neighbors that the stream exists. If the neighboring brokers also have a queue defined for the stream (or can create one using SYSTEM.BROKER.MODEL.STREAM), they also recognize the stream and pass information about it to their neighbors.

If a broker that is told about the stream does not have a queue for the stream and does not have the SYSTEM.BROKER.MODEL.STREAM, it does not pass information about the stream to its neighbors.

Deleting a stream

Before you delete a stream you should quiesce all applications that use the stream.

In order to delete a stream, you need to delete the stream queue. In order to delete the queue, you must ensure that no applications (or channels) have the queue open. If there are messages on the queue, you will have to remove them from the queue, or purge them when you delete the queue.

You must also ensure that you do not have a definition of the SYSTEM.BROKER.MODEL.STREAM on the broker. If you do, and the old one is deleted, a new version of the stream queue will be created dynamically when the broker is restarted.

Deleting a stream on an isolated broker

To delete a stream on a broker that is not part of a broker network:

1. Stop the broker (using **endmqbrk**).
2. Delete the queue.
3. Restart the broker (using **strmqbrk**).

When the broker realizes that the queue no longer exists, it deregisters all subscriptions to the stream, and publishes a message to the SYSTEM.BROKER.ADMIN.STREAM advertising that the stream has been deleted. (For information about the format of this message see “Format of broker administration messages” on page 150.)

Deleting a stream on a broker that is part of a network

A stream on a broker that is part of a broker network is deleted in the same way as for an isolated broker. Other brokers in the network are advised that the stream has been deleted and stop sending publications and subscription requests to the broker for that stream. Messages sent from other brokers before they receive notification that the stream has been deleted are handled as follows:

- Publication messages are put to the dead-letter queue.
- Registration messages are put to the dead-letter queue.

Adding a broker to a network

It is recommended that you define the broker topology from the root down.

Before you can add a broker to the network, channels in both directions must exist between the queue manager which hosts the new broker and the queue manager which hosts the parent. Brokers use explicit addressing when sending messages to queues which reside on another queue manager. When the queue is opened by the broker both the queue and queue manager names will be specified. To facilitate multi-broker operation this queue manager name must resolve to the appropriate transmission queue. The simplest method of achieving this is for the transmission queue to have the same name as the remote queue manager name.

If you do not adopt this naming scheme, then a queue manager alias definition can be used to ensure that messages get placed on the appropriate transmission queue. For example, to specify that messages sent to queue manager PARENT are placed on transmission queue, PARENT.XMITQ:

```
DEFINE QREMOTE (PARENT) RNAME() RQMNAME(PARENT) XMITQ(PARENT.XMITQ)
```

To add a broker to the network, start the broker with the **strmqbrk** command, specifying the name of the parent broker if appropriate. When the broker has been started with a parent named you cannot change the name of its parent, even when the broker is restarted. You cannot change the parent of a broker as part of normal operational procedures without disrupting service.

This command is described in “strmqbrk (Start broker function)” on page 135.

Deleting a broker from the network

Brokers must always be deleted from the bottom of the broker hierarchy. You cannot delete a broker if it has one or more child brokers. (See “Sequence of commands for adding and deleting brokers” on page 123 for more information.)

The broker needs to delete any queues that were created by the broker, so these queues need to be closed and empty.

1. Stop the broker (using **endmqbrk**).
2. Quiesce all applications that use the broker.
3. Applications and brokers can use channels to talk to the broker, so receiving channels might have queues open. If a channel has a queue open, stop and restart the channel.
4. Use the **dltmqbrk** command to delete the broker. This command is described in “dltmqbrk (Delete broker)” on page 129.

The broker performs the actions listed in “dltmqbrk (Delete broker)” on page 129 and sends a message to tell its parent broker that it is no longer active. **This message needs to be processed by its parent broker before the parent can be deleted.** The parent broker will only process this message while running.

If you don’t quiesce all of your applications before deleting the broker, messages might be sent from other brokers before they receive notification that the broker has been deleted. Because there is no broker to handle these messages, the queue manager deals with them according to the report options set for these messages. This means that publication and registration messages are put to the dead-letter queue. Therefore, **you should ensure that a dead-letter queue has been set up for this queue manager before attempting to delete a broker.**

Problems when deleting brokers

If you are unable to delete your broker, consider the following:

- Are any queues that are to be quiesced by the broker open to an application or a channel?

  If so, you will receive an error message containing reason code 5840. The error log will contain information about which queues can't be quiesced.

- Does the broker have any children?

  If it does, you will receive an error message containing reason code 5838. The error log will contain information about the broker’s children.

Deleting a broker that has a child broker

If you are unable to delete a child of a broker you want to delete (for example, because the queue manager of the child broker has been deleted) you can use the **clrmqbrk** command to clear the broker’s memory of the child broker. **This command should be used only in exceptional circumstances, and must be used with great care.** If it is not used correctly, brokers will see an inconsistent view of the hierarchy; this is likely to cause severe disruption to the service.

The command makes it appear as if the child broker has been deleted so that the parent broker can be deleted. If you use this command, you **must** remember to make sure that both ends have the same view of the relationship (see page 126).

Sequence of commands for adding and deleting brokers

This example shows the sequence of commands for adding and deleting brokers in a network. Queue manager A is to host the parent broker and queue manager B is to host the child broker. Channels are defined between the two queue managers. Broker A is the parent broker, so this must be created first. Broker B is then created as a child broker of broker A.

The sequence of commands to achieve this is shown in Figure 22.

```
START CHANNEL (B.to.A)
START CHANNEL (A.to.B)
strmqbrk -m A
strmqbrk -m B -p A
```

Figure 22. Sequence of commands to create brokers in a network

When both brokers are deleted, broker B must be deleted first, and broker A must be available for this to happen. Only when broker B has been deleted can broker A be deleted.

The sequence of commands to achieve this is shown in Figure 23.

```
endmqbrk -m B
STOP CHANNEL (A.to.B)
START CHANNEL (A.to.B)
dltmqbrk -m B

endmqbrk -m A
STOP CHANNEL (B.to.A)
START CHANNEL (B.to.A)
dltmqbrk -m A
```

Figure 23. Sequence of commands to delete brokers in a network

Chapter 12. Control commands

This chapter describes the commands that you can use to manage your brokers. Chapter 11, “Controlling the broker” on page 119 discusses the circumstances under which you would use these commands. The commands are:

- “clrmqbrk (Clear broker’s memory of a neighboring target broker)” on page 126
- “dltmqbrk (Delete broker)” on page 129
- “dspmqbrk (Display broker status)” on page 131
- “endmqbrk (End broker function)” on page 132
- “migmqbrk (Migrate broker to MQSeries Integrator)” on page 133
- “strmqbrk (Start broker function)” on page 135

© Copyright IBM Corp. 1998, 2000

clrmqbrk (Clear broker’s memory of a neighboring target broker)

Purpose

Use the **clrmqbrk** command to clear the broker’s memory of a neighboring (parent or child) target broker. **Using this command should be regarded as exceptional rather than normal.**

The broker cancels all subscriptions from the target broker. The broker must be stopped when this command is issued. The command is synchronous, and when it has completed the broker can be restarted normally. No messages are read from any of the input queues.

After restart, the broker detects any messages on its input queues that came from this broker, and processes them according to their report options.

You need to clear the memory of the broker at each end of the connection. This means that you must issue this command (or an equivalent) to both the parent and the child broker. If you don’t do this, one broker will continue to send messages to the other broker, which will be processed according to their report options. This might lead to a build-up of messages on the dead-letter queue, and unnecessary report messages being sent across the network.

Deleting a broker with **dltmqbrk** requires first deleting its children. If this is impractical (for example, if the child broker is no longer reachable) the **clrmqbrk** command can be used to make the child broker appear deleted to its parent so that the parent can be deleted. The child broker must be deleted whenever practical.

You can also use this command at the child with the **-p** parameter to break the link with its parent. Using such a pair of **clrmqbrk** commands, one at the child and one at the parent, causes the child and its descendants (if any), together with their publishers and subscribers, to be isolated from the rest of the network. The child now becomes the root node of a hierarchy. It can operate this way or be restarted with another parent (or even with its old parent) provided the new parent is not also a descendant.

Note: This command might mean that publications are not sent to subscribers that should receive them, even if the publishers or subscribers have registered with other brokers in the network. When making topology changes such as this to the broker hierarchy, it is the administrator’s responsibility to ensure that publishers are quiesced, and not restarted until the effects of the topology change on the subscription state have propagated through the broker network.

Syntax

```
►►──clrmqbrk──┬─ -p ──────────────┬── -m QMgrName──────────────────────►◄
              └─ -c ChildQMgrName─┘
```

Required parameters

-p Specifies that the link is to be broken with the parent broker. If you specify this parameter, do not specify the -c parameter

-c *ChildQMgrName*

Specifies that the link is to be broken with a child broker; you also need to specify the name of the queue manager that hosts the child broker. If you specify this parameter, do not specify the -p parameter

-m *QMgrName*
The name of the queue manager hosting the broker whose link is to be broken.

Return codes

0 Command completed normally
10 Command completed with unexpected results
20 An error occurred during processing

Examples

In a broker network like this:

```
grandparentQM
      |
  parentQM
      |
   childQM
```

remove the parentQM from the network like this:

1. `clrmqbrk -m grandparentQM -c parentQM`	breaks the link between the broker on grandparentQM and its child on parentQM
2. `clrmqbrk -m parentQM -p`	breaks the link between the broker on parentQM and its parent
3. `clrmqbrk -m parentQM -c childQM`	breaks the link between the broker on parentQM and its child on childQM
4. `clrmqbrk -m childQM -p`	breaks the link between the broker on childQM and its parent

If the broker on childQM is started by `strmqbrk -m childQM -p grandparentQM` at restart, the broker network will now look like this:

```
grandparentQM
      |
   childQM
```

Attention

If you do not issue the **clrmqbrk** command at both ends of a connection, for example, by omitting step 3 above, and you try to reconnect the brokers on parentQM and childQM at restart, it will fail with an AMQ5839 message at the parent, an AMQ5822 message at the child, and an AMQ5839 FDC file will be generated. If you issue the **clrmqbrk** command at both ends of the connection now, it will not fix the problem. You must issue the following commands, assuming that the parent and child brokers are running with channels between them:

1. `endmqbrk -m childQM`

   (wait a few seconds for the failed registration message to reach the child)
2. `clrmqbrk -m childQM -p`
3. `strmqbrk -m childQM`

   (note that there is no parent argument, wait a few seconds for the failed registration message to be removed)
4. `endmqbrk -m parentQM`
5. `endmqbrk -m childQM`
6. `clrmqbrk -m parentQM -c childQM`
7. `strmqbrk -m parentQM`
8. `strmqbrk -m childQM -p parentQM`

These commands restore the connection between the brokers on parentQM and childQM and leave the network looking like this:

```
parentQM
    |
 childQM
```

dltmqbrk (Delete broker)

Purpose

Use the **dltmqbrk** command to delete the broker. The broker must be stopped when this command is issued, and the queue manager running. If the broker is already started, you must issue the **endmqbrk** before issuing this command. To delete more than one broker in the hierarchy, it is essential that you stop (using the **endmqbrk** command) and delete each broker one at a time. You should not attempt to stop all the brokers in the hierarchy that you want to delete first and then try to delete them.

The broker must not have children when this command is issued, because they might be cut off from the rest of the network as a result. If the broker has children and this command is issued, an error message naming at least one child broker will be received. Any children should be deleted before the broker is deleted or, in exceptional circumstances, cleared using the **clrmqbrk** command on this broker.

The broker performs the following actions:

- Put-inhibits its input queues (SYSTEM.BROKER.CONTROL.QUEUE and all stream queues).
- Deregisters all of its subscribers and publishers.
- Sends Delete Publication commands to its parent for its metatopics.
- Deregisters all of its subscriptions with the parent.
- Processes any messages on its input queues according to their report options.

  Note: You are advised to have a dead-letter queue because any input messages will be processed according to their report options. If there is no dead-letter queue, commands might fail.
- Deletes internal queues (purging any messages on the queues).
- Deletes any empty input queues. that were created by the broker in question.
- Terminates.

If the queue manager terminates before the broker has finished deleting itself (the finish is indicated by a message to the operator), the operator must issue **dltmqbrk** again when the queue manager has been restarted.

Syntax

```
►►──dltmqbrk── -m QMgrName─────────────────────────────────────────►◄
```

Required parameters

-m *QMgrName*

The name of the queue manager for which the broker function is to be deleted.

Return codes

0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Examples

`dltmqbrk -m exampleQM`	Deletes the broker on exampleQM

---

dspmqbrk (Display broker status)

Purpose

Use the **dspmqbrk** command to display the status of a broker. The status value returned from this command can be one of:

- Starting
- Running
- Stopping (immediate shutdown)
- Quiescing (controlled shutdown)
- Not active
- Ended abnormally

Syntax

```
►►──dspmqbrk──┬────────────────┬──────────────────────────────────────►◄
              └── -m QMgrName──┘
```

Optional parameters

-m *QMgrName*
The name of the queue manager for which the broker status is to be displayed. If you do not specify this parameter, the command applies to the default queue manager.

Return codes

0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Examples

`dspmqbrk`	Displays information about the broker on the default queue manager
`dspmqbrk -m exampleQM`	Displays information about the broker on exampleQM

endmqbrk (End broker function)

Purpose

Use the **endmqbrk** command to stop a broker.

Control information is retained and registrations for publishers and subscribers remain in force. Messages are queued by the queue manager until the broker is restarted using the **strmqbrk** command.

Syntax

```
                ┌─ -c ─┐
▶▶──endmqbrk────┤      ├──┬───────────────┬──────────────────────────────▶◀
                └─ -i ─┘  └─ -m QMgrName ─┘
```

Optional parameters

-c Requests a controlled shutdown. This is the default value.

-i Requests an immediate shutdown. The broker does not attempt any further gets or puts, and backs out any in-flight units-of-work. This might mean that a nonpersistent input message is only published to a subset of subscribers, or lost, depending on the broker configuration parameters. (See the description of *SyncPointIfPersistent* in “Broker configuration parameters” on page 113.)

-m *QMgrName*
The name of the queue manager for which the broker function is to be ended. If you do not specify this parameter, the command is routed to the default queue manager.

Return codes

0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Examples

`endmqbrk`	Stops the broker on the default queue manager with a controlled shutdown
`endmqbrk -i -m exampleQM`	Stops the broker on exampleQM immediately

migmqbrk (Migrate broker to MQSeries Integrator)

Purpose

Use the **migmqbrk** command to migrate an MQSeries Publish/Subscribe broker to an MQSeries Integrator broker. This command is only available on platforms that support MQSeries Integrator Version 2.0 and above. Please make sure that you have applied the necessary CSD to the MQSeries base product before running this command (see “Prerequisites” on page 10).

Please read the “Planning for migration and integration” appendix in the *MQSeries Integrator Version 2.0 Introduction and Planning* book before deciding to migrate. In particular, read the “Product differences” section which outlines the impact that migration will have on your current broker network.

Migration exports the following state to a replacement MQSeries Integrator broker. This broker must reside on the same queue manager as the Publish/Subscribe broker.

Subscriptions
All client subscriptions are exported from all streams except SYSTEM.BROKER.ADMIN.STREAM

Retained publications
All retained publications in MQRFH format are exported from all streams except SYSTEM.BROKER.ADMIN.STREAM

Local publishers
Registrations for all publishers that produce local publications are exported from all streams except SYSTEM.BROKER.ADMIN.STREAM

Related brokers
If the broker is part of a multibroker hierarchy, details of all of its relations are exported. This includes the names of all streams that the broker to be migrated has in common with each relation.

The MQSeries Integrator broker and the MQSeries Publish/Subscribe broker which it is to replace must have been created on the same queue manager. Before you start the migration, the MQSeries Integrator broker must be made ready. See the *MQSeries Integrator Version 2.0 Administration Guide* for guidance on performing the migration.

When migration is complete, the Publish/Subscribe broker will be deleted automatically. Therefore, you are advised to back up the queue manager that hosts the Publish/Subscribe broker before you start the migration. If migration fails, the Publish/Subscribe broker remains operational and you can restart it.

Syntax

```
►►──migmqbrk── -m QMgrName─────────────────────────────────────►◄
```

Optional parameters

-m *QMgrName*
The name of the queue manager for which the broker function is to be migrated. This must match the queue manager that hosts the replacement MQSeries Integrator broker.

Return codes

0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Examples

`migmqbrk -m exampleQM`	Migrates the broker on exampleQM

strmqbrk (Start broker function)

Purpose

Use the **strmqbrk** command to start a broker, either as a restart after an **endmqbrk** command (in which case control information is maintained) or initially.

On MQSeries for Windows NT and Windows 2000, the **strmqbrk** command can be added to the reference command file used when starting a queue manager automatically. See the description of the **scmmqm** command in the *MQSeries System Administration* manual for more information.

Syntax

```
►►──strmqbrk──┬─────────────────────┬──┬───────────────┬──────────────────────►◄
              └─ -p ParentQMgrName ─┘  └─ -m QMgrName ─┘
```

Optional parameters

-p *ParentQMgrName*

The name of the queue manager that provides the parent broker function.

Before you can add a broker to the network, channels in both directions must exist between the queue manager that hosts the new broker, and the queue manager that hosts the parent. See "Adding a broker to a network" on page 121 for more details.

On restart, this parameter is optional. If present, it must be the same as it was when previously specified. If this is the root-node broker, the queue manager specified will become its parent. You cannot specify the name of the parent broker when you use triggering to start a broker.

Once a parent has been specified, it is only possible to change parentage in exceptional circumstances in conjunction with the **clrmqbrk** command.

By changing a root node to become the child of an existing broker, two hierarchies can be joined. This will cause subscriptions to be propagated across the two hierarchies, which now become one. After that, publications will start to flow across them. To ensure predictable results, it is essential that you quiesce all publishing applications at this time. If the changed broker detects a hierarchical error (that is, if the new parent is found also to be a descendant), it will immediately shutdown. The administrator must then use **clrmqbrk** at both the changed broker and the new, false parent to restore the previous status. Note that a hierarchical error is detected by propagating a message up the hierarchy, which can complete only when the relevant brokers and links are available.

-m *QMgrName*

The name of the queue manager for which the broker function is to be started. If you do not specify this parameter, the command is routed to the default queue manager.

Return codes

0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Examples

`strmqbrk -p parentQM`	Starts the broker on the default queue manager specifying that it is a child of the broker on parentQM
`strmqbrk -m exampleQM`	Starts the broker on exampleQM

Chapter 13. Message broker exit

An exit can be configured at the broker to customize publications. This exit can be used, for example, to cause traffic for different streams to be sent along different channels.

The exit is invoked after the broker has decided to send a publication to a particular broker or subscriber, and the exit can modify the publication and message descriptor. You are strongly advised not to change the message descriptor for a publication that is being sent between brokers.

Exits are configured in the queue manager configuration file, qm.ini (described in “Broker configuration stanza” on page 112).

The following topics are discussed in this chapter:

- “Publish/subscribe routing exit” on page 138
- “Writing a publish/subscribe routing exit program” on page 145
- “Compiling a publish/subscribe routing exit program” on page 146
- “Sample routing exit” on page 146

© Copyright IBM Corp. 1998, 2000

Publish/subscribe routing exit

This call definition describes the parameters that are passed to the publish/subscribe routing exit called by the publish/subscribe broker.

Note: No entry point called MQ_PUBSUB_ROUTING_EXIT is actually provided by the broker. This is because the name of the publish/subscribe routing exit is defined by the *RoutingExitPath* parameter in the *Broker* stanza of the queue manager's initialization file qm.ini.

MQ_PUBSUB_ROUTING_EXIT *(ExitParms)*

Parameters

ExitParms (MQPXP) – input/output

Exit parameter block.

This structure contains information relating to the invocation of the exit. The exit sets information in this structure to indicate the destination to which the message should be sent.

Usage notes

1. The function performed by the publish/subscribe routing exit is defined by the provider of the exit. The exit, however, must conform to the rules defined in the associated control block MQPXP.
2. No entry point called MQ_PUBSUB_ROUTING_EXIT is actually provided by the publish/subscribe broker. However, a **typedef** is provided for the name MQ_PUBSUB_ROUTING_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 following example illustrates how this can be used:

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

MQ_PUBSUB_ROUTING_EXIT MyRoutingExit;

void MQENTRY MyRoutingExit(PMQPXP pExitParms)
{
  /* C language statements to perform the function of the exit */
}
```

C invocation

```
exitname (&ExitParms);
```

Declare the parameters as follows:

```
MQPXP  ExitParms;  /* Exit parameter block */
```

Publish/subscribe routing exit parameter structure

The following table summarizes the fields in the structure.

Table 5. Fields in MQPXP

Field	Description	Page
StrucId	Structure identifier	139
Version	Structure version number	140
ExitId	Type of exit	140
ExitReason	Reason for invoking exit	140
ExitResponse	Response from exit	141
Feedback	Feedback code	142
ExitNumber	Exit number	142
ExitUserArea	Exit user area	142
ExitData	Exit data	142
MsgInLength	Length of input message	143
MsgOutLength	Length of output message	143
DestinationType	Type of destination	143
MsgDescPtr	Address of message descriptor (MQMD)	143
MsgInPtr	Address of input message	143
MsgOutPtr	Address of output message	144
StreamName	Name of stream	144
QMgrName	Name of local queue manager	144
DestinationQName	Name of destination queue	144
DestinationQMgrName	Name of destination queue manager	145

The MQPXP structure describes the information that is passed to the publish/subscribe routing exit. The exit is invoked each time a broker sends a publication to a subscriber or to another broker. The exit is also invoked when a stream is initialized or terminated.

| This structure is supported in the following environments: AIX, Compaq Tru64
| UNIX, HP-UX, Linux, Sun Solaris, Windows NT, Windows 2000.

Fields

StrucId (MQCHAR4)
Structure identifier.

The value is:

MQPXP_STRUC_ID
: Identifier for publish/subscribe routing-exit parameter structure.

  For the C programming language, the constant MQPXP_STRUC_ID_ARRAY is also defined; this has the same value as MQPXP_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 is:

MQPXP_VERSION_1
Version-1 publish/subscribe routing-exit parameter structure.

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

MQPXP_CURRENT_VERSION
Current version of publish/subscribe routing-exit parameter structure.

This is an input field to the exit.

ExitId (MQLONG)
Type of exit.

This indicates the type of exit being called. The value is:

MQXT_PUBSUB_ROUTING_EXIT
Publish/subscribe routing exit.

This is an input field to the exit.

ExitReason (MQLONG)
Reason for invoking exit.

This indicates the reason why the exit is being called. Possible values are:

MQXR_INIT
Exit initialization.

This indicates that the exit for the stream identified by the *StreamName* field is being invoked for the first time. It allows the exit to acquire and initialize any resources that it may need (for example: main storage).

MQXR_TERM
Exit termination.

This indicates that the exit for the stream identified by the *StreamName* field is about to be terminated. The exit should free any resources that it may have acquired since it was initialized (for example: main storage).

MQXR_MSG
Process a message.

This indicates that the exit is being invoked to process a message.

This is an input field to the exit.

ExitResponse (MQLONG)
Response from exit.

This is set by the exit to indicate how processing should continue. It must be one of the following:

MQXCC_OK
Continue normally.

This indicates that processing should continue normally. It is valid for all values of *ExitReason*.

When *ExitReason* has the value MQXR_MSG, *DestinationQName* and *DestinationQMgrName* identify the destination to which the message should be sent.

MQXCC_SUPPRESS_FUNCTION
Suppress function.

This indicates that the normal processing of the message should be discontinued. It is valid only when *ExitReason* has the value MQXR_MSG.

The processing performed on the message is determined by the MQRO_DISCARD_MSG option in the *Report* field of the message descriptor of the message:

- If the exit specifies MQRO_DISCARD_MSG, the message is discarded.
- If the exit does not specify MQRO_DISCARD_MSG, the message is placed on the dead-letter queue (undelivered-message queue). If there is no dead-letter queue, or the message cannot be placed successfully on the dead-letter queue:
  - The message is discarded if the *Persistence* field in the message descriptor has the value MQPER_NOT_PERSISTENT and the *DiscardNonPersistentPublication* parameter in the queue manager's initialization file has the value *yes*.
  - In all other cases, the message is retried intermittently.

MQXCC_SUPPRESS_EXIT
Suppress exit.

This indicates that the exit should not be invoked again until termination of the stream. It is valid only when *ExitReason* has the value MQXR_INIT or MQXR_MSG.

The broker processes subsequent messages as if no publish/subscribe routing exit were defined. Processing of the current message (if there is one) continues normally; *DestinationQName* and *DestinationQMgrName* identify the destination to which the current message should be sent.

If any other value is returned by the exit, the broker processes the message as if MQXCC_OK had been specified.

This is an output field from the exit.

ExitResponse2 (MQLONG)
Reserved.

This is a reserved field. The value is zero.

Feedback (MQLONG)
Feedback code.

This is the feedback code to be used if the exit returns MQXCC_SUPPRESS_FUNCTION in the *ExitResponse* field.

On input to the exit, this field always has the value MQFB_NONE. If the exit decides to return MQXCC_SUPPRESS_FUNCTION, the exit should set *Feedback* to the value to be used for the message when the broker places it on the dead-letter queue.

If MQXCC_SUPPRESS_FUNCTION is returned by the exit, but *Feedback* still has the value MQFB_NONE, the following feedback code is used:

MQFB_STOPPED_BY_PUBSUB_EXIT
Message stopped by publish/subscribe routing exit.

This is an input/output field to the exit.

ExitNumber (MQLONG)
Exit number.

This is the sequence number of the exit. The value is one.

This is an input field to the exit.

ExitUserArea (MQBYTE16)
Exit user area.

This is a field that is available for the exit to use. It is initialized to MQXUA_NONE (binary zero) on the first invocation of the exit for the stream, and thereafter any changes made to this field by the exit are preserved across invocations of the exit. The first invocation of the exit is indicated by the *ExitReason* field having the value MQXR_INIT. There is a separate *ExitUserArea* for each stream.

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.

ExitData (MQCHAR32)
Exit data.

This is the fixed exit data defined by the *RoutingExitData* parameter of the *Broker* stanza in the queue manager's initialization file. The data is padded with blanks to the full length of the field. If there is no fixed exit data defined in the initialization file, this field is completely blank.

The length of this field is given by MQ_EXIT_DATA_LENGTH. This is an input field to the exit.

HeaderLength (MQLONG)
Reserved.

This is a reserved field. The value is zero.

This is an input field to the exit.

MsgInLength (MQLONG)
Length of input message data.

This is the length in bytes of the message data passed to the exit. The address of the data is given by *MsgInPtr*.

This is an input field to the exit.

MsgOutLength (MQLONG)
Length of output message data.

This is the length in bytes of the message data returned by the exit. On input to the exit, this field is always zero. On output from the exit, this field is ignored if *MsgOutPtr* is the null pointer. See the description of the *MsgOutPtr* field for information about modifying the message data.

This is an input/output field to the exit.

DestinationType (MQLONG)
Type of destination for message.

This is the type of the destination to which the message is being sent. It is one of the following:

MQDT_APPL
Application.

MQDT_BROKER
Broker.

This is an input field to the exit.

MsgDescPtr (PMQMD)
Address of message descriptor.

This is the address of the message descriptor (MQMD) of the message being processed. The exit can change the contents of the message descriptor, but caution should be exercised. In particular:

- If *DestinationType* has the value MQDT_BROKER, the *CorrelId* field in the message descriptor must *not* be changed.

No message descriptor is passed to the exit if *ExitReason* is MQXR_INIT or MQXR_TERM; in these cases, *MsgDescPtr* is the null pointer.

This is an input field to the exit.

MsgInPtr (PMQVOID)
Address of input message data.

This is the address of a buffer containing the message data that is input to the exit. The contents of this buffer can be modified by the exit; see *MsgOutPtr*.

This is an input field to the exit.

MsgOutPtr (PMQVOID)
Address of output message data.

This is the address of a buffer containing the message data that is output from the exit. On input to the exit, this field is always the null pointer. On output from the exit, if the value is still the null pointer, the broker sends the message specified by *MsgInPtr*, with the length given by *MsgInLength*.

If the exit needs to modify the message data, one of the following procedures should be used:

- If the length of the data does not change, the data can be modified in the buffer addressed by *MsgInPtr*. In this case *MsgOutPtr* and *MsgOutLength* should not be changed.
- If the modified data is shorter than the original data, the data can be modified in the buffer addressed by *MsgInPtr*. In this case *MsgOutPtr* must be set to the address of the input message buffer, and *MsgOutLength* set to the new length of the message data.
- If the modified data is (or may be) longer than the original data, the exit must obtain a buffer of the required size and copy the modified data into it. In this case *MsgOutPtr* must be set to the address of the new buffer, and *MsgOutLength* set to the new length of the message data. The exit is responsible for freeing the buffer on a subsequent invocation of the exit.

  Note: Because *MsgOutPtr* is always the null pointer on input to the exit, the exit must save the address of the buffer it obtains, either in *ExitUserArea*, or in a control block whose address is saved in *ExitUserArea*.

This is an input/output field to the exit.

StreamName (MQCHAR48)
Name of stream.

This is the name of the stream to which the message belongs. The name is padded with blanks to the full length of the field.

The length of this field is given by MQ_OBJECT_NAME_LENGTH. This is an input field to the exit.

QMgrName (MQCHAR48)
Name of local queue manager.

This is the name of the local queue manager. The name is padded with blanks to the full length of the field.

The length of this field is given by MQ_Q_MGR_NAME_LENGTH. This is an input field to the exit.

DestinationQName (MQCHAR48)
Name of destination queue.

This is the name of the queue to which the message is being sent. The name is padded with blanks to the full length of the field. The name can be altered by the exit.

The length of this field is given by MQ_Q_NAME_LENGTH. This is an input/output field to the exit.

DestinationQMgrName (MQCHAR48)
Name of destination queue manager.

This is the name of the queue manager to which the message is being sent. The name is padded with blanks to the full length of the field. The name can be altered by the exit.

The length of this field is given by MQ_Q_MGR_NAME_LENGTH. This is an input/output field to the exit.

C declaration

```
typedef struct tagMQPXP {
  MQCHAR4   StrucId;             /* Structure identifier */
  MQLONG    Version;             /* Structure version number */
  MQLONG    ExitId;              /* Type of exit */
  MQLONG    ExitReason;          /* Reason for invoking exit */
  MQLONG    ExitResponse;        /* Response from exit */
  MQLONG    ExitResponse2;       /* Reserved */
  MQLONG    Feedback;            /* Feedback code */
  MQLONG    ExitNumber;          /* Exit number */
  MQBYTE16  ExitUserArea;        /* Exit user area */
  MQCHAR32  ExitData;            /* Exit data */
  MQLONG    HeaderLength;        /* Reserved */
  MQLONG    MsgInLength;         /* Length of input message data */
  MQLONG    MsgOutLength;        /* Length of output message data */
  MQLONG    DestinationType;     /* Type of destination for message */
  PMQMD     MsgDescPtr;          /* Address of message descriptor */
  PMQVOID   MsgInPtr;            /* Address of input message data */
  PMQVOID   MsgOutPtr;           /* Address of output message data */
  MQCHAR48  StreamName;          /* Name of stream */
  MQCHAR48  QMgrName;            /* Name of local queue manager */
  MQCHAR48  DestinationQName;    /* Name of destination queue */
  MQCHAR48  DestinationQMgrName; /* Name of destination queue
                                    manager */
 } MQPXP;
```

Writing a publish/subscribe routing exit program

The MQSeries Publish/Subscribe routing exit is a stream related exit; the parameters (for example, *ExitUserArea*) passed to the exit have the scope of a stream.

The broker uses two threads per stream and the exit can be invoked under either thread. The broker does not call the exit for a single stream under two threads concurrently (that is, the exit does not need to serialize access to the *ExitUserArea* or other stream related data).

| If the exit uses thread related resources (for example, a connection handle or queue handle) the exit must manage these resources on a thread basis. The connection handle obtained by a thread is not usable by any other thread. The exit can use operating system thread services such as **pthread_set_specific** and **pthread_get_specific** on Unix, or **TlsSetValue** and **TlsGetValue** on Windows NT and Windows 2000 to manage thread related resources.

The routing exit is called before a broker sends a publication to a subscriber or another broker. It is also called at initialization and termination of a stream.

Limitations on MQSeries work done in the routing exit

When writing routing exit programs, be aware of the following restrictions on MQI calls:

- Do not issue **MQDISC**.
- Do not issue **MQCMIT** or **MQBACK** within the exit:
  - If you are using *SyncPointIfPersistent*=yes (described in “Broker configuration stanza” on page 112), do not take recoverable action within the exit when processing nonpersistent messages.
  - If you are using *SyncPointIfPersistent*=no, or persistent messages, the exit is invoked within the scope of the publication unit of work.

Security considerations

If the routing exit changes the destination queue or queue manager name, by default no new authority check is carried out.

Compiling a publish/subscribe routing exit program

The routing exit is a dynamically loaded library; it can be thought of as a channel-exit. For information on writing and compiling channel-exit programs see the *MQSeries Intercommunication* book.

Sample routing exit

A sample routing exit program is provided with MQSeries Publish/Subscribe (see Chapter 9, “Sample programs” on page 101). The exit sample is invoked using the *RoutingExitPath* in the `Broker` stanza of queue manager initialization file (see “Broker configuration stanza” on page 112).

The sample program changes either the destination queue or queue manager, depending upon the parameters supplied, as follows:

- If the destination of the message is an application, and the stream name is the default stream:
  - If the destination queue name is Q1, change it to Q2
  - If the destination queue name is Q2, change it to Q3
  - If the destination queue name is Q3, change it to Q4
- If the destination of the message is a broker, and the stream name is `MY.ROUTING.STREAM`:
  - If the destination queue manager is queue manager 1, change it to queue manager 2.
  - If the destination queue manager is queue manager 2, change it to queue manager 3.
  - If the destination queue manager is queue manager 3, change it to queue manager 4.

Part 4. System programming

Chapter 14. Writing system management applications 149
Format of broker administration messages 150
  Subscription deregistered message 150
  Stream deleted message 150
  Broker deleted message 151
  Stream support messages 151
  Children messages 151
  Parent messages 151
MQCFH - PCF header 152
  Reason codes returned from publish/subscribe messages 153
PCF Command Messages 155
  Delete Publication 155
  Deregister Publisher 155
  Deregister Subscriber 156
  Publish 156
  Register Publisher 157
  Register Subscriber 157
  Request Update 158

Chapter 15. Finding out about other publishers and subscribers 159
Metatopics 160
Subscribing to metatopics 161
  Using wildcards 161
  Example requests 161
Authorized metatopics 162
Finding out about brokers 162
Message format for metatopics 162
  Parameters 163
Sample program for administration information 166
  Operation 166
  Example of metatopic information 167

© Copyright IBM Corp. 1998, 2000

Chapter 14. Writing system management applications

Brokers communicate with their neighbors in the hierarchy to establish the topology, and to inform their neighbors about the streams they support. They do this by publishing broker administration messages, as retained messages, using the MQSeries Programmable Command Format (PCF).

Please note that the format of administration information (including metatopics) may be changed in future products.

A PCF message starts with an MQCFH structure, which includes a definition of the type of command the message represents. This is followed by a succession of MQCFIN (integer parameter) and MQCFST (string parameter) structures. The PCF format is described in the *MQSeries Programmable System Management* manual. The MQSeries administration interface (MQAI) has been provided to help you write PCF applications. It is described in the *MQSeries Administration Interface Programming Guide and Reference* manual (SC34-5390).

The SYSTEM.BROKER.ADMIN.STREAM queue is used for broker administration messages. System management applications can subscribe to these messages, provided that they have the correct security authorization. Subscription requests for these topics are sent to the SYSTEM.BROKER.CONTROL.QUEUE in the normal way.

Topics starting 'MQ/' are reserved for MQSeries use, but other topics can be defined. The broker passes these publications to subscribers in the same way as for other streams.

Brokers publish on the 'MQ/*QMgrName*/Children' and 'MQ/*QMgrName*/Parent' topics if applicable. This enables applications to build a view of the broker topology.

The 'MQ/*QMgrName*/StreamSupport' topic is published on by all brokers. This enables applications to build a view of the stream topology in relation to the broker topology.

Brokers also publish messages to this queue when a stream or broker has been deleted, and when a subscription has been deregistered by the broker because it is no longer valid.

This chapter discusses the following topics:

- "Format of broker administration messages" on page 150
- "MQCFH - PCF header" on page 152
- "PCF Command Messages" on page 155

Metatopics are published on the stream to which they relate so the relevant ones are published on SYSTEM.BROKER.ADMIN.STREAM. For information about metatopics see "Metatopics" on page 160.

© Copyright IBM Corp. 1998, 2000

Format of broker administration messages

The broker sends administration messages as **Publish** messages in PCF format. The following parameters are always present:

PublicationOptions (MQCFIN)
MQPUBO_RETAIN_PUBLICATION is set if the publication is retained.

StreamName (MQCFST)
Set to the reserved stream name 'SYSTEM.BROKER.ADMIN.STREAM'.

Topic (MQCFST)
This is one of the following:

- 'MQ/*QMgrName*/Event/SubscriptionDeregistered'
- 'MQ/*QMgrName*/Event/StreamDeleted'
- 'MQ/*QMgrName*/Event/BrokerDeleted'
- 'MQ/*QMgrName*/StreamSupport'
- 'MQ/*QMgrName*/Children'
- 'MQ/*QMgrName*/Parent'

where *QMgrName* is the queue manager name of the broker sending the message (this is 48 characters long padded with blanks if necessary).

PublishTimestamp (MQCFST)
Set to the time of publication (Universal time).

Subscription deregistered message

An 'MQ/*QMgrName*/Event/SubscriptionDeregistered' message is published when a subscription is deregistered by the broker because it has become invalid (for example, it is no longer authorized).

For 'MQ/*QMgrName*/Event/SubscriptionDeregistered' messages, the following group of parameters is published to identify the subscription which has been removed by the broker.

- *RegistrationStreamName*
- *RegistrationTopic*
- *RegistrationQMgrName*
- *RegistrationQName*
- *RegistrationCorrelId* (if applicable)
- *RegistrationUserIdentifier*
- *RegistrationRegistrationOptions*

These additional parameters are described in "Message format for metatopics" on page 162.

Stream deleted message

An 'MQ/*QMgrName*/Event/StreamDeleted' message is published when a stream is deleted. The following additional parameter is present:

RegistrationStreamName (MQCFST)
Name of deleted stream (parameter identifier: MQCACF_REG_STREAM_NAME).

Broker deleted message

When a broker is deleted with the *dltmqbrk* command, it publishes an ‘MQ/*QMgrName*/Event/BrokerDeleted’ message.

The administrator is advised to stop affected application programs before making changes to broker network and stream topology. However, a program could be written to subscribe to these administrative event topics and take appropriate action. In the case of the BrokerDeleted event, such a program cannot rely on this message being propagated to the parent, but the program will receive the message if it has subscribed to this topic at the affected broker.

Stream support messages

An ‘MQ/*QMgrName*/StreamSupport’ message (a retained publication) gives information about which streams the broker supports. The following parameter is repeated for each stream supported:

SupportedStreamName (MQCFST)
Name of supported stream (parameter identifier: MQCACF_SUPPORTED_STREAM_NAME).

Children messages

An ‘MQ/*QMgrName*/Children’ message (a retained publication) gives information about a broker’s children. It is published only by those brokers that have children. The following parameter is repeated for each child:

QMgrName (MQCFST)
Queue manager name of child broker (parameter identifier: MQCACF_CHILD_Q_MGR_NAME).

This list gives all of the broker’s immediate children in the hierarchy.

Parent messages

An ‘MQ/*QMgrName*/Parent’ message (a retained publication) gives information about a broker’s parent. It is published only by those brokers that have a parent. The following parameter occurs once:

QMgrName (MQCFST)
Queue manager name of parent broker (parameter identifier: MQCACF_PARENT_Q_MGR_NAME).

MQCFH - PCF header

Each message or response in PCF format starts with an MQCFH structure. The field contents of the MQCFH structure for MQSeries Publish/Subscribe are as follows:

Type (MQLONG)
Structure type.

The following values are valid:

MQCFT_COMMAND
Command message (for example, Publish, Register Subscribers).

MQCFT_RESPONSE
Message is a response to a command.

StrucLength (MQLONG)
Structure length. The value must be MQCFH_STRUC_LENGTH.

Version (MQLONG)
Structure version number. The value must be MQCFH_VERSION_1.

Command (MQLONG)
Command identifier.

For a command message, this identifies the function to be performed. For a response message, it identifies the command to which this is the reply. The following values are valid:

MQCMD_DELETE_PUBLICATION	Delete Publication
MQCMD_DEREGISTER_PUBLISHER	Deregister Publisher
MQCMD_DEREGISTER_SUBSCRIBER	Deregister Subscriber
MQCMD_PUBLISH	Publish
MQCMD_REGISTER_PUBLISHER	Register Publisher
MQCMD_REGISTER_SUBSCRIBER	Register Subscriber
MQCMD_REQUEST_UPDATE	Request Update
MQCMD_BROKER_INTERNAL	Used internally by brokers

MsgSeqNumber (MQLONG)
Message sequence number. The value must be 1 for MQSeries Publish/Subscribe messages and responses.

Control (MQLONG)
Control options.

The value must be MQCFC_LAST for MQSeries Publish/Subscribe messages and responses.

CompCode (MQLONG)
Completion code.

This field is meaningful only for a response; its value is not significant for a command. The following values are possible:

MQCC_OK
Command completed successfully.

MQCC_WARNING
Command completed with warning.

MQCC_FAILED
Command failed.

Reason (MQLONG)
Reason code qualifying completion code.

This field is meaningful only for a response; its value is not significant for a command.

The reason codes that might be returned in response to a command are listed in “Reason codes returned from publish/subscribe messages.”

ParameterCount (MQLONG)
Count of parameter structures (MQCFIN, MQCFST) following.

The value of this field is zero or greater.

Reason codes returned from publish/subscribe messages

The following reason codes can be returned by a broker in response to any command message in PCF format. They are described in the *MQSeries Programmable System Management* manual.

MQRCCF_CFH_COMMAND_ERROR	Command identifier not valid.
MQRCCF_CFH_CONTROL_ERROR	Control option not valid.
MQRCCF_CFH_LENGTH_ERROR	Structure length not valid.
MQRCCF_CFH_MSG_SEQ_NUMBER_ERROR	Message sequence number not valid.
MQRCCF_CFH_PARM_COUNT_ERROR	Parameter count not valid.
MQRCCF_CFH_TYPE_ERROR	Type not valid.
MQRCCF_CFH_VERSION_ERROR	Structure version number not valid.
MQRCCF_CFIN_DUPLICATE_PARM	Duplicate MQCFIN parameter.
MQRCCF_CFIN_LENGTH_ERROR	MQCFIN structure length not valid.
MQRCCF_CFIN_PARM_ID_ERROR	Parameter identifier not valid.
MQRCCF_CFST_DUPLICATE_PARM	Duplicate MQCFST parameter.
MQRCCF_CFST_LENGTH_ERROR	MQCFST structure length not valid.
MQRCCF_CFST_PARM_ID_ERROR	Parameter identifier not valid.
MQRCCF_CFST_STRING_LENGTH_ERR	MQCFST string length not valid.
MQRCCF_COMMAND_FAILED	Command failed.
MQRCCF_ENCODING_ERROR	Encoding error.
MQRCCF_INCORRECT_Q	Command sent to wrong broker queue.
MQRCCF_MD_FORMAT_ERROR	Format not valid.
MQRCCF_MSG_LENGTH_ERROR	Message length not valid.
MQRCCF_PARM_COUNT_TOO_SMALL	Mandatory parameter for command missing.
MQRCCF_STRUCTURE_TYPE_ERROR	Structure type invalid.

The following reason codes might be returned by a broker in response to a command message in PCF format, depending on the parameters used in that message. They are described in Appendix A, “Reason codes” on page 173.

MQRCCF_CORREL_ID_ERROR	Correlation identifier used as part of identity but is all binary zero.
MQRCCF_DEL_OPTIONS_ERROR	Invalid delete options supplied.
MQRCCF_DUPLICATE_IDENTITY	Publisher or subscriber identity already assigned to another user ID.
MQRCCF_INCORRECT_STREAM	Stream name different from queue name.
MQRCCF_NO_RETAINED_MSG	No retained message exists for this topic.
MQRCCF_NOT_AUTHORIZED	Subscriber not authorized to browse broker's stream queue or subscriber queue.
MQRCCF_NOT_REGISTERED	Publisher or subscriber not registered.
MQRCCF_PUB_OPTIONS_ERROR	Invalid publication options supplied.
MQRCCF_Q_MGR_NAME_ERROR	Queue manager name invalid.
MQRCCF_Q_NAME_ERROR	Queue name invalid.
MQRCCF_REG_OPTIONS_ERROR	Invalid registration options supplied.
MQRCCF_STREAM_ERROR	Stream name too long or contains invalid characters.
MQRCCF_TOPIC_ERROR	Topic name has an invalid length or contains invalid characters.
MQRCCF_UNKNOWN_STREAM	Stream not defined to broker and cannot be created.

PCF Command Messages

This section lists the parameters and options that are relevant for each command message in PCF format. Parameter identifiers and types (MQCFIN or MQCFST) are shown. Broker administration and metatopic messages use the PCF format.

The usage of each parameter and option is the same as for the corresponding command messages in RFH format, which are described in Chapter 7, "Publish/Subscribe command messages" on page 71. In principle, publishers and subscribers could send command messages to a broker in PCF format. **This is not recommended.** You should use the RFH format for command messages to ensure interoperability with other MQSeries business integration functions.

Delete Publication

Topic (MQCFST)
Topic (parameter identifier: MQCACF_TOPIC).

DeleteOptions (MQCFIN)
Delete options (parameter identifier: MQIACF_DELETE_OPTIONS).

The following option can be set:

MQDELO_LOCAL

StreamName (MQCFST)
Stream name (parameter identifier: MQCACF_STREAM_NAME).

Deregister Publisher

RegistrationOptions (MQCFIN)
Registration options (parameter identifier: MQIACF_REGISTRATION_OPTIONS).

The following options can be set:

MQREGO_DEREGISTER_ALL
MQREGO_CORREL_ID_AS_IDENTITY

StreamName (MQCFST)
Stream name (parameter identifier: MQCACF_STREAM_NAME).

Topic (MQCFST)
Topic (parameter identifier: MQCACF_TOPIC).

QMgrName (MQCFST)
Publisher's queue manager name (parameter identifier: MQCA_Q_MGR_NAME).

QName (MQCFST)
Publisher's queue name (parameter identifier: MQCA_Q_NAME).

Deregister Subscriber

RegistrationOptions (MQCFIN)
Registration options (parameter identifier: MQIACF_REGISTRATION_OPTIONS).

The following options can be set:

MQREGO_DEREGISTER_ALL
MQREGO_CORREL_ID_AS_IDENTITY

StreamName (MQCFST)
Stream name (parameter identifier: MQCACF_STREAM_NAME).

Topic (MQCFST)
Topic (parameter identifier: MQCACF_TOPIC).

QMgrName (MQCFST)
Subscriber's queue manager name (parameter identifier: MQCA_Q_MGR_NAME).

QName (MQCFST)
Subscriber's queue name (parameter identifier: MQCA_Q_NAME).

Publish

Topic (MQCFST)
Topic (parameter identifier: MQCACF_TOPIC).

RegistrationOptions (MQCFIN)
Registration options (parameter identifier: MQIACF_REGISTRATION_OPTIONS).

The following options can be set:

MQREGO_ANONYMOUS
MQREGO_LOCAL
MQREGO_DIRECT_REQUESTS
MQREGO_CORREL_ID_AS_IDENTITY

PublicationOptions (MQCFIN)
Publication options (parameter identifier: MQIACF_PUBLICATION_OPTIONS).

The following options can be set:

MQPUBO_NO_REGISTRATION
MQPUBO_RETAIN_PUBLICATION
MQPUBO_IS_RETAINED_PUBLICATION
MQPUBO_OTHER_SUBSCRIBERS_ONLY
MQPUBO_CORREL_ID_AS_IDENTITY

StreamName (MQCFST)
Stream name (parameter identifier: MQCACF_STREAM_NAME).

QMgrName (MQCFST)
Publisher's queue manager name (parameter identifier: MQCA_Q_MGR_NAME).

QName (MQCFST)
Publisher’s queue name (parameter identifier: MQCA_Q_NAME).

PublishTimestamp (MQCFST)
Publication timestamp (parameter identifier: MQCACF_PUBLISH_TIMESTAMP).

SequenceNumber (MQCFIN)
Publication sequence number (parameter identifier: MQIACF_SEQUENCE_NUMBER).

StringData (MQCFST)
String publication data (parameter identifier: MQCACF_STRING_DATA).

IntegerData (MQCFIN)
Integer publication data (parameter identifier: MQIACF_INTEGER_DATA).

Register Publisher

Topic (MQCFST)
Topic (parameter identifier: MQCACF_TOPIC).

RegistrationOptions (MQCFIN)
Registration options (parameter identifier: MQIACF_REGISTRATION_OPTIONS).

The following options can be set:

MQREGO_ANONYMOUS
MQREGO_LOCAL
MQREGO_DIRECT_REQUESTS
MQREGO_CORREL_ID_AS_IDENTITY

StreamName (MQCFST)
Stream name (parameter identifier: MQCACF_STREAM_NAME).

QMgrName (MQCFST)
Publisher’s queue manager name (parameter identifier: MQCA_Q_MGR_NAME).

QName (MQCFST)
Publisher’s queue name (parameter identifier: MQCA_Q_NAME).

Register Subscriber

Topic (MQCFST)
Topic (parameter identifier: MQCACF_TOPIC).

RegistrationOptions (MQCFIN)
Registration options (parameter identifier: MQIACF_REGISTRATION_OPTIONS).

The following options can be set:

MQREGO_ANONYMOUS
MQREGO_LOCAL
MQREGO_NEW_PUBLICATIONS_ONLY
MQREGO_PUBLISH_ON_REQUEST_ONLY
MQREGO_CORREL_ID_AS_IDENTITY

MQREGO_INCLUDE_STREAM_NAME
MQREGO_INFORM_IF_RETAINED

StreamName (MQCFST)
Stream name (parameter identifier: MQCACF_STREAM_NAME).

QMgrName (MQCFST)
Subscriber's queue manager name (parameter identifier: MQCA_Q_MGR_NAME).

QName (MQCFST)
Subscriber's queue name (parameter identifier: MQCA_Q_NAME).

Request Update

Topic (MQCFST)
Topic (parameter identifier: MQCACF_TOPIC).

RegistrationOptions (MQCFIN)
Registration options (parameter identifier: MQIACF_REGISTRATION_OPTIONS).

The following option can be set:

MQREGO_CORREL_ID_AS IDENTITY

StreamName (MQCFST)
Stream name (parameter identifier: MQCACF_STREAM_NAME).

QMgrName (MQCFST)
Subscriber's queue manager name (parameter identifier: MQCA_Q_MGR_NAME).

QName (MQCFST)
Subscriber's queue name (parameter identifier: MQCA_Q_NAME).

Chapter 15. Finding out about other publishers and subscribers

Brokers publish information about the publishers and subscribers that are registered with them. The information is published as a special set of topics, known as *metatopics*, within each supported stream. They are published as persistent messages, and use the default priority for the stream queue (at the last time the broker started).

Applications can subscribe to this information in the same way as they can register any other subscription. Whenever the information changes, brokers publish the changed information in the form of retained publications so that new subscribers to it will receive the current state.

Metatopic command messages can be sent to a broker using the MQSeries Programmable Command Format (PCF), which is described in Chapter 14, “Writing system management applications” on page 149. Publications containing the metatopic data will be sent in PCF format, as will any broker response messages. This is illustrated in the “Sample program for administration information” on page 166.

Alternatively, metatopic command messages can be sent with the Rules and Formatting header (RFH), which is described in Chapter 6, “Format of command messages” on page 61. In this case any broker response messages will be in RFH format, but publications containing the metatopic data will be sent in PCF format.

This chapter discusses the following topics:

- “Metatopics” on page 160
- “Subscribing to metatopics” on page 161
- “Authorized metatopics” on page 162
- “Finding out about brokers” on page 162
- “Message format for metatopics” on page 162

© Copyright IBM Corp. 1998, 2000

Metatopics

Brokers publish information about the publishers and subscribers that are registered with them. The information is published as a special set of topics, known as *metatopics*, within each supported stream.

Each broker publishes on metatopics to each stream to describe the publishers, subscribers and topics on that stream. Metatopics include subscribers to metatopics. All metatopic publications are global.

Metatopics always begin with ‘MQ/’, and topics starting with ‘MQ/’ are reserved for all streams. These metatopic strings are of the form:

- ‘MQ/S/*QMgrName*/Publishers/Topics’
- ‘MQ/S/*QMgrName*/Publishers/Summary’
- ‘MQ/S/*QMgrName*/Publishers/Summary/*Topic*’
- ‘MQ/S/*QMgrName*/Publishers/Identities’
- ‘MQ/S/*QMgrName*/Publishers/Identities/*Topic*’
- ‘MQ/SA/*QMgrName*/Publishers/AllIdentities’
- ‘MQ/SA/*QMgrName*/Publishers/AllIdentities/*Topic*’

- ‘MQ/S/*QMgrName*/Subscribers/Topics’
- ‘MQ/S/*QMgrName*/Subscribers/Summary’
- ‘MQ/S/*QMgrName*/Subscribers/Summary/*Topic*’
- ‘MQ/S/*QMgrName*/Subscribers/Identities’
- ‘MQ/S/*QMgrName*/Subscribers/Identities/*Topic*’
- ‘MQ/SA/*QMgrName*/Subscribers/AllIdentities’
- ‘MQ/SA/*QMgrName*/Subscribers/AllIdentities/*Topic*’

Where:

- *QMgrName* is the name of the broker’s queue manager. This is 48 characters long padded with blanks if necessary.
- *Topic* is any topic for which the broker has a registered publisher or subscriber (depending on whether the subscription is for publishers or subscribers).

Metatopics that do not include *Topic* each represent a single metatopic (for one broker), so a broker receiving a **Register Subscriber** message for one of these metatopics generates one retained **Publish** message as a result (additional retained **Publish** messages are generated whenever the information changes). However, for metatopics that do include *Topic*, one retained **Publish** message is generated for each registered topic that matches the *Topic* specification (and again further messages are generated as the information changes).

The strings in the fifth part of the metatopic offer varying levels of detail, as follows:

Summary
: Minimal information including counts. If *Topic* is included, one message is generated for each matching topic.

Topics
: A list of registered topics in a single message.

Identities
: Identities of publishers or subscribers, including user ID and time of registration. If *Topic* is included, one message is generated for each matching topic, otherwise all identities are packaged into a single message.

Anonymous publishers or subscribers are not included (this means that no message is generated for topics that have only anonymous publishers and subscribers registered).

AllIdentities

This is the equivalent of *Identities* for authorized metatopics (see “Authorized metatopics” on page 162) and gives the same information, but also includes anonymous publishers and subscribers.

If an application subscribes to an ‘AllIdentities’ metatopic, the application requires **altusr** authority for the queue manager, as well as the normal **browse** authority for that stream queue.

Subscribing to metatopics

Applications should use this facility carefully because it can produce a large amount of data. Applications using metatopics are recommended to register subscriptions with each broker individually (using the SYSTEM.BROKER.ADMIN.STREAM to determine the queue manager names of the brokers). These applications are recommended to subscribe only to the information from that broker and set the MQREGO_PUBLISH_ON_REQUEST option in the **Register Subscriber** message and use **Request Update** to minimize network traffic.

Using wildcards

Metatopics describing subscribers include information about wildcard subscriptions. This is an exception to the rule that publications should not include wildcards in their topics.

In subscriptions to metatopics, wildcards can be used in the usual way. For example, if the metatopic ‘MQ/S/QM1/Publishers/S*’ is specified, this matches ‘MQ/S/QM1/Publishers/Summary’ plus all of the ‘MQ/S/QM1/Publishers/Summary/*Topic*’ metatopics (one for each topic registered implicitly or explicitly for publishers, except for those published anonymously), and the broker sends this number of retained **Publish** messages as a result.

If the metatopic ‘MQ/S/QM1/Subscribers/S*’ is specified, the resultant messages show all the topics registered for subscribers (except for those registered anonymously), including wildcard subscriptions. (The wildcard characters in metatopics only match wildcard subscriptions.)

A wildcard subscription of the form ‘*’ will give all topics on a stream except for the metatopics. You need to specify at least the first five characters (‘MQ/S/’) to receive publications about metatopics.

Example requests

The following examples show valid metatopic requests.

- To find out what topics are being published on QM22 in a single **Publish** message:

  `MQ/S/QM22/Publishers/Topics`

  No information is returned about publishers’ identities.

- To find the identities of each subscriber (except anonymous subscribers) on all brokers in the network, for any topics starting with ‘Trade/’:

  `MQ/S/*/Subscribers/Identities/Trade/*`

  One **Publish** message is generated for each matching topic, by each broker. Requesting this much information could have an adverse effect on the performance of your system.

Authorized metatopics

There is a subclass of metatopics, called *authorized metatopics*, that are only available to users with **altusr** authority for that queue manager. These show the identities of all publishers and subscribers including the anonymous ones. Subscribers (who must be authorized) will only receive authorized metatopics by specifying at least the first six characters ‘MQ/SA/’. A wildcard subscription of the form ’MQ/S*’ will give no metatopics at all, ‘MQ/SA/*’ will give all the authorized metatopics and ‘MQ/S/*’ will give all the others.

Finding out about brokers

To find out about all brokers, a subscriber can specify a `Topic` parameter of, say, ‘MQ/S/*/Publishers/Summary’. If no `StreamName` parameter is specified, this defaults to the default stream, which all brokers support. At least one message will be received from each broker that is connected. More than one message might be received from a broker if the state changes. For efficiency, however, it is recommended to register a subscription with each broker individually.

To determine which brokers support a particular stream, the program can issue the **Register Subscriber** command to the SYSTEM.BROKER.ADMIN.STREAM at its local broker and specify an appropriate StreamSupport topic.

Message format for metatopics

These messages are sent as **Publish** messages in PCF format with MQPUBO_RETAIN_PUBLICATION (for ongoing subscriptions registered with **Register Subscriber**). In these messages, `Command` is MQCMD_PUBLISH, and `Type` is MQCFT_COMMAND.

The following table summarizes which parameters are included for which metatopics. An explanation of each parameter follows the table.

Table 6. Parameters for publisher and subscriber information messages

	Topics	**Summary**	**Summary /<Topic>**	**Identities** [1]	**Identities /<Topic>** [1]
Number of messages sent	1	1	1 per topic	1	1 per topic
StreamName	Y	Y	Y	Y	Y
Topic	Y	Y	Y	Y	Y
PublishTimestamp	Y	Y	Y	Y	Y
BrokerCount	Y	Y	Y	Y	Y
ApplCount	Y	Y	Y	Y	Y
AnonymousCount	Y	Y	Y	Y	Y
RegistrationTopic	Y [2]	N	N [3]	N	N [3]
RegistrationQMgrName	N	N	N	Y	Y
RegistrationQName	N	N	N	Y	Y
RegistrationCorrelId	N	N	N	Y	Y
RegistrationUserIdentifier	N	N	N	Y	Y
RegistrationRegistrationOptions	N	N	N	N	Y
RegistrationTime	N	N	N	N	Y

Notes:

[1] ‘AllIdentities’ subscriptions are the same except that they include anonymous as well as non-anonymous publishers and subscribers.

[2] Repeated for each registered topic.

[3] *Topic* parameter contains the registered topic.

Parameters

These parameters might be included in publisher and subscriber information messages sent by the broker. Table 6 summarizes the parameters that are used for each metatopic.

StreamName (MQCFST)
Stream Name (parameter identifier: MQCACF_STREAM_NAME).

Name of the stream for which this information applies.

Topic (MQCFST)
Topic (parameter identifier: MQCACF_TOPIC).

The metatopic under which this publication is published. These are listed in “Metatopics” on page 160.

PublishTimestamp (MQCFST)
Time this **Publish** message was generated (parameter identifier: MQCACF_PUBLISH_TIMESTAMP).

This is of length 16 characters, in the format `YYYYMMDDHHMMSSTH`, using Universal Time.

BrokerCount (MQCFIN)
Number of broker publishers or subscribers (parameter identifier: MQIACF_BROKER_COUNT).

Count of publisher or subscriber registrations from brokers, for the specified topic if this is a 'MQ/*QMgrName*/.../*Topic*' message.

For publishers, this count is normally zero, as brokers do not register as publishers. The role of a broker in acting as a publisher itself for metatopics on stream queues is not counted, nor is its role as a publisher for administrative topics on the SYSTEM.BROKER.ADMIN.STREAM stream.

ApplCount (MQCFIN)
Number of application publishers or subscribers (parameter identifier: MQIACF_APPL_COUNT).

Count of publisher or subscriber registrations from applications, for the specified topic if this is a 'MQ/S/*QMgrName*/.../*Topic*' or 'MQ/SA/*QMgrName*/.../*Topic*' message. The latter includes anonymous registrations.

AnonymousCount (MQCFIN)
Number of anonymous publishers or subscribers (parameter identifier: MQIACF_ANONYMOUS_COUNT).

Count of anonymous publisher or subscriber registrations from applications, for the specified topic if this is a 'MQ/SA/*QMgrName*/.../*Topic*' message.

RegistrationTopic (MQCFST)
Topic (parameter identifier: MQCACF_REG_TOPIC).

A topic for which at least one publisher or subscriber is registered. Wildcards are not present for publishers, but might be for subscribers.

This parameter is repeated for as many topics as necessary for 'MQ/S/*QMgrName*/Publishers/Topics' and 'MQ/S/*QMgrName*/Subscribers/Topics' messages. Each topic is present only once, even if there are several publishers or subscribers registered for the same topic.

RegistrationQMgrName (MQCFST)
Publisher's or subscriber's queue manager name (parameter identifier: MQCACF_REG_Q_MGR_NAME).

RegistrationQName (MQCFST)
Publisher's or subscriber's queue name (parameter identifier: MQCACF_REG_Q_NAME).

RegistrationCorrelId (MQCFST)
Publisher's or subscriber's correlation identifier (parameter identifier: MQCACF_REG_CORREL_ID).

This is a 48-byte character string of hexadecimal characters representing the contents of the 24-byte binary correlation identifier. Each character in the string is in the range 0 through 9 or A through F.

This parameter is present only if the publisher's or subscriber's identity includes a correlation identifier.

RegistrationUserIdentifier (MQCFST)
Publisher's or subscriber's user ID (parameter identifier: MQCACF_REG_USER_ID).

RegistrationRegistrationOptions (MQCFST)
Publisher's or subscriber's registration options (parameter identifier: MQIACF_REG_REG_OPTIONS).

RegistrationOptions parameter as specified (or defaulted) by the publisher or subscriber when it registered.

RegistrationTime (MQCFST)
Registration time (parameter identifier: MQCACF_REG_TIME).

This is of length 16 characters, in the format `YYYYMMDDHHMMSSTH`, using Universal Time.

Sample program for administration information

The sample administration program will attach to a broker, subscribe to the appropriate streams to obtain the required metatopic information, and then detach from the broker. The following *RegistrationOptions* will be used:

MQREGO_ANONYMOUS
MQREGO_PUBLISH_ON_REQUEST_ONLY

The information listed below can be dumped into a file or to standard output.

1. The parent and children for the broker.
2. All the streams supported at the broker (unless overridden by the **-s** option).
3. All the subscribers and publishers registered for these streams (unless overridden by the **-p** or **-u** options), with the following parameters:

   StreamName
   Topic (max 255 chars)
   BrokerCount
   ApplCount
   AnonymousCount
   RegistrationQMgrName
   RegistrationQName
   RegistrationCorrelId
   RegistrationUserIdentifier
   RegistrationRegistrationOptions
   RegistrationTime

4. All retained messages at the broker for the given topic (if and only if the **-r** option is set), with the following parameters:

   StreamName
   Topic (max 255 chars)
   StringData (max 255 chars, PCF only)
   IntegerData (PCF only)
   QMgrName
   QName
   SequenceNumber
   PublishTimestamp
   Expiry

Operation

To run the sample administration program, first run `amqspsda.tst` on the queue manager. Then enter the following:

```
amqspsd options
```

where *options* are:

-l *LogFileName*
The name of the log file that the information is sent to.

The default is that output will be sent to the screen (`stdio`).

-m *QMgrName*
The queue manager name.

The default is that the default queue manager will be used.

-q *QName*
The name of the queue which is subscribed to.

The default is that the program will attempt to create a permanent-dynamic queue based on AMQSPSDA.PERMDYN.MODEL.QUEUE. This queue will be deleted at program termination.

-s *StreamName*
The stream name.

The default is that all streams will be dumped.

-t *Topic*
The topic.

The default is that * (all topics) will be used as the topic.

-r *Topic*
Dump retained messages for this topic (* can be used for all topics).

The default is not to dump retained messages.

-p
Dump information for publishers only.

The default is to dump information for publishers and subscribers.

-u
Dump information for subscribers only.

The default is to dump information for publishers and subscribers.

-a
Dump information for anonymous publishers and subscribers.

The default is to dump information for non-anonymous publishers and subscribers.

On successful termination, zero will be returned to any calling application.

Example of metatopic information

Here is an example of output from the sample administration program, which was obtained from a newly created broker using:

```
amqspsd -m PubSub -r *
```

It shows two retained messages, one in RFH and one in PCF format.

```
MQSeries Message Broker Dumper
Start time Wed-18-Nov-1998 10:35:31

Broker Hierarchy
----------------
QMgrName:
   PubSub
Parent:
   None
Children:
   None

Streams supported
-----------------
SYSTEM.BROKER.DEFAULT.STREAM
SYSTEM.BROKER.ADMIN.STREAM

Publishers
----------
StreamName: SYSTEM.BROKER.ADMIN.STREAM
   None
StreamName: SYSTEM.BROKER.DEFAULT.STREAM
   Topic: Topic 3
      BrokerCount: 0
      ApplCount: 1
      AnonymousCount: 0
      RegistrationQMgrName: PubSub
      RegistrationQName: Q2
      RegistrationUserIdentifier: hgdd
      RegistrationRegistrationOptions: 0 : MQREGO_NONE
      RegistrationTime: 1998111810350435
   Topic: Topic 2
      BrokerCount: 0
      ApplCount: 1
      AnonymousCount: 0
      RegistrationQMgrName: PubSub
      RegistrationQName: Q2
      RegistrationUserIdentifier: hgdd
      RegistrationRegistrationOptions: 0 : MQREGO_NONE
      RegistrationTime: 1998111810350435
   Topic: Topic 1
      BrokerCount: 0
      ApplCount: 1
      AnonymousCount: 0
      RegistrationQMgrName: PubSub
      RegistrationQName: Q1
      RegistrationUserIdentifier: hgdd
      RegistrationRegistrationOptions: 0 : MQREGO_NONE
      RegistrationTime: 1998111810341148
```

```
Subscribers
-----------
StreamName: SYSTEM.BROKER.ADMIN.STREAM
   Topic: MQ/PubSub                                          /StreamSupport
      BrokerCount: 0
      ApplCount: 1
      AnonymousCount: 0
      RegistrationQMgrName: PubSub
      RegistrationQName: SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS
      RegistrationCorrellId: 414D5159010100000000000000000000000000000000
      RegistrationUserIdentifier: mqm
      RegistrationRegistrationOptions: 17 : MQREGO_CORREL_ID_AS_IDENTITY
                                            MQREGO_NEW_PUBLICATIONS_ONLY
      RegistrationTime: 1998111810330750
   Topic: MQ/S/PubSub                             /Subscribers/Identities/*
      BrokerCount: 0
      ApplCount: 1
      AnonymousCount: 1
StreamName: SYSTEM.BROKER.DEFAULT.STREAM
   Topic: MQ/S/PubSub                             /Subscribers/Identities/*
      BrokerCount: 0
      ApplCount: 1
      AnonymousCount: 1

Retained messages
-----------------
StreamName: SYSTEM.BROKER.DEFAULT.STREAM
   RFH Message
      Expiry: -1
      Topic: Topic 2
      Topic: Topic 3
      QMgrName: PubSub
      QName: Q2
      SequenceNumber: None
      PublishTimestamp: None

      ****   Message   ****   length - 92 bytes

      0000:  0100 0000 2400 0000 0100 0000 0000 0000 '....¢...........'
      0010:  0100 0000 0100 0000 0000 0000 0000 0000 '................'
      0020:  0200 0000 0400 0000 2800 0000 DB0B 0000 '........(...⅞...'
      0030:  0000 0000 1200 0000 4D79 2072 6574 6169 '........My retai'
      0040:  6E65 6420 7374 7269 6E67 0000 0300 0000 'ned string......'
      0050:  1000 0000 3804 0000 15CD 5B07           '....8...."£.    '
   PCF Message
      Expiry: -1
      Topic: Topic 1
      QMgrName: PubSub
      QName: Q1
      SequenceNumber: None
      PublishTimestamp: None
      IntegerData: 123456789
      StringData: My retained string
```

Part 5. Appendixes

© Copyright IBM Corp. 1998, 2000

Appendix A. Reason codes

This appendix describes the MQRCCF_* and MQRC_RFH_* reason codes that can be issued by the broker.

If the code you are looking for is not in this list, see the *MQSeries Programmable System Management* manual for information. The MQRC_* return codes issued by the broker are described in the *MQSeries Application Programming Reference* manual.

MQRCCF_BROKER_DELETED

Broker has been deleted.

When a broker is deleted using the **dltmqbrk** command all broker queues created by the broker will be deleted. Before this can be done the queues are emptied of all command messages; any found are placed on the dead-letter queue with this reason code.

Corrective action: Process the command messages that were placed on the dead-letter queue.

MQRCCF_CORREL_ID_ERROR

Correlation identifier used as part of an identity is all binary zeroes.

Each publisher and subscriber is identified by a queue manager name, a queue name, and optionally a correlation identifier. The correlation identifier is typically used to allow multiple subscribers to share the same subscriber queue. In this instance a publisher or subscriber has indicated within the Registration or Publication options supplied on the command that their identity does include a correlation identifier, but a valid identifier has not been supplied.

Corrective action: Retry the command ensuring that the correlation identifier supplied in the message descriptor of the command message is not all binary zeroes.

MQRCCF_DEL_OPTIONS_ERROR

Invalid delete options have been supplied.

The options provided with a **Delete Publication** command are not valid.

Corrective action: Retry the command with a valid combination of options.

MQRCCF_DUPLICATE_IDENTITY

Publisher or subscriber identity already assigned to another user ID.

Each publisher and subscriber has a unique identity consisting of a queue manager name, a queue name, and optionally a correlation identifier. Associated with each identity is the user ID under which that publisher or subscriber first registered. A given identity can only be assigned to one user ID at a time. While the identity is registered with the broker all commands wanting to use it must specify the correct user ID. When a publisher or a subscriber no longer has any registrations with the broker the identity can be used by another user ID.

Corrective action: Either retry the command using a different identity or remove all registrations associated with the identity so that it can be used by a different user ID. The user ID to which the identity is currently

© Copyright IBM Corp. 1998, 2000

assigned is returned within the error response message. A **Deregister** command could be issued to remove these registrations. If the user ID in question cannot be used to execute such a command, you will need to have the necessary authority to open the SYSTEM.BROKER.CONTROL.QUEUE using the MQOO_ALTERNATE_USER_AUTHORITY option.

MQRCCF_ENCODING_ERROR
Encoding error.

The *Encoding* field in the message descriptor of the command does not match that required for the platform at which the command is being processed.

Corrective action: Construct the command with the correct encoding, and specify this in the message descriptor when sending the command.

MQRCCF_INCORRECT_Q
Command sent to wrong broker queue.

The command is a valid broker command but the queue it has been sent to is incorrect. **Publish** and **Delete Publication** commands need to be sent to the stream queue, all other commands need to be sent to the SYSTEM.BROKER.CONTROL.QUEUE.

Corrective action: Retry the command by sending it to the correct queue.

MQRCCF_INCORRECT_STREAM
Stream name does not match the stream queue it was sent to.

A command has been sent to a stream queue that specified a different stream name parameter.

Corrective action: Retry the command either by sending it to the correct stream queue or by modifying the command so that the stream name parameter matches.

MQRCCF_MD_FORMAT_ERROR
Format not valid.

The MQMD *Format* field value was not MQFMT_ADMIN.

Corrective action: Specify the valid format.

MQRCCF_MSG_LENGTH_ERROR
Message length not valid.

A message length error was detected. The message data length was inconsistent with the length implied by the parameters in the message, or a positional parameter was out of sequence.

Corrective action: Specify a valid message length, and check that positional parameters are in the correct sequence.

MQRCCF_NO_RETAINED_MSG
No retained message exists for the topic specified.

A **Request Update** command has been issued to request the retained message associated with the specified topic. No retained message exists for that topic.

Corrective action: If the topic or topics in question should have retained messages the publishers of these topics might not be publishing with the correct publication options to cause their publications to be retained.

MQRCCF_NOT_AUTHORIZED

Subscriber has insufficient authority.

To receive publications a subscriber application needs both browse authority for the stream queue that it is subscribing to, and put authority for the queue that publications are to be sent to. Subscriptions are rejected if the subscriber does not have both authorities. In addition to having browse authority for the stream queue, a subscriber would also require **altusr** authority for the stream queue in order to subscribe to certain topics that the broker itself publishes information on. These topics start with the MQ/SA/ prefix.

Corrective action: Ensure that the subscriber has the necessary authorities and re-issue the request. The problem might occur because the subscriber's user ID is not known to the broker. This can be identified if a further error reason code of MQRC_UNKNOWN_ENTITY is returned within the error response message.

MQRCCF_NOT_REGISTERED

Subscriber or publisher is not registered.

A **Deregister** command has been issued to remove registrations for a topic, or topics, for which the publisher or subscriber is not registered. If multiple topics were specified on the command it will fail with a completion code of MQCC_WARNING if the publisher or subscriber was registered for some, but not all, of the topics specified. This error code is also returned to a subscriber issuing a **Request Update** command for a topic for which he does not have a subscription.

Corrective action: Investigate why the publisher or subscriber is not registered. In the case of a subscriber, the subscriptions might have expired, or been removed automatically by the broker if the subscriber is no longer authorized.

MQRCCF_PUB_OPTIONS_ERROR

Invalid publication options have been supplied.

The publication options provided on a Publish command are not valid.

Corrective action: Retry the command with a valid combination of options.

MQRCCF_Q_MGR_NAME_ERROR

An invalid or unknown queue manager name has been supplied.

A queue manager name has been supplied as part of a publisher or subscriber identity. This might have been supplied as an explicit parameter or in the *ReplyToQMgr* field in the message descriptor of the command. Either the queue manager name is not valid, or in the case of a subscriber identity, the subscriber's queue could not be resolved because the remote queue manager is not known to the broker queue manager.

Corrective action: Retry the command with a valid queue manager name. If appropriate the broker will include a further error reason code within the error response message. If one is supplied then follow the guidance for that reason code in the *MQSeries Application Programming Reference* manual to resolve the problem.

MQRCCF_Q_NAME_ERROR

An invalid or unknown queue name has been supplied.

A queue name has been supplied as part of a publisher or subscriber identity. This might have been supplied as an explicit parameter or in the *ReplyToQ* field in the message descriptor of the command. Either the queue name is not valid, or in the case of a subscriber identity, the broker has failed to open the queue.

Corrective action: Retry the command with a valid queue name. If appropriate the broker will include a further error reason code within the error response message. If one is supplied then follow the guidance for that reason code in the *MQSeries Application Programming Reference* manual to resolve the problem.

MQRCCF_REG_OPTIONS_ERROR

Invalid registration options have been supplied.

The registration options provided on a command are not valid.

Corrective action: Retry the command with a valid combination of options.

MQRCCF_STREAM_ERROR

Stream name is not valid.

The stream name parameter is not valid. Stream names must obey the same naming rules as for MQSeries queues.

Corrective action: Retry the command with a valid stream name parameter.

MQRCCF_TOPIC_ERROR

Topic name is invalid.

A command has been sent to the broker containing a topic name that is not valid. Note that wildcard topic names are not allowed for **Register Publisher** and **Publish** commands.

Corrective action: Retry the command with a valid topic name parameter. Up to 256 characters of the topic name in question are returned with the error response message. If the topic name contains a null character this is assumed to terminate the string and is not considered to be part of it. A zero length topic name is not valid, as is one which contains an escape sequence that is not valid.

MQRCCF_UNKNOWN_BROKER

Command received from an unknown broker.

Within a multi-broker network, related brokers pass subscriptions and publications between each other as a series of command messages. One such command message has been received from a broker that is not, or is no longer, related to the detecting broker.

Corrective action: This situation can occur if the broker network is not quiesced while topology changes are made to the network. When removing a broker from the network ensure that the channels between the two related brokers in question are active.

MQRCCF_UNKNOWN_STREAM

Stream is not known by the broker or could not be created.

A command message has been put to the SYSTEM.BROKER.CONTROL.QUEUE for an unknown stream. This error code is also returned if dynamic stream creation is enabled and the broker failed to create a stream queue for the new stream using the SYSTEM.BROKER.MODEL.STREAM queue.

Corrective action: Retry the command for a stream that the broker supports. If the broker should support the stream, either define the stream queue manually, or correct the problem which prevented the broker from creating the stream queue itself.

MQRC_RFH_COMMAND_ERROR

Command not valid.

The message contains an MQRFH structure, but the command name contained in the *NameValueString* field is not valid.

Corrective action: Modify the application that generated the message to ensure that it places in the *NameValueString* field a command name that is valid.

MQRC_RFH_DUPLICATE_PARM

Duplicate parameter.

The message contains an MQRFH structure, but a parameter occurs more than once in the *NameValueString* field when only one occurrence is valid for the specified command.

Corrective action: Modify the application that generated the message to ensure that it places in the *NameValueString* field only one occurrence of the parameter.

MQRC_RFH_ERROR

MQRFH structure not valid.

The message contains an MQRFH structure, but the structure is not valid.

Corrective action: Modify the application that generated the message to ensure that it places a valid MQRFH structure in the message data. You may find it helpful to stop the broker and rerun the failing application. The `amqsbcg` sample program can then be used to examine the failing input message on the broker's queue.

MQRC_RFH_PARM_ERROR

Parameter not valid.

The message contains an MQRFH structure, but a parameter name contained in the *NameValueString* field is not valid for the command specified.

Corrective action: Modify the application that generated the message to ensure that it places in the *NameValueString* field only parameters that are valid for the specified command.

MQRC_RFH_PARM_MISSING

Parameter missing.

The message contains an MQRFH structure, but the command specified in the *NameValueString* field requires a parameter that is not present.

Corrective action: Modify the application that generated the message to ensure that it places in the *NameValueString* field all parameters that are required for the specified command.

MQRC_RFH_STRING_ERROR

"NameValueString" field not valid.

The contents of the *NameValueString* field in the MQRFH structure are not valid. *NameValueString* must adhere to the following rules:

- The string must consist of zero or more name/value pairs separated from each other by one or more blanks; the blanks are not significant.
- If a name or value contains blanks that are significant, the name or value must be enclosed in double-quote characters.
- If a name or value itself contains one or more double-quote characters, the name or value must be enclosed in double-quote characters, and each embedded double-quote character must be doubled.
- A name or value can contain any characters other than the null, which acts as a delimiter. The null and characters following it, up to the defined length of *NameValueString*, are ignored.

The following is a valid *NameValueString*:

```
Famous_Words "The program displayed ""Hello World"""
```

Corrective action: Modify the application that generated the message to ensure that it places in the *NameValueString* field data that adheres to the rules listed above. Check that the *StrucLength* field is set to the correct value.

Appendix B. Error messages

AMQ5805 MQSeries message broker currently running for queue manager.

Explanation: The command was unsuccessful because queue manager &3 currently has an MQSeries message broker running.

User action: None.

AMQ5806 MQSeries message broker started for queue manager &3.

Explanation: MQSeries message broker started for queue manager &3.

User action: None.

AMQ5807 MQSeries message broker for queue manager &3 ended.

Explanation: The MQSeries message broker on queue manager &3 has ended.

User action: None.

AMQ5808 MQSeries message broker for queue manager &3 is already quiescing.

Explanation: The endmqbrk command was unsuccessful because an orderly shutdown of the MQSeries message broker running on queue manager &3 is already in progress.

User action: None.

AMQ5809 MQSeries message broker for queue manager &3 starting.

Explanation: The dspmqbrk command has been issued to query the state of the MQSeries message broker. The MQSeries broker is currently initializing.

User action: None.

AMQ5810 MQSeries message broker for queue manager &3 running.

Explanation: The dspmqbrk command has been issued to query the state of the MQSeries message broker. The MQSeries broker is currently running.

User action: None.

AMQ5811 MQSeries message broker for queue manager &3 quiescing.

Explanation: The dspmqbrk command has been issued to query the state of the MQSeries message broker. The MQSeries broker is currently performing a controlled shutdown.

User action: None.

AMQ5812 MQSeries message broker for queue manager &3 stopping.

Explanation: Either the dspmqbrk command or the endmqbrk command has been issued. The MQSeries broker is currently performing an immediate shutdown. If the endmqbrk command has been issued to request that the broker terminate, the command is unsuccessful because the broker is already performing an immediate shutdown.

User action: None.

AMQ5813 MQSeries message broker for queue manager &3 not active.

Explanation: An MQSeries message broker administration command has been issued to query or change the state of the message broker. The MQSeries broker is not currently running.

User action: None.

AMQ5814 MQSeries message broker for queue manager &3 ended abnormally.

Explanation: The dspmqbrk command has been issued to query the state of the MQSeries message broker. The MQSeries broker has ended abnormally.

User action: Refer to the queue manager error logs to determine why the broker ended abnormally.

AMQ5815 Invalid MQSeries message broker initialization file stanza for queue manager &3.

Explanation: The MQSeries message broker was started using the strmqbrk command. The Broker stanza in the queue manager initialization file is not valid. The broker will terminate immediately. The attribute which is not valid is &5.

User action: Correct the Broker stanza in the queue manager initialization file.

© Copyright IBM Corp. 1998, 2000

AMQ5816 Unable to open MQSeries message broker control queue for reason &1,&2.

Explanation: The MQSeries message broker has failed to open the message broker control queue (&3). The attempt to open the queue failed with completion code &1 and reason &2.

The most likely reasons for this error are that an application program has opened the message broker control queue for exclusive access, or that the message broker control queue has been defined incorrectly. The broker will terminate immediately.

User action: Correct the problem and restart the broker.

AMQ5817 An invalid stream queue (&3) has been detected by the MQSeries message broker.

Explanation: MQSeries has detected an attempt to use a queue (&3) as a stream queue, but the attributes of the queue make it unsuitable for use as a stream queue. The most likely reason for this error is that the queue is:

1. Not a local queue
2. A shareable queue
3. A temporary dynamic queue.

If the queue was created using implicit stream creation, the model stream might have been defined incorrectly. The message that caused the stream to be created will be rejected or put to the dead-letter queue, depending upon the message report options and broker configuration.

User action: Correct the problem and resubmit the request.

AMQ5818 Unable to open MQSeries message broker stream queue (&3) for reason &1,&2.

Explanation: The MQSeries message broker has failed to open a stream queue (&3) for the reason indicated. The attempt to open the queue failed with completion code &1 and reason &2.

The most likely reason for this error is that an application has the queue open for exclusive access. The stream will be temporarily shut down and an attempt will be made to restart the stream after a short interval.

User action: Correct the problem.

AMQ5819 MQSeries message broker stream &3 has ended abnormally for reason &1.

Explanation: An MQSeries message broker stream has ended abnormally. The broker will attempt to restart the stream. If the stream should repeatedly fail then the broker will progressively increase the time between attempts to restart the stream.

User action: Investigate why the problem occurred and take appropriate action to correct the problem. If the problem persists, contact your IBM service representative.

AMQ5820 MQSeries message broker stream &3 restarted.

Explanation: The MQSeries message broker has restarted a stream that ended abnormally. This message will frequently be preceded by message AMQ5867 or AMQ5819 indicating why the stream ended.

User action: Correct the problem.

AMQ5821 MQSeries message broker unable to contact parent broker (&3) for reason &1.

Explanation: The MQSeries message broker has been started specifying a parent broker. The message broker has been unable to send a message to the parent broker. The message broker will terminate immediately.

User action: Investigate why the problem occurred and take appropriate action to correct the problem. The problem is likely to be caused by the parent broker name not resolving to the name of a transmission queue on the local broker.

AMQ5822 MQSeries message broker failed to register as child of broker(&3) for reason &1.

Explanation: The MQSeries message broker has been started specifying a parent broker. The message broker attempted to register as a child of the parent but received an exception response indicating that this was not possible. The message broker will attempt to reregister as a child of the parent periodically. The child might not be able to process global publications or subscriptions correctly until this registration process has completed normally.

User action: Investigate why the problem occurred and take appropriate action to correct the problem. The problem is likely to be caused by the parent broker not yet existing, or a problem with the SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS queue at the parent broker.

AMQ5823 Exit path attribute invalid in MQSeries message broker stanza.

Explanation: The MQSeries message broker exit path attribute (&3) is not valid. The attribute should be specified as: <path><module name>(<function name>). The MQSeries message broker will terminate immediately.

User action: Correct the problem with the attribute and restart the broker.

AMQ5824 MQSeries message broker exit module could not be loaded.

Explanation: MQSeries message broker exit module '&3' could not be loaded for reason '&1:&4'. The broker will terminate immediately.

User action: Correct the problem with MQSeries message broker exit module '&3' and restart the broker.

AMQ5825 The address of the MQSeries message broker exit function could not be found.

Explanation: The address of the MQSeries message broker exit function &4 could not be found in module &3 for reason &1:&5. The broker will terminate immediately.

User action: Correct the problem with the MQSeries message broker exit function &4 in module &3, and restart the broker.

AMQ5826 MQSeries message broker failed to propagate subscription to stream &4 at broker &3. Reason codes &1 and &2.

Explanation: An application has either registered or deregistered a global subscription to stream &4. The broker has attempted to propagate the subscription change to broker &3 but the request has not been successful. The message broker will immediately attempt to refresh the state of the global subscriptions for stream &4 at broker &3.

Until the subscription state has been successfully refreshed, messages published on stream &4 through broker &3 might not reach this broker.

User action: Use the reason codes to investigate why the problem occurred and take appropriate action to correct the problem.

AMQ5827 MQSeries message broker failed to subscribe to stream &4 at broker &3. Reason codes &1 and &2.

Explanation: Related MQSeries message brokers learn about each other's configuration by subscribing to information published by each other. An MQSeries message broker has discovered that one of these internal subscriptions has failed. The broker will reissue the subscription immediately.

The broker cannot function correctly without knowing some information about neighboring brokers. The information that this broker has about broker &3 is not complete and this could lead to subscriptions and publications not being propagated around the network correctly.

User action: Investigate why the problem occurred and take appropriate action to correct the problem. The most likely cause of this failure is a problem with the SYSTEM.BROKER.CONTROL.QUEUE at broker &3, or a problem with the definition of the route between this broker and broker &3.

AMQ5828 MQSeries message broker exit returned an ExitResponse that is not valid.

Explanation: The MQSeries message broker exit returned an ExitResponse &1 that is not valid. The message has been allowed to continue and an FFST has been generated that contains the entire exit parameter structure.

User action: Correct the problem with the MQSeries message broker exit.

AMQ5829 Usage: strmqbrk [-m QMgrName] [-p ParentQMgrName]

Explanation: This shows the correct usage.

User action: None.

AMQ5830 Usage: endmqbrk [-c | -i] [-m QMgrName]

Explanation: This shows the correct usage.

User action: None.

AMQ5831 Usage: dspmqbrk [-m QMgrName]

Explanation: This shows the correct usage.

User action: None.

AMQ5832 MQSeries message broker failed to publish configuration information on SYSTEM.BROKER.ADMIN.STREAM.

Explanation: Related MQSeries message brokers learn about each other's configuration by subscribing to information published by each other. An MQSeries message broker has discovered that one of these internal publications has failed. The broker will republish the information immediately.

Brokers cannot function correctly without knowing some information about neighboring brokers. The information that neighboring brokers have of this broker might not be complete and this could lead to some subscriptions and publications not being propagated around the network.

User action: Investigate why the problem occurred and take appropriate action to correct the problem.

AMQ5833 A loop has been detected in the MQSeries message broker hierarchy.

Explanation: The MQSeries message broker, on queue manager &3, introduced a loop in the broker hierarchy. This broker will terminate immediately.

User action: Remove broker &3 from the hierarchy, either by deleting the broker, or by removing knowledge of the broker's parent, using the clrmqbrk command.

AMQ5834 Conflicting queue manager names in the MQSeries message broker hierarchy.

Explanation: The names of the queue managers &3 and &4 in the MQSeries message broker hierarchy both start with the same 12 characters. The first 12 characters of a broker's queue manager name should be unique to ensure that no confusion arises within the broker hierarchy, and to guarantee unique message ID allocation.

User action: Use a queue manager naming convention that guarantees uniqueness of the first 12 characters of the queue manager name.

AMQ5835 MQSeries message broker failed to inform its parent of a relation for reason &1.

Explanation: The message broker failed to notify its parent on queue manager '&3' of the relation '&4' in the broker hierarchy. The notification message will be put to the parent's dead-letter queue. A failure to notify a broker of a new relation will mean that no loop detection can be performed for the new relation.

User action: Diagnose and correct the problem on the parent queue manager. One possible reason for this is that the parent broker does not yet exist.

AMQ5836 Duplicate queue manager name located in the MQSeries message broker hierarchy.

Explanation: Multiple instances of the queue manager name '&3' have been located. This could either be the result of a previously resolved loop in the broker hierarchy, or multiple queue managers in the broker hierarchy having the same name.

User action: If this broker introduced a loop in the hierarchy (typically identified by message AMQ5833), this message can be ignored. It is strongly recommended that every queue manager in a broker hierarchy has a unique name. It is not recommended that multiple queue managers use the same name.

AMQ5837 MQSeries message broker failed to quiesce queue &3 for reason &1.

Explanation: When a broker is deleted, the broker's input queues are quiesced by making the queue get inhibited, and writing the contents of the queue to the dead-letter queue (depending upon the report options of the message). The broker was unable to quiesce the named queue for the reason shown. The attempt to delete the broker will fail.

User action: Investigate why the problem occurred, take appropriate action to correct the problem, and reissue the dltmqbrk command. Likely reasons include the queue being open for input by another process, there being no dead-letter queue defined at this queue manager, or the operator setting the queue to get inhibited while the dltmqbrk command is running.

If there is no dead-letter queue defined, the reason will be reported as MQRC_UNKNOWN_OBJECT_NAME. If the problem occurs because there is no dead-letter queue defined at this broker, the operator can either define a dead-letter queue, or manually empty the queue causing the problem.

AMQ5838 MQSeries message broker cannot be deleted as child &3 is still registered.

Explanation: An MQSeries message broker cannot be deleted until all other brokers that have registered as children of that broker have deregistered as its children.

User action: Use the clrmqbrk and dltmqbrk commands to change the broker topology so that broker &3 is not registered as a child of the broker being deleted.

AMQ5839 MQSeries message broker received unexpected inter-broker communication from broker &3.

Explanation: An MQSeries message broker has received an inter-broker communication that it did not expect. The message was sent by broker &3. The message will be processed according to the report options in that message.

The most likely reason for this message is that the broker topology has been changed while inter-broker communication messages were in transit (for example, on a transmission queue) and that a message relating to the previous broker topology has arrived at a broker in the new topology. This message may be accompanied by an informational FFST including details of the unexpected communication.

User action: If the broker topology has changed and the broker named in the message is no longer related to the broker issuing this message, this message can be ignored.

If the clrmqbrk command was issued to unilaterally remove knowledge of broker &3 from this broker, the clrmqbrk command should also be used to remove knowledge of this broker from broker &3.

If the clrmqbrk command was issued to unilaterally remove knowledge of this broker from broker &3, the clrmqbrk command should also be used to remove knowledge of broker &3 at this broker.

AMQ5840 MQSeries message broker unable to delete queue &3 for reason &2.

Explanation: The MQSeries message broker has failed to delete the named queue for the reason shown. The broker typically attempts to delete queues during dltmqbrk processing, in which case the dltmqbrk command will fail.

User action: The most likely reason for this error is that some other process has the queue open. Determine why the queue cannot be deleted, remove the inhibitor, and retry the failed operation. In a multi-broker environment, it is likely that a message channel agent might have queues open, which the broker needs to delete for a dltmqbrk command to complete.

AMQ5841 MQSeries message broker &3 deleted.

Explanation: The MQSeries message broker &3 has been deleted using the dltmqbrk command.

User action: None.

AMQ5842 MQSeries message broker &3 cannot be deleted for reason &1&5.

Explanation: An attempt has been made to delete the MQSeries message broker &3 but the request has failed for reason &1.

User action: Determine why the dltmqbrk command cannot complete successfully. The message logs for the queue manager might contain more detailed information on why the broker cannot be deleted. Resolve the problem that is preventing the command from completing and reissue the dltmqbrk command.

AMQ5843 MQSeries message broker &3 cannot be started as it is partially deleted.

Explanation: An attempt has been made to start an MQSeries message broker that is in a partially deleted state. An earlier attempt to delete the broker has failed. The broker deletion must be completed before the broker will be allowed to restart.

When broker deletion is successful, message AMQ5841 is issued, indicating that the broker has been deleted. If this message is not received on completion of a dltmqbrk command, the broker deletion has not been completed and the command will have to be reissued.

User action: Investigate why the earlier attempt to delete the broker failed. Resolve the problem and reissue the dltmqbrk command.

AMQ5844 MQSeries message broker relation &4 is unknown to broker &3.

Explanation: The clrmqbrk command has been issued in an attempt to remove a broker's knowledge of a relation of that broker. The relative &4 is unknown at broker &3. If the "-p" flag was specified, the broker does not currently have a parent. If the "-c" flag was specified, the broker does not recognize the named child.

User action: Investigate why the broker is unknown.

AMQ5845 Usage: dltmqbrk -m QMgrName

Explanation: This shows the correct usage.

User action: None.

AMQ5846 Usage: clrmqbrk -p | -c ChildQMgrName -m QMgrName

Explanation: This shows the correct usage.

User action: None.

AMQ5847 MQSeries message broker &3 has removed knowledge of relation &4.

Explanation: The clrmqbrk command has been used to remove knowledge of broker &4 from broker &3.

User action: None.

AMQ5848 MQSeries message broker &3 has failed to remove references to relation &4 for reason &1&5.

Explanation: An attempt has been made to remove references to broker &4 from broker &3 using the clrmqbrk command, but the request has been unsuccessful.

User action: Determine why the clrmqbrk command cannot complete successfully. The message logs for the queue manager might contain more detailed information on why the broker cannot be deleted. Resolve the problem that is preventing the command from completing and then reissue the clrmqbrk command.

AMQ5849 MQSeries message broker &3 may not change parent from &5 to &4.

Explanation: An attempt has been made to start broker &3, nominating broker &4 as its parent. Message broker &3 has previously been started, nominating broker &5 as its parent. The strmqbrk command cannot be used to change an existing relationship.

User action: Do not attempt to change the broker topology by using the strmqbrk command. The dltmqbrk and clrmqbrk commands are the only supported means of changing the broker topology. Refer to the documentation of those commands for guidance on changing the broker topology.

AMQ5850 MQSeries message broker interrupted while creating queue &3 for user ID &4.

Explanation: When the MQSeries message broker creates a queue, it first creates the queue with default security attributes and it then sets the appropriate security attributes for the queue. If the broker should be interrupted during this operation (for example the queue manager is shut down), the broker cannot reliably detect that the security attributes have not been set correctly.

The MQSeries message broker was creating a queue, but was interrupted before it could complete creation of the queue and setting the initial authority. If the interrupt occurred before the initial authority of the queue could be set, it might be necessary for the operator to set the appropriate authorities using the setmqaut command.

User action: Confirm that the named queue has the appropriate security attributes and modify them as necessary.

AMQ5851 MQSeries message broker interrupted while creating internal queue &3 for user ID &4.

Explanation: When the MQSeries message broker creates an internal queue, it first creates the queue with default security attributes and it then sets the appropriate security attributes for the queue. If the broker should be interrupted during this operation (for example the queue manager is shut down), the broker attempts to delete and redefine the queue.

If the internal queue is available to users (for example, the default stream or the administration stream), it is possible that a user will put a message on the queue while it is in this invalid state, or that a user application has the queue open. In this situation the broker does not automatically redefine the queue and cannot be restarted until the queue has been emptied or closed.

User action: Examine any messages on the named queue and take appropriate action to remove them from the queue. Ensure that no applications have the queue open.

AMQ5852 MQSeries message broker failed to propagate delete publication command for stream &3 to related broker &4 for reason &1.

Explanation: When an application issues a delete publication command to delete a global publication, the command has to be propagated to all brokers in the sub-hierarchy supporting the stream.

The broker reporting the error has failed to forward a delete publication command to a related broker (&4) who supports stream (&3).

Delete publication commands are propagated without MQRO_DISCARD and the command message might have been written to a dead-letter queue. The topic for which the delete publication has failed is &5.

User action: If the delete publication has failed because the stream has been deleted at the related broker, this message can be ignored. Investigate why the delete publication has failed and take the appropriate action to recover the failed command.

AMQ5853 MQSeries message broker failed to propagate a delete publication command for stream &3 to a previously related broker.

Explanation: When an application issues a delete publication command to delete a global publication, the command is propagated to all brokers in the sub-hierarchy supporting the stream.

The broker topology was changed after deleting the publication, but before a broker removed by the topology change processed the propagated delete publication message. The topic for which the delete publication has failed is &5.

User action: It is the user's responsibility to quiesce broker activity before changing the broker topology using the clrmqbrk command. Investigate why this delete publication activity was not quiesced. The delete publication command will have been written to the dead-letter queue at the broker that was removed from the topology. In this case, further action might be necessary to propagate the delete publication command that was not quiesced before the clrmqbrk command was issued.

If this message occurs as a result of the dltmqbrk command, the publication will have been deleted as a result of the dltmqbrk command, and the delete publication message will have been written to the dead-letter queue at the queue manager where the broker was deleted. In this case the delete publication message on the dead-letter queue can be discarded.

AMQ5854 MQSeries message broker failed to propagate a delete publication command for stream &3 to broker &4.

Explanation: When an application issues a delete publication command to delete a global publication, the command has to be propagated to all brokers in the sub-hierarchy supporting the stream.

At the time the delete publication was propagated, broker &4 was a known relation of this message broker supporting stream &3. Before the delete publication command arrived at the related broker, the broker topology was changed so that broker &4 no longer supported stream &3. The topic for which the delete publication has failed is &5.

User action: It is the user's responsibility to quiesce broker activity before changing the stream topology of the broker. Investigate why this delete publication activity was not quiesced. The delete publication command will have been written to the dead-letter queue at broker &4.

AMQ5855 MQSeries message broker &3 ended for reason &1&5.

Explanation: An attempt has been made to run the MQSeries message broker &3 but the broker has ended for reason &1.

User action: Determine why the broker ended. The message logs for the queue manager might contain more detailed information on why the broker cannot be started. Resolve the problem that is preventing the command from completing and reissue the strmqbrk command.

AMQ5856 Broker publish command message cannot be processed. Reason code &1.

Explanation: The MQSeries broker failed to process a publish message for stream &3. The broker was unable to write the publication to the dead-letter queue and was not permitted to discard the publication.

The broker will temporarily stop the stream and will restart the stream and consequently retry the publication after a short interval.

User action: Investigate why the error has occurred and why the publication cannot be written to the dead-letter queue. Either manually remove the publication from the stream queue, or correct the problem that is preventing the broker from writing the publication to the dead-letter queue.

AMQ5857 Broker control command message cannot be processed. Reason code &1.

Explanation: The MQSeries broker failed to process a command message on the SYSTEM.BROKER.CONTROL.QUEUE. The broker was unable to write the command message to the dead-letter queue and was not permitted to discard the command message.

The broker will temporarily stop the stream and will restart the stream and consequently retry the command message after a short interval. Other broker control commands cannot be processed until this command message has been processed successfully or removed from the control queue.

User action: Investigate why the error has occurred and why the command message cannot be written to the dead-letter queue. Either manually remove the command message from the stream queue, or correct the problem that is preventing the broker from writing the command message to the dead-letter queue.

AMQ5858 Broker could not send publication to subscriber queue &4 at queue manager &3 for reason &1.

Explanation: A failure has occurred sending a publication to subscriber queue &4 at queue manager &3. The broker configuration options prevent it from recovering from this failure by discarding the publication or by sending it to the dead-letter queue. Instead the broker will back out the unit of work under which the publication is being sent and retry the failing command message a fixed number of times.

If the problem still persists, the broker will then attempt to recover by failing the command message with a negative reply message. If the issuer of the command did not request negative replies, the broker will either discard or send to the dead-letter queue the failing command message. If the broker configuration options prevent this, the broker will restart the affected stream, which will reprocess the failing command message again.

This behavior will be repeated until such time as the failure is resolved. During this time the stream will be unable to process further publications or subscriptions.

User action: Usually the failure will be due to a transient resource problem, for example, the subscriber queue, or an intermediate transmission queue, becoming full. Use reason code &1 to determine what remedial action is required.

If the problem persists for a long time, you will notice the stream being continually restarted by the broker. Evidence of this occurring will be a large number of AMQ5820 messages, indicating stream restart, being written to the error logs. In such circumstances, manual intervention will be required to allow the broker to dispose of the failing publication. To do this, you will need to end the broker using the endmqbrk command and restart it with appropriate disposition options. This will allow the publication to be sent to the rest of the subscribers, while allowing the broker to discard or send to the dead-letter queue the publication that could not be sent.

AMQ5859 MQSeries message broker stream &3 is terminating due to an internal resource problem. Reason code &1.

Explanation: The MQSeries stream &3 has run out of internal resources and will terminate. If the command in progress was being processed under syncpoint control, it will be backed out and retried when the stream is restarted by the broker. If the command was being processed out of syncpoint control, it will not be able to be retried when the stream is restarted.

User action: This message should only be issued in very unusual circumstances. If this message is issued repeatedly for the same stream, and the stream is not especially large in terms of subscriptions, topics, and retained publications, save all generated diagnostic information and contact your IBM Support Center for problem resolution.

AMQ5864 Broker reply message could not be sent to queue &4 at queue manager &3 for reason &1. The command will be retried.

Explanation: While processing a publish/subscribe command, the MQSeries message broker could not send a reply message to queue &4 at queue manager &3. The broker was also unable to write the message to the dead-letter queue. Since the command is being processed under syncpoint control, the broker will attempt to retry the command in the hope that the problem is only of a transient nature.

If, after a set number of retries, the reply message still could not be sent, the command message will be discarded if the report options allow it. If the command message is not discardable, the stream will be restarted, and processing of the command message recommenced.

User action: Use reason code &1 to determine what remedial action is required. If the failure is due to a resource problem (for example, a queue being full), you might find that the problem has already cleared itself. If not, this message will be issued repeatedly each time the command is retried. In this case you are strongly advised to define a dead-letter queue to receive the reply message so that the MQSeries broker can process other commands while the problem is being investigated. Check the application from which the command originated and ensure that it is specifying its reply-to queue correctly.

AMQ5865 Broker reply message could not be sent to queue &4 at queue manager &3 for reason &1.

Explanation: While processing a publish/subscribe command, the MQSeries message broker could not send a reply message to queue &4 at queue manager &3. The broker was also unable to write the message to the dead-letter queue. As the command is not being processed under syncpoint control, the broker is not able to retry the command.

User action: Use reason code &1 to determine what remedial action is required. If the failure is due to a resource problem (for example, a queue being full), you might find that the problem has already cleared itself. If not, check the application from which the command originated and ensure that it is specifying its reply-to queue correctly. You might find that defining a dead-letter queue to capture the reply message on a subsequent failure will help you with this task.

AMQ5866 Broker command message has been discarded. Reason code &1.

Explanation: The MQSeries broker failed to process a publish/subscribe command message, which has now been discarded. The broker will begin to process new command messages again.

User action: Look for previous error messages to indicate the problem with the command message. Correct the problem to prevent the failure from happening again.

AMQ5867 MQSeries message broker stream &3 has ended abnormally for reason &1.

Explanation: An MQSeries message broker stream has ended abnormally. The broker will attempt to restart the stream. If the stream should repeatedly fail, the broker will progressively increase the time between attempts to restart the stream.

User action: Use the reason code &1 to investigate why the problem occurred.

A reason code of 1 indicates that the stream ended because a command message could not be processed successfully. Look in the error logs for earlier messages to determine the reason why the command message failed.

A reason code of 2 indicates that the stream ended because the broker exit could not be loaded. Until the problem with the broker exit has been resolved, the stream will continue to fail.

AMQ5868 User ID &3 is no longer authorized to subscribe to stream &4.

Explanation: The broker has attempted to publish a publication to a subscriber, but the subscriber no longer has browse authority to stream queue &4. The publication is not sent to the subscriber and its subscription is deregistered. An event publication containing details of the subscription that was removed is published on SYSTEM.BROKER.ADMIN.STREAM. While user ID &3 remains unauthorized, the broker will continue to deregister subscriptions associated with that user ID.

User action: If the authority of user ID &3 was intentionally removed, consider removing all of that user ID’s subscriptions immediately by issuing a ‘Deregister Subscriber’ command, specifying the ‘Deregister All’ option on the subscriber’s behalf. If the authority was revoked accidentally, reinstate it, but be aware that some, if not all, of the subscriber’s subscriptions will have been deregistered by the broker.

AMQ5869 MQSeries message broker is checkpointing registrations for stream &3.

Explanation: A large number of changes have been made to the publisher and subscriber registrations of stream &3. These changes are being checkpointed in order to minimize both stream restart time and the amount of internal queue space being used.

User action: None.

AMQ5875 MQSeries message broker cannot write a message to the dead-letter queue (&3) for reason &1:&4.

Explanation: The MQSeries message broker attempted to put a message to the dead-letter queue (&3) but the message could not be written to the dead-letter queue for reason &1:&4. The message was being written to the dead-letter queue with a reason of &2:&5.

User action: Determine why the message cannot be written to the dead-letter queue. Also, if the message was not deliberately written to the dead-letter queue, for example by a message broker exit, determine why the message was written to the dead-letter queue and resolve the problem that is preventing the message from being sent to its destination.

AMQ5876 A parent conflict has been detected in the MQSeries message broker hierarchy.

Explanation: MQSeries message broker &3 has been started, naming this broker as its parent. This broker was started naming broker &3 as its parent. The MQSeries message broker will send an exception message to broker &3 indicating that a conflict has been detected.

The most likely reason for this message is that the broker topology has been changed while inter-broker communication messages were in transit (for example, on a transmission queue) and that a message relating to the previous broker topology has arrived at a broker in the new topology. This message may be accompanied by an informational FFST including details of the unexpected communication.

User action: If the broker topology has changed and the broker named in the message no longer identifies this broker as its parent, this message can be ignored - for example, if the command "clrmqbrk -m &3 -p" was issued. If broker &3 has been defined as this broker’s parent, and this broker has been defined as broker &3's parent, the clrmqbrk or the dltmqbrk commands should be used to resolve the conflict.

AMQ5877 MQSeries message broker stream &3 has ended abnormally for reason &1.

Explanation: An MQSeries message broker stream has ended abnormally. The broker recovery routines failed to reset the stream state and the stream cannot be restarted automatically.

User action: Investigate why the stream failed and why the broker’s recovery routine could not recover following the failure. Take appropriate action to correct the problem.

Depending upon the broker configuration and the nature of the problem it will be necessary to restart either the MQSeries message broker, or both the queue manager and the MQSeries message broker, to make the stream available. If the problem persists contact your IBM service representative.

AMQ5878 MQSeries message broker recovery failure detected.

Explanation: An earlier problem has occurred with the MQSeries message broker, and either a stream has been restarted or the broker has been restarted. The restarted stream or broker has detected that the previous instance of the stream or broker did not clean up successfully and the restart will fail.

User action: Investigate the cause of the failure that caused a stream or broker restart to be necessary, and why the broker or stream was unable to clean up its resources following the failure.

When the broker processes with a non-trusted routing exit (RoutingExitConnectType=STANDARD), the broker runs in a mode where it is more tolerant of unexpected failures and it is likely that the restart will succeed after a short delay. In the case of a stream restart, the broker will normally periodically retry the failing restart. In the case of a broker restart, it will be necessary to manually retry the broker restart after a short delay.

When the broker processes without a routing exit, or with a trusted routing exit (RoutingExitConnectType=FASTPATH), the broker runs in a mode where it is less tolerant of unexpected failures and a queue manager restart will be necessary to resolve this problem. When the broker is running in this mode, it is important that the broker processes are not subjected to unnecessary asynchronous interrupts, for example, kill. If the problem persists, contact your IBM service representative.

AMQ5880 User ID &3 is no longer authorized to subscribe to stream &4.

Explanation: The broker has attempted to publish a publication to a subscriber but the subscriber no longer has altusr authority to stream queue &4. The publication is not sent to the subscriber and that user ID's subscription is deregistered. An event publication containing details of the subscription that was removed is published on SYSTEM.BROKER.ADMIN.STREAM.

While user ID &3 remains unauthorized, the broker will continue to deregister subscriptions associated with that user ID.

User action: If the authority of user ID &3 was intentionally removed, consider removing subscriptions immediately by issuing a 'Deregister Subscriber' command for the appropriate topics on the subscriber's behalf. If the authority was revoked accidentally, reinstate it, but be aware that some, if not all, of the subscriber's subscriptions will have been deregistered by the broker.

AMQ5881 Invalid MQSeries message broker configuration parameter combination &1.

Explanation: A combination of Broker stanzas in the queue manager initialization file is not valid. The broker will not operate until this has been corrected.

An invalid combination of (1) indicates that SyncPointIfPersistent has been set to TRUE and DiscardNonPersistentInputMsg has been set to FALSE. The latter must be set to TRUE when SyncPointIfPersistent is set to TRUE.

An invalid combination of (2) indicates that SyncPointIfPersistent has been set to TRUE and DiscardNonPersistentResponse has been set to FALSE. The latter must be set to TRUE when SyncPointIfPersistent is set to TRUE.

An invalid combination of (3) indicates that SyncPointIfPersistent has been set to TRUE and DiscardNonPersistentPublication has been set to FALSE. The latter must be set to TRUE when SyncPointIfPersistent is set to TRUE.

User action: Alter the message broker stanzas to comply with the above rules and retry the command.

AMQ5882 MQSeries message broker has written a message to the dead-letter queue.

Explanation: The MQSeries message broker has written a message to the dead-letter queue (&3) for reason &1:&4.

User action: If the message was not deliberately written to the dead-letter queue, for example by a message broker exit, determine why the message was written to the dead-letter queue, and resolve the problem that is preventing the message from being sent to its destination.

AMQ5883 MQSeries message broker state on stream &3 not recorded while processing a publication outside of syncpoint.

Explanation: A nonpersistent publication has requested a change to either a retained message or a publisher registration. This publication is being processed outside of syncpoint because the broker has been configured with the SyncPointIfPersistent option set. A failure has occurred hardening either the publisher registration or the retained publication to the broker's internal queue.

All state changes attempted as a result of this publication will be backed-out. Processing of the publication will continue and the broker will attempt to deliver it to all subscribers.

User action: Investigate why the failure occurred. It is probably due to a resource problem occurring on the broker. The most likely cause is 'queue full' on a broker queue. If your publications also carry state changes, you are advised to send them either as persistent publications or turn off the SyncPointIfPersistent option. In this way, they will be carried out under syncpoint and the broker can retry them in the event of a failure such as this.

AMQ5884 MQSeries message broker control queue is not a local queue.

Explanation: MQSeries has detected that the queue 'SYSTEM.BROKER.CONTROL.QUEUE' exists and is not a local queue. This makes the queue unsuitable for use as the control queue of the MQSeries message broker. The broker will terminate immediately.

User action: Delete the definition of the existing queue and, if required, re-create the queue to be of type MQQT_LOCAL. If you do not re-create the queue the broker will automatically create one of the correct type when started.

Appendix C. Constants

This appendix lists the values of the named constants used by the functions described in this manual. For information about MQSeries constants not in this list, see the *MQSeries Application Programming Reference* manual and the *MQSeries Programmable System Management* manual.

The constants are grouped according to the parameter or field to which they relate. Names of the constants in a group begin with a common prefix of the form MQxxxx_, where xxxx represents a string of 0 through 4 characters that indicates the nature of the values defined in that group.

All string constants, such as MQPS_COMMAND, have alternative constants defined. Those with suffix '_A' (for example MQPS_COMMAND_A) are in blank enclosed array form, and those with suffix '_B' are in blank enclosed string form. Character strings are shown delimited by double quotation marks; the quotation marks are not part of the value. Each character in an array is shown delimited by single quotation marks; the quotation marks are not part of the value.

For constants with numeric values, the values are shown in both decimal and hexadecimal forms. Hexadecimal values are represented using the notation `X'hhhhhhhh'`, where each `'h'` denotes a single hexadecimal digit.

String constants

MQPS_⋆ (Publish/Subscribe tag names)

MQPS_COMMAND	"MQPSCommand"
MQPS_COMP_CODE	"MQPSCompCode"
MQPS_DELETE_OPTIONS	"MQPSDelOpts"
MQPS_ERROR_ID	"MQPSErrorId"
MQPS_ERROR_POS	"MQPSErrorPos"
MQPS_INTEGER_DATA	"MQPSIntData"
MQPS_PARAMETER_ID	"MQPSParmId"
MQPS_PUBLICATION_OPTIONS	"MQPSPubOpts"
MQPS_PUBLISH_TIMESTAMP	"MQPSPubTime"
MQPS_Q_MGR_NAME	"MQPSQMgrName"
MQPS_Q_NAME	"MQPSQName"
MQPS_REASON	"MQPSReason"
MQPS_REASON_TEXT	"MQPSReasonText"
MQPS_REGISTRATION_OPTIONS	"MQPSRegOpts"
MQPS_SEQUENCE_NUMBER	"MQPSSeqNum"
MQPS_STREAM_NAME	"MQPSStreamName"
MQPS_STRING_DATA	"MQPSStringData"
MQPS_TOPIC	"MQPSTopic"
MQPS_USER_ID	"MQPSUserId"

© Copyright IBM Corp. 1998, 2000

The following blank-enclosed versions are also defined:

```
MQPS_COMMAND_B                  " MQPSCommand "
MQPS_COMP_CODE_B                " MQPSCompCode "
MQPS_DELETE_OPTIONS_B           " MQPSDelOpts "
MQPS_ERROR_ID_B                 " MQPSErrorId "
MQPS_ERROR_POS_B                " MQPSErrorPos "
MQPS_INTEGER_DATA_B             " MQPSIntData "
MQPS_PARAMETER_ID_B             " MQPSParmId "
MQPS_PUBLICATION_OPTIONS_B      " MQPSPubOpts "
MQPS_PUBLISH_TIMESTAMP_B        " MQPSPubTime "
MQPS_Q_MGR_NAME_B               " MQPSQMgrName "
MQPS_Q_NAME_B                   " MQPSQName "
MQPS_REASON_B                   " MQPSReason "
MQPS_REASON_TEXT_B              " MQPSReasonText "
MQPS_REGISTRATION_OPTIONS_B     " MQPSRegOpts "
MQPS_SEQUENCE_NUMBER_B          " MQPSSeqNum "
MQPS_STREAM_NAME_B              " MQPSStreamName "
MQPS_STRING_DATA_B              " MQPSStringData "
MQPS_TOPIC_B                    " MQPSTopic "
MQPS_USER_ID_B                  " MQPSUserId "
```

The following array versions are also defined:

```
MQPS_COMMAND_A                  ' ','M','Q','P','S',\
                                 'C','o','m','m','a','n','d',' '
MQPS_COMP_CODE_A                ' ','M','Q','P','S',\
                                 'C','o','m','p','C','o','d','e',' '
MQPS_DELETE_OPTIONS_A           ' ','M','Q','P','S',\
                                 'D','e','l','O','p','t','s',' '
MQPS_ERROR_ID_A                 ' ','M','Q','P','S',\
                                 'E','r','r','o','r','I','d',' '
MQPS_ERROR_POS_A                ' ','M','Q','P','S',\
                                 'E','r','r','o','r','P','o','s',' '
MQPS_INTEGER_DATA_A             ' ','M','Q','P','S',\
                                 'I','n','t','D','a','t','a',' '
MQPS_PARAMETER_ID_A             ' ','M','Q','P','S',\
                                 'P','a','r','m','I','d',' '
MQPS_PUBLICATION_OPTIONS_A      ' ','M','Q','P','S',\
                                 'P','u','b','O','p','t','s',' '
MQPS_PUBLISH_TIMESTAMP_A        ' ','M','Q','P','S',\
                                 'P','u','b','T','i','m','e',' '
MQPS_Q_MGR_NAME_A               ' ','M','Q','P','S',\
                                 'Q','M','g','r','N','a','m','e',' '
MQPS_Q_NAME_A                   ' ','M','Q','P','S',\
                                 'Q','N','a','m','e',' '
MQPS_REASON_A                   ' ','M','Q','P','S',\
                                 'R','e','a','s','o','n',' '
MQPS_REASON_TEXT_A              ' ','M','Q','P','S','R','e','a',\
                                 's','o','n','T','e','x','t',' '
MQPS_REGISTRATION_OPTIONS_A     ' ','M','Q','P','S',\
                                 'R','e','g','O','p','t','s',' '
MQPS_SEQUENCE_NUMBER_A          ' ','M','Q','P','S',\
                                 'S','e','q','N','u','m',' '
MQPS_STREAM_NAME_A              ' ','M','Q','P','S','S','t','r',\
                                 'e','a','m','N','a','m','e',' '
MQPS_STRING_DATA_A              ' ','M','Q','P','S','S','t','r',\
                                 'i','n','g','D','a','t','a',' '
```

MQPS_TOPIC_A	' ','M','Q','P','S',\ 'T','o','p','i','c',' '
MQPS_USER_ID_A	' ','M','Q','P','S',\ 'U','s','e','r','I','d',' '

MQPS_★ (Command tag values)

MQPS_DELETE_PUBLICATION	"DeletePub"
MQPS_DEREGISTER_PUBLISHER	"DeregPub"
MQPS_DEREGISTER_SUBSCRIBER	"DeregSub"
MQPS_PUBLISH	"Publish"
MQPS_REGISTER_PUBLISHER	"RegPub"
MQPS_REGISTER_SUBSCRIBER	"RegSub"
MQPS_REQUEST_UPDATE	"ReqUpdate"

The following blank-enclosed versions are also defined:

MQPS_DELETE_PUBLICATION_B	" DeletePub "
MQPS_DEREGISTER_PUBLISHER_B	" DeregPub "
MQPS_DEREGISTER_SUBSCRIBER_B	" DeregSub "
MQPS_PUBLISH_B	" Publish "
MQPS_REGISTER_PUBLISHER_B	" RegPub "
MQPS_REGISTER_SUBSCRIBER_B	" RegSub "
MQPS_REQUEST_UPDATE_B	" ReqUpdate "

The following array versions are also defined:

MQPS_DELETE_PUBLICATION_A	' ','D','e','l','e','t','e',\ 'P','u','b',' '
MQPS_DEREGISTER_PUBLISHER_A	' ','D','e','r','e','g',\ 'P','u','b',' '
MQPS_DEREGISTER_SUBSCRIBER_A	' ','D','e','r','e','g',\ 'S','u','b',' '
MQPS_PUBLISH_A	' ','P','u','b','l','i','s','h',' '
MQPS_REGISTER_PUBLISHER_A	' ','R','e','g','P','u','b',' '
MQPS_REGISTER_SUBSCRIBER_A	' ','R','e','g','S','u','b',' '
MQPS_REQUEST_UPDATE_A	' ','R','e','q','U','p','d','a','t','e',' '

MQPS_★ (Delete, publication and registration options)

MQPS_ANONYMOUS	"Anon"
MQPS_CORREL_ID_AS_IDENTITY	"CorrelAsId"
MQPS_DEREGISTER_ALL	"DeregAll"
MQPS_DIRECT_REQUESTS	"DirectReq"
MQPS_INCLUDE_STREAM_NAME	"InclStreamName"
MQPS_INFORM_IF_RETAINED	"InformIfRet"
MQPS_IS_RETAINED_PUBLICATION	"IsRetainedPub"
MQPS_LOCAL	"Local"
MQPS_NEW_PUBLICATIONS_ONLY	"NewPubsOnly"
MQPS_NO_REGISTRATION	"NoReg"
MQPS_NONE	"None"
MQPS_OTHER_SUBSCRIBERS_ONLY	"OtherSubsOnly"
MQPS_PUBLISH_ON_REQUEST_ONLY	"PubOnReqOnly"
MQPS_RETAIN_PUBLICATION	"RetainPub"

The following blank-enclosed versions are also defined:

```
MQPS_ANONYMOUS_B                " Anon "
MQPS_CORREL_ID_AS_IDENTITY_B    " CorrelAsId "
MQPS_DEREGISTER_ALL_B           " DeregAll "
MQPS_DIRECT_REQUESTS_B          " DirectReq "
MQPS_INCLUDE_STREAM_NAME_B      " InclStreamName "
MQPS_INFORM_IF_RETAINED_B       " InformIfRet "
MQPS_IS_RETAINED_PUBLICATION_B  " IsRetainedPub "
MQPS_LOCAL_B                    " Local "
MQPS_NEW_PUBLICATIONS_ONLY_B    " NewPubsOnly "
MQPS_NO_REGISTRATION_B          " NoReg "
MQPS_NONE_B                     " None "
MQPS_OTHER_SUBSCRIBERS_ONLY_B   " OtherSubsOnly "
MQPS_PUBLISH_ON_REQUEST_ONLY_B  " PubOnReqOnly "
MQPS_RETAIN_PUBLICATION_B       " RetainPub "
```

The following array versions are also defined:

```
MQPS_ANONYMOUS_A                ' ','A','n','o','n',' '
MQPS_CORREL_ID_AS_IDENTITY_A    ' ','C','o','r','r','e','l',\
                                 'A','s','I','d',' '
MQPS_DEREGISTER_ALL_A           ' ','D','e','r','e','g','A','l','l',' '
MQPS_DIRECT_REQUESTS_A          ' ','D','i','r','e','c','t',\
                                 'R','e','q',' '
MQPS_INCLUDE_STREAM_NAME_A      ' ','I','n','c','l','S','t','r',\
                                 'e','a','m','N','a','m','e',' '
MQPS_INFORM_IF_RETAINED_A       ' ','I','n','f','o','r','m',\
                                 'I','f','R','e','t',' '
MQPS_IS_RETAINED_PUBLICATION_A  ' ','I','s','R','e','t','a','i','n',\
                                 'e','d','P','u','b',' '
MQPS_LOCAL_A                    ' ','L','o','c','a','l',' '
MQPS_NEW_PUBLICATIONS_ONLY_A    ' ','N','e','w','P','u','b','s',\
                                 'O','n','l','y',' '
MQPS_NO_REGISTRATION_A          ' ','N','o','R','e','g',' '
MQPS_NONE_A                     ' ','N','o','n','e',' '
MQPS_OTHER_SUBSCRIBERS_ONLY_A   ' ','O','t','h','e','r','S','u','b','s',\
                                 'O','n','l','y',' '
MQPS_PUBLISH_ON_REQUEST_ONLY_A  ' ','P','u','b','O','n','R','e','q',\
                                 'O','n','l','y',' '
MQPS_RETAIN_PUBLICATION_A       ' ','R','e','t','a','i','n',\
                                 'P','u','b',' '
```

MQRFH_★ (Rules and formatting header structure identifier)

```
MQRFH_STRUC_ID                  "RFH "
```

The following array version is also defined:

```
MQRFH_STRUC_ID_ARRAY            'R','F','H',' '
```

Integer constants

MQAT_⋆ (Application type for message descriptor)

MQAT_BROKER	26	X'0000001A'

MQCACF_⋆ (Character parameter identifiers for PCF)

MQCACF_STREAM_NAME	3030	X'00000BD6'
MQCACF_TOPIC	3031	X'00000BD7'
MQCACF_PARENT_Q_MGR_NAME	3032	X'00000BD8'
MQCACF_PUBLISH_TIMESTAMP	3034	X'00000BDA'
MQCACF_STRING_DATA	3035	X'00000BDB'
MQCACF_SUPPORTED_STREAM_NAME	3036	X'00000BDC'
MQCACF_REG_TOPIC	3037	X'00000BDD'
MQCACF_REG_TIME	3038	X'00000BDE'
MQCACF_REG_USER_ID	3039	X'00000BDF'
MQCACF_CHILD_Q_MGR_NAME	3040	X'00000BE0'
MQCACF_REG_STREAM_NAME	3041	X'00000BE1'
MQCACF_REG_Q_MGR_NAME	3042	X'00000BE2'
MQCACF_REG_Q_NAME	3043	X'00000BE3'
MQCACF_REG_CORREL_ID	3044	X'00000BE4'

MQCMD_⋆ (Command identifiers for PCF)

MQCMD_DELETE_PUBLICATION	60	X'0000003C'
MQCMD_DEREGISTER_PUBLISHER	61	X'0000003D'
MQCMD_DEREGISTER_SUBSCRIBER	62	X'0000003E'
MQCMD_PUBLISH	63	X'0000003F'
MQCMD_REGISTER_PUBLISHER	64	X'00000040'
MQCMD_REGISTER_SUBSCRIBER	65	X'00000041'
MQCMD_REQUEST_UPDATE	66	X'00000042'
MQCMD_BROKER_INTERNAL	67	X'00000043'

MQDELO_⋆ (Delete options)

MQDELO_NONE	0	X'00000000'
MQDELO_LOCAL	4	X'00000004'

MQDT_⋆ (Destination type for routing exit)

MQDT_APPL	1	X'00000001'
MQDT_BROKER	2	X'00000002'

MQIACF_⋆ (Integer parameter identifiers for PCF)

MQIACF_INTEGER_DATA	1080	X'00000438'
MQIACF_REGISTRATION_OPTIONS	1081	X'00000439'
MQIACF_PUBLICATION_OPTIONS	1082	X'0000043A'
MQIACF_BROKER_COUNT	1088	X'00000440'
MQIACF_APPL_COUNT	1089	X'00000441'
MQIACF_ANONYMOUS_COUNT	1090	X'00000442'
MQIACF_REG_REG_OPTIONS	1091	X'00000443'
MQIACF_DELETE_OPTIONS	1092	X'00000444'

MQPUBO_★ (Publication options)

MQPUBO_NONE	0	X'00000000'
MQPUBO_CORREL_ID_AS_IDENTITY	1	X'00000001'
MQPUBO_RETAIN_PUBLICATION	2	X'00000002'
MQPUBO_OTHER_SUBSCRIBERS_ONLY	4	X'00000004'
MQPUBO_NO_REGISTRATION	8	X'00000008'
MQPUBO_IS_RETAINED_PUBLICATION	16	X'00000010'

MQREGO_★ (Registration options)

MQREGO_NONE	0	X'00000000'
MQREGO_CORREL_ID_AS_IDENTITY	1	X'00000001'
MQREGO_ANONYMOUS	2	X'00000002'
MQREGO_LOCAL	4	X'00000004'
MQREGO_DIRECT_REQUESTS	8	X'00000008'
MQREGO_NEW_PUBLICATIONS_ONLY	16	X'00000010'
MQREGO_PUBLISH_ON_REQUEST_ONLY	32	X'00000020'
MQREGO_DEREGISTER_ALL	64	X'00000040'
MQREGO_INCLUDE_STREAM_NAME	128	X'00000080'
MQREGO_INFORM_IF_RETAINED	256	X'00000100'

MQRFH_★ (Rules and formatting header)

MQRFH_NONE	0	X'00000000'
MQRFH_VERSION_1	1	X'00000001'
MQRFH_STRUC_LENGTH_FIXED	32	X'00000020'

MQUA_★ (User-attribute selectors for PCF)

MQUA_FIRST	65536	X'00010000'
MQUA_LAST	999999999	X'3B9AC9FF'

Reason codes

MQRC_RFH_★ (RFH reason codes)

MQRC_RFH_ERROR	2334	X'0000091E'
MQRC_RFH_STRING_ERROR	2335	X'0000091F'
MQRC_RFH_COMMAND_ERROR	2336	X'00000920'
MQRC_RFH_PARM_ERROR	2337	X'00000921'
MQRC_RFH_PARM_MISSING	2338	X'00000922'
MQRC_RFH_STRING_ERROR	2339	X'00000923'

MQRCCF_★ (PCF reason codes)

MQRCCF_MSG_LENGTH_ERROR	3016	X'00000BC8'
MQRCCF_MD_FORMAT_ERROR	3023	X'00000BCF'
MQRCCF_ENCODING_ERROR	3050	X'00000BEA'
MQRCCF_BROKER_DELETED	3070	X'00000BFE'
MQRCCF_STREAM_ERROR	3071	X'00000BFF'
MQRCCF_TOPIC_ERROR	3072	X'00000C00'
MQRCCF_NOT_REGISTERED	3073	X'00000C01'
MQRCCF_Q_MGR_NAME_ERROR	3074	X'00000C02'
MQRCCF_INCORRECT_STREAM	3075	X'00000C03'
MQRCCF_Q_NAME_ERROR	3076	X'00000C04'
MQRCCF_NO_RETAINED_MSG	3077	X'00000C05'
MQRCCF_DUPLICATE_IDENTITY	3078	X'00000C06'
MQRCCF_INCORRECT_Q	3079	X'00000C07'
MQRCCF_CORREL_ID_ERROR	3080	X'00000C08'
MQRCCF_NOT_AUTHORIZED	3081	X'00000C09'
MQRCCF_UNKNOWN_STREAM	3082	X'00000C0A'
MQRCCF_REG_OPTIONS_ERROR	3083	X'00000C0B'
MQRCCF_PUB_OPTIONS_ERROR	3084	X'00000C0C'
MQRCCF_UNKNOWN_BROKER	3085	X'00000C0D'
MQRCCF_Q_MGR_CCSID_ERROR	3086	X'00000C0E'
MQRCCF_DEL_OPTIONS_ERROR	3087	X'00000C0F'
MQRCCF_BROKER_COMMAND_FAILED	3094	X'00000C16'

Appendix D. Header files

This appendix lists the C-language header files necessary for publish/subscribe applications:

cmqpsc.h	Contains string constants for publish/subscribe messages using the MQRFH header.
cmqc.h	Contains elementary data types, and some named constants for events and PCF commands.
cmqcfc.h	Contains named constants specific to publish/subscribe messages, definitions for PCF structures, and additional named constants for events and PCF commands.
cmqbc.h	Contains definitions unique to the MQAI. This file is required only for metatopics and system management functions.

© Copyright IBM Corp. 1998, 2000

Appendix E. Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

© Copyright IBM Corp. 1998, 2000

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Trademarks

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX
IBM
MQSeries
MVS/ESA
OS/2
OS/390
SupportPac

Tivoli is a trademark of Tivoli Systems Inc. in the United States, other countries, or both.

Java is a trademark of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

Other company, product, and service names may be trademarks or service marks of others.

Glossary of terms and abbreviations

This glossary defines terms and abbreviations used in this book. If you do not find the term you are looking for, see the Index or the *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

B

Broker. An MQSeries application that receives messages from publishers on selected topics, and subscription requests from subscribers to a range of topics. The broker routes the published data to target subscribers. There is a maximum of one broker per queue manager.

Broker hierarchy. The topology of a *broker network*. Each broker (except the root broker) has a parent, and might have several children.

Broker network. A group of interconnected brokers that can share publications and subscription requests. The brokers are arranged in a *broker hierarchy*.

Broker queue. A queue on which a broker puts or receives messages. There are several types of broker queue, including system queues used for administration, and the queues used to hold publications.

C

Child broker. The brokers that sit below the current broker in the broker hierarchy. Brokers can have zero or more child brokers. Contrast with *parent broker*.

E

Event publication. A publication that contains information about an event, such as the sale of some stock. Different events are unrelated to each other. Contrast with *state publication*.

G

Global publication. A publication that is distributed through other brokers in the network to all subscribers, except those who have requested *local publications*.

L

Local publication. A publication that is available only to subscribers, on the same broker as the publisher, that have requested local publications. Contrast with *global publication*.

M

Metatopic. A special set of topics on which the broker publishes information about publishers and subscribers.

P

Parent broker. The broker that sits above the current broker in the broker hierarchy. Every broker except the root broker has a parent broker. Contrast with *child broker*.

Publish message. A message sent by a publisher to a broker, a broker to a broker, or a broker to a subscriber, that contains information being published.

Publisher. An application that makes information about a specified topic available.

Publisher queue. A queue to which any direct requests are sent by subscribers. This queue can also be used for response messages from the broker.

R

Registration message. A message sent to a broker, by a publisher or subscriber application, that registers that application's interest in a topic.

Response message. A message generated by a broker, in response to a message from a publisher or a subscriber, to indicate the success or failure of a request. Publishers and subscribers can request that they are always sent response messages, are never sent response messages, are sent a message only if an error occurs, or only if an error does not occur.

Retained publication. A publication that is retained by the broker so that it can be sent to new subscribers. When a new retained publication arrives for the same topic, it replaces the preceding retained publication.

© Copyright IBM Corp. 1998, 2000

S

State publication. A publication that contains the latest information about the state of something, such as the current price of stock. State publications are usually implemented as *retained publications*. Contrast with *event publication*.

Stream. A group of related topics.

Stream queue. A queue to which publications about a particular stream are sent. The queue has the same name as the stream, and there is one occurrence of the stream queue at each broker that supports the stream.

Subscriber. An application that requests information about a specified topic.

Subscriber queue. A queue on a subscriber to which published messages are sent by the broker. A subscriber can have more than one subscriber queue.

T

Topic. A character string that describes the nature of the data that is being published.

W

Wildcard. A special character that can be used to represent one or more characters. Any character or set of characters can be used to replace a wildcard character.

Bibliography

This section describes the documentation available for all current MQSeries products.

MQSeries cross-platform publications

Most of these publications, which are sometimes referred to as the MQSeries “family” books, apply to all MQSeries Level 2 products. The latest MQSeries Level 2 products are:

- MQSeries for AIX, V5.2
- MQSeries for AS/400, V5.2
- MQSeries for AT&T GIS UNIX, V2.2
- MQSeries for Compaq (DIGITAL) OpenVMS, V2.2.1.1
- MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for HP-UX, V5.2
- MQSeries for OS/2 Warp, V5.1
- MQSeries for OS/390, V5.2
- MQSeries for SINIX and DC/OSx, V2.2
- MQSeries for Sun Solaris, V5.2
- MQSeries for Sun Solaris, Intel Platform Edition, V5.1
- MQSeries for Tandem NonStop Kernel, V2.2.0.1
- MQSeries for VSE/ESA, V2.1.1
- MQSeries for Windows V2.0
- MQSeries for Windows V2.1
- MQSeries for Windows NT and Windows 2000, V5.2

The MQSeries cross-platform publications are:

- *MQSeries Brochure*, G511-1908
- *MQSeries: An Introduction to Messaging and Queuing*, GC33-0805
- *MQSeries Intercommunication*, SC33-1872
- *MQSeries Queue Manager Clusters*, SC34-5349
- *MQSeries Clients*, GC33-1632
- *MQSeries System Administration*, SC33-1873
- *MQSeries MQSC Command Reference*, SC33-1369
- *MQSeries Event Monitoring*, SC34-5760
- *MQSeries Programmable System Management*, SC33-1482
- *MQSeries Administration Interface Programming Guide and Reference*, SC34-5390
- *MQSeries Messages*, GC33-1876
- *MQSeries Application Programming Guide*, SC33-0807
- *MQSeries Application Programming Reference*, SC33-1673
- *MQSeries Programming Interfaces Reference Summary*, SX33-6095
- *MQSeries Using C++*, SC33-1877
- *MQSeries Using Java*, SC34-5456
- *MQSeries Application Messaging Interface*, SC34-5604

MQSeries platform-specific publications

Each MQSeries product is documented in at least one platform-specific publication, in addition to the MQSeries family books.

MQSeries for AIX, V5.2

MQSeries for AIX Quick Beginnings, GC33-1867

MQSeries for AS/400, V5.2

MQSeries for AS/400 Quick Beginnings, GC34-5557

MQSeries for AS/400 System Administration, SC34-5558

MQSeries for AS/400 Application Programming Reference (ILE RPG), SC34-5559

MQSeries for AT&T GIS UNIX, V2.2

MQSeries for AT&T GIS UNIX System Management Guide, SC33-1642

MQSeries for Compaq (DIGITAL) OpenVMS, V2.2.1.1

MQSeries for Compaq (DIGITAL) OpenVMS System Management Guide, GC33-1791

MQSeries for Compaq Tru64 UNIX, V5.1

MQSeries for Compaq Tru64 UNIX Quick Beginnings , GC34-5684

MQSeries for HP-UX, V5.2

MQSeries for HP-UX Quick Beginnings, GC33-1869

MQSeries for Linux, V5.2

MQSeries for Linux Quick Beginnings, GC34-5691

MQSeries for OS/2 Warp, V5.1

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

MQSeries for OS/390, V5.2

MQSeries for OS/390 Concepts and Planning Guide, GC34-5650

MQSeries for OS/390 System Setup Guide, SC34-5651

MQSeries for OS/390 System Administration Guide, SC34-5652

MQSeries for OS/390 Problem Determination Guide, GC34-5892

© Copyright IBM Corp. 1998, 2000

MQSeries for OS/390 Messages and Codes, GC34-5891

MQSeries for OS/390 Licensed Program Specifications, GC34-5893

MQSeries for OS/390 Program Directory

MQSeries link for R/3, V1.2

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

MQSeries for SINIX and DC/OSx, V2.2

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

MQSeries for Sun Solaris, V5.2

MQSeries for Sun Solaris Quick Beginnings, GC33-1870

MQSeries for Sun Solaris, Intel Platform Edition, V5.1

MQSeries for Sun Solaris, Intel Platform Edition Quick Beginnings, GC34-5851

MQSeries for Tandem NonStop Kernel, V2.2.0.1

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

MQSeries for VSE/ESA, V2.1.1

MQSeries for VSE/ESA Licensed Program Specifications, GC34-5365

MQSeries for VSE/ESA System Management Guide, GC34-5364

MQSeries for Windows, V2.0

MQSeries for Windows User's Guide, GC33-1822

MQSeries for Windows, V2.1

MQSeries for Windows User's Guide, GC33-1965

MQSeries for Windows NT and Windows 2000, V5.2

MQSeries for Windows NT Quick Beginnings, GC34-5389

MQSeries for Windows NT Using the Component Object Model Interface, SC34-5387

MQSeries LotusScript Extension, SC34-5404

MQSeries Integrator publications

The following books make up the MQSeries Integrator Version 2 library:

- *IBM MQSeries Integrator Version 2.0.1 Introduction and Planning*, GC34-5599
- *IBM MQSeries Integrator for Windows NT Version 2.0.1 Installation Guide*, GC34-5600
- *IBM MQSeries Integrator Version 2.0.1 Messages*, GC34-5601
- *IBM MQSeries Integrator Version 2.0.1 Using the Control Center*, SC34-5602
- *IBM MQSeries Integrator Version 2.0.1 Programming Guide*, SC34-5603
- *IBM MQSeries Integrator Version 2.0.1 Administration Guide*, SC34-5792
- *IBM MQSeries Integrator for Sun Solaris Version 2.0.1 Installation Guide*, GC34-5842
- *IBM MQSeries Integrator for AIX Version 2.0.1 Installation Guide*, GC34-5841

The *MQSeries Integrator for Windows NT Installation Guide* is provided in hardcopy with the product. The book is also available, separately, in hardcopy.

All books in the MQSeries Integrator library are provided in softcopy, in Adobe Portable Document Format (PDF) in a searchable PDF library for the Windows NT platform.

The PDF library is also supplied for UNIX platforms.

Softcopy books

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

HTML format

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

- MQSeries for AIX, V5.2
- MQSeries for AS/400, V5.2
- MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for HP-UX, V5.2
- MQSeries for OS/2 Warp, V5.1
- MQSeries for OS/390, V5.2
- MQSeries for Sun Solaris, V5.2
- MQSeries for Sun Solaris, Intel Platform Edition, V5.1
- MQSeries for Windows NT and Windows 2000, V5.2 (compiled HTML)
- MQSeries link for R/3 V1.2

The MQSeries books are also available in HTML format from the MQSeries product family Web site at:

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

Portable Document Format (PDF)

PDF files can be viewed and printed using the Adobe Acrobat Reader.

If you need to obtain the Adobe Acrobat Reader, or would like up-to-date information about the platforms on which the Acrobat Reader is supported, visit the Adobe Systems Inc. Web site at:

`http://www.adobe.com/`

PDF versions of relevant MQSeries books are supplied with these MQSeries products:

- MQSeries for AIX, V5.2
- MQSeries for AS/400, V5.2
- MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for HP-UX, V5.2
- MQSeries for OS/2 Warp, V5.1
- MQSeries for OS/390, V5.2
- MQSeries for Sun Solaris, V5.2
- MQSeries for Sun Solaris, Intel Platform Edition, V5.1
- MQSeries for Windows NT and Windows 2000, V5.2
- MQSeries link for R/3 V1.2

PDF versions of all current MQSeries books are also available from the MQSeries product family Web site at:

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

BookManager format

The MQSeries library is supplied in IBM BookManager format on a variety of online library collection kits, including the *Transaction Processing and Data* collection kit, SK2T-0730. You can view the softcopy books in IBM BookManager format using the following IBM licensed programs:

BookManager READ/2
BookManager READ/6000
BookManager READ/DOS
BookManager READ/MVS
BookManager READ/VM
BookManager READ for Windows

PostScript format

The MQSeries library is provided in PostScript (.PS) format with many MQSeries Version 2 products. Books in PostScript format can be printed on a PostScript printer or viewed with a suitable viewer.

Windows Help format

The *MQSeries for Windows User's Guide* is provided in Windows Help format with MQSeries for Windows Version 2.0 and MQSeries for Windows Version 2.1.

MQSeries information available on the Internet

The MQSeries product family Web site is at:

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

By following links from this Web site you can:

- Obtain latest information about the MQSeries product family.
- Access the MQSeries books in HTML and PDF formats.
- Download an MQSeries SupportPac.

Index

A
access control
  defining 111
  using streams 25
AccountingToken parameter
  publications forwarded by broker 46
adding a broker to a network 121
adding a stream 120
adding and removing brokers 123
AIX, installation on 12
AMQ58xx messages 179
amqsfmda.tst sample 111
appearance of text in this book xi
application
  sample program 102
Application Messaging Interface 47
application programming 37
application type for message descriptor, constants 195
applications, system management 149
ApplIdentityData parameter
  publications forwarded by broker 46
ApplOriginData parameter
  publications forwarded by broker 46
authorization checks 55, 111

B
backup 112
bibliography 207
BookManager 209
broker
  adding to a network 121
  administration messages 150
  backup 112
  configuration parameters 113
  configuration tool 112
  controlling 119
  deleting from a network 122
  deregistering as a publisher 53
  deregistering as subscriber 59
  exit program 137
  finding children 151
  finding out about 162
  finding parent 151
  finding supported streams 151
  interactions with subscriber and publisher 38
  introduction 3
  registering as a publisher 49
  registering as a subscriber 55
  response message 94
  routing exit 137
broker *(continued)*
  setting up 109
  stanza of qm.ini 112
broker deleted message 151
broker hierarchy, example 25
broker networks 25
broker queues, defining 109
broker response message 94

C
character parameter identifier constants 195
child broker 25
children messages 151
ChkPtActiveCount parameter 114
ChkPtMsgSize parameter 114
ChkPtRestartCount parameter 115
class of service 25
clear broker's memory, control command 126
clrmqbrk command 126
cluster queues 47, 109
CodedCharSetId field
  MQRFH structure 63
Command field
  MQCFH structure 152
command identifier constants 195
command message
  name/value pairs 71
  PCF format 149
  RFH format 61
  structure 37
Command parameter
  Broker response message 95
  Delete Publication command 72
  Deregister Publisher command 74
  Deregister Subscriber command 76
  Publish command 78
  Register Publisher command 83
  Register Subscriber command 86
  Request Update command 89
command tag string constants 193
Compaq Tru64 UNIX, installation on 14
CompCode field
  MQCFH structure 152
CompCode parameter
  Broker response message 94
compiling, routing exit 146
configuration file 112
constants 191
control commands
  clear broker's memory (clrmqbrk) 126
  deregister or delete broker function (dltmqbrk) 129

© Copyright IBM Corp. 1998, 2000

control commands *(continued)*
  display broker status (dspmqbrk) 131
  end broker function (endmqbrk) 132
  migrate broker to MQSeries Integrator function (migmqbrk) 133
  start broker function (strmqbrk) 135
Control field
  MQCFH structure 152
controlling brokers 119
CorrelId parameter
  message sent to broker 45
  publications forwarded by broker 46
  response messages 93
creating queues 109

D

data conversion 47
data, publication 68
dead-letter queue 111
dead-letter queue processing 91
defining queues 109
delete options
  integer constants 195
  string constants 193
Delete Publication command 72, 155
DeleteOptions parameter
  Broker response message 95
  Delete Publication command 72
deleting a broker from a network 122
deleting a stream 120
deleting publications 53
deregister or delete broker function, control command 129
Deregister Publisher command 74, 155
Deregister Subscriber command 76, 156
deregistering as a publisher 53
deregistering as a subscriber 59
destination type for routing exit, constants 195
DestinationQMgrName field
  MQPXP structure 145
DestinationQName field
  MQPXP structure 144
DestinationType field
  MQPXP structure 143
DiscardNonPersistentInputMsg parameter 116
DiscardNonPersistentPublication parameter 117
DiscardNonPersistentResponse parameter 116
disk space requirements for installation 11
display broker status, control command 131
DLQNonPersistentPublication parameter 117
DLQNonPersistentResponse parameter 116
dltmqbrk command 129
double-byte character sets 68
dspmqbrk command 131

E

Encoding field
  MQRFH structure 63
end broker function, control command 132
endmqbrk command 132
error codes 173
  Broker response message 97
  Delete Publication command 73
  Deregister Publisher command 75
  Deregister Subscriber command 77
  Publish command 82
  Register Publisher command 85
  Register Subscriber command 88
  Request Update command 90
error handling 91
error messages 179
error response 93
ErrorId parameter
  Broker response message 95
ErrorPos parameter
  Broker response message 95
event publications 28
example
  administration information program 166
  application program 102
  broker hierarchy 25
  Broker response message 96
  clrmqbrk command 127
  Delete Publication command 73
  Deregister Publisher command 75
  Deregister Subscriber command 77
  dltmqbrk command 130
  dspmqbrk command 131
  endmqbrk command 132
  metatopic information 167
  metatopic requests 161
  migmqbrk command 134
  multiple broker configuration 5
  multiple subscriptions 26
  NameValueString 65
  propagation of publications 27
  propagation of subscriptions 26
  publication data 68
  Publish command 82
  qm.ini broker stanza 112
  Register Publisher command 84
  Register Subscriber command 88
  Request Update command 90
  routing exit 146
  simple broker configuration 4
  strmqbrk command 136
exit program 137
ExitData field
  MQPXP structure 142

ExitId field
  MQPXP structure 140
ExitNumber field
  MQPXP structure 142
ExitParms parameter 138
ExitReason field
  MQPXP structure 140
ExitResponse field
  MQPXP structure 141
ExitResponse2 field
  MQPXP structure 142
ExitUserArea field
  MQPXP structure 142
Expiry parameter
  message sent to broker 45
  publications forwarded by broker 45

F

Feedback field
  MQPXP structure 142
Flags field
  MQRFH structure 64
Format field
  MQRFH structure 63
Format parameter
  message sent to broker 45
  publications forwarded by broker 45
  response messages 92

G

global publications
  introduction 28
  publishing 52
glossary 205
group messages 47
GroupId parameter 117

H

header files 199
HeaderLength field
  MQPXP structure 143
how to use this book xi
HP-UX 10, installation on 16
HP-UX 11 (DCE), installation on 18
HP-UX 11 (non-DCE), installation on 17
HTML (Hypertext Markup Language) 208
Hypertext Markup Language (HTML) 208

I

identity of publisher and subscriber 43
initialization file 112
installation
  AIX 12

installation *(continued)*
  Compaq Tru64 UNIX 14
  disk space requirements 11
  HP-UX 10 16
  HP-UX 11 17, 18
  Linux 20
  package 11
  prerequisites 10
  Sun Solaris 21
  Windows 2000 22
  Windows NT 22
integer parameter identifier constants 195
IntegerData parameter
  Publish command 82
internal queues 111

L

limitations 7, 8, 47
Linux, installation on 20
local publications
  introduction 28
  publishing 52

M

managing brokers 119
MaxMsgRetryCount parameter 113
message descriptor (MQMD)
  message sent to broker 44
  publications forwarded by broker 45
  response messages 92
message flow 38
message format
  broker response 94
  commands 61, 71
  metatopic 162
message order 42
messages
  broker administration 150
  error 179
  group 47
  response 92
  segmented 47
metatopics 160
  example 167
  sample program 166
migmqbrk command 133
migrate broker to MQSeries Integrator function, control command 133
MQ_PUBSUB_ROUTING_EXIT call 138
MQAT_⋆ constants 195
MQBACK, routing exit 146
MQCACF_⋆ constants 195
MQCFH structure 152

MQCFT_⋆ values 152
MQCMD_⋆ constants 195
MQCMIT, routing exit 146
MQDELO_⋆ constants 195
MQDISC, routing exit 146
MQDT_⋆ constants 195
MQFB_⋆ values 142
MQIACF_⋆ constants 195
MQMD (message descriptor)
  message sent to broker 44
  publications forwarded by broker 45
  response messages 92
MQPS_⋆ constants 191, 193
MQPUBO_⋆ constants 196
MQPXP structure 139
MQPXP_⋆ values 139
MQRC_RFH_⋆ constants 197
MQRCCF_* codes 173
MQRCCF_⋆ constants 197
MQREGO_⋆ constants 196
MQRFH 62
MQRFH_⋆ constants 194, 196
MQRFH_⋆ values 62, 64
MQRFH_DEFAULT 65
MQSeries Integrator, relationship with 8
MQSeries publications 207
MQSeries, relationship with 7
MQUA_⋆ constants 196
MQXCC_⋆ values 141
MQXR_⋆ values 140
MQXUA_⋆ values 142
MsgDescPtr field
  MQPXP structure 143
MsgId parameter
  response messages 93
MsgInLength field
  MQPXP structure 143
MsgInPtr field
  MQPXP structure 143
MsgOutLength field
  MQPXP structure 143
MsgOutPtr field
  MQPXP structure 144
MsgSeqNumber field
  MQCFH structure 152
MsgType parameter
  message sent to broker 44
  publications forwarded by broker 45
  response messages 92
multiple subscriptions, example 26

N

NameValueString 65
NameValueString field 64
network
  adding a broker 121
  broker 25
  deleting a broker 122

O

OK response 93
OpenCacheExpiry parameter 113
OpenCacheSize parameter 113

P

ParameterCount field
  MQCFH structure 153
ParameterId parameter
  Broker response message 95
parent broker 25
parent messages 151
PCF definitions
  command messages 155
  Delete Publication 155
  Deregister Publisher 155
  Deregister Subscriber 156
  Publish 156
  Register Publisher 157
  Register Subscriber 157
  Request Update 158
PDF (Portable Document Format) 209
persistence 46
Persistence parameter
  publications forwarded by broker 46
  response messages 93
Portable Document Format (PDF) 209
PostScript format 209
prerequisites for installation 10
Priority parameter
  publications forwarded by broker 46
  response messages 93
problem determination 98
publication data 68
publication options
  integer constants 196
  string constants 193
publication propagation, example 27
PublicationOptions parameter
  Broker response message 95
  Publish command 79
publications
  customizing 137
  deleting 53
  MQSeries 207
Publish command 78, 156
publish/subscribe
  command messages 61, 71
  exit structure 139

publish/subscribe *(continued)*
  string constants 191
PublishBatchInterval parameter 114
PublishBatchSize parameter 114
publisher
  broker restart 51
  changing registration 51
  deregistering with the broker 53
  exit program 137
  identity 43
  interactions with subscriber and broker 38
  introduction 3
  registering with the broker 49
  writing applications 49
publisher information messages 159
publishing information 51
PublishTimestamp parameter
  Publish command 81
PutApplName parameter
  publications forwarded by broker 46
  response messages 93
PutApplType
  publications forwarded by broker 46
PutApplType parameter
  response messages 93
PutDate parameter
  publications forwarded by broker 46
PutTime parameter
  publications forwarded by broker 46

Q

qm.ini 112
QMgrName field
  MQPXP structure 144
QMgrName parameter
  Broker response message 96
  Deregister Publisher command 75
  Deregister Subscriber command 77
  Publish command 81
  Register Publisher command 84
  Register Subscriber command 87
  Request Update command 89
QName parameter
  Broker response message 96
  Deregister Publisher command 75
  Deregister Subscriber command 77
  Publish command 81
  Register Publisher command 84
  Register Subscriber command 88
  Request Update command 90
queue manager initialization file 112
queues
  cluster 47
  dead letter 111
  internal 111

queues *(continued)*
  stream 110
  SYSTEM.BROKER.ADMIN.STREAM 110
  SYSTEM.BROKER.CONTROL.QUEUE 109
  SYSTEM.BROKER.DEFAULT.STREAM 109
  SYSTEM.BROKER.MODEL.STREAM 110

R

range delimiter constants 196
reason codes
  constants 197
  description 173
  PCF messages 153
Reason field
  MQCFH structure 153
Reason parameter
  Broker response message 94
ReasonText parameter
  Broker response message 94
Register Publisher command 83, 157
Register Subscriber command 86, 157
registering as a publisher 49
registering as a subscriber 55
registration
  changing for a publisher 51
  changing for subscriber 57
registration options
  integer constants 196
  string constants 193
RegistrationOptions parameter
  Broker response message 96
  Deregister Publisher command 74
  Deregister Subscriber command 76
  Publish command 78
  Register Publisher command 83
  Register Subscriber command 86
  Request Update command 89
ReplyToQ parameter
  message sent to broker 45
  publications forwarded by broker 46
ReplyToQMgr parameter
  message sent to broker 45
  publications forwarded by broker 46
Report parameter 92
  message sent to broker 44
  publications forwarded by broker 45
  response messages 92
request update
  message flow 39
Request Update command 89, 158
requesting information 58
response messages 92
retained publication
  introduction 28
  publishing 52

return codes
  clrmqbrk command 127
  dltmqbrk command 130
  dspmqbrk command 131
  endmqbrk command 132
  migmqbrk command 134
  strmqbrk command 136
RFH definitions
  Delete Publication 72
  Deregister Publisher 74
  Deregister Subscriber 76
  Publish 78
  Register Publisher 83
  Register Subscriber 86
  Request Update 89
root broker 25
routing exit 137
RoutingExitAuthorityCheck parameter 115
RoutingExitConnectType parameter 115
RoutingExitData parameter 115
RoutingExitPath parameter 115
rules and formatting header
  constants 194, 196
  definition 62
  use of 65

S

sample program
  administration information 166
  application 102
  Application Messaging Interface 105
  routing exit 146
security, setting up 111
segmented messages 47
SequenceNumber parameter
  Publish command 81
softcopy books 208
start broker function, control command 135
starting a broker 119
state publications 28
stopping a broker 119
stream
  adding 120
  deleting 120
  finding which are supported 151
  implementation 24
  introduction 3
  reasons for using 24
stream deleted message 150
stream queues 110
stream support messages 151
StreamName field
  MQPXP structure 144
StreamName parameter
  Broker response message 96

StreamName parameter *(continued)*
  Delete Publication command 72
  Deregister Publisher command 74
  Deregister Subscriber command 76
  Publish command 80
  Register Publisher command 84
  Register Subscriber command 87
  Request Update command 89
StreamsPerProcess parameter 113
StringData parameter
  Publish command 81
strmqbrk command 135
StrucId field
  MQPXP structure 139
  MQRFH structure 62
StrucLength field
  MQCFH structure 152
  MQRFH structure 63
structures
  MQCFH 152
  MQPXP 139
  MQRFH 62
subscriber
  broker restart 57
  changing registration 57
  deregistering with the broker 59
  identity 43
  interactions with publisher and broker 38
  introduction 3
  message arrival order 42
  registering with the broker 55
  writing applications 55
subscriber information messages 159
subscribing to metatopics 161
subscription deregistered message 150
subscription propagation, example 26
subscriptions
  passing between brokers 26
Sun Solaris, installation on 21
SyncPointIfPersistent parameter 116
system design 23
system management programs 149
SYSTEM.BROKER.ADMIN.STREAM 110
SYSTEM.BROKER.CONTROL.QUEUE 109
SYSTEM.BROKER.DEFAULT.STREAM 109
SYSTEM.BROKER.MODEL.STREAM 110

T

terminology used in this book 205
threads, routing exit 145
Topic parameter
  Broker response message 96
  Delete Publication command 72
  Deregister Publisher command 74
  Deregister Subscriber command 76

Topic parameter *(continued)*
  Publish command 78
  Register Publisher command 83
  Register Subscriber command 86
  Request Update command 89
topics
  introduction 3
  using wildcards 23
triggering a broker 119
Type field
  MQCFH structure 152

U

unit of work 46
UserId parameter
  Broker response message 96
  publications forwarded by broker 46

V

Version field
  MQCFH structure 152
  MQPXP structure 140
  MQRFH structure 62

W

Warning response 93
what you need to know to understand this book xi
who this book is for xi
wildcards 23
  using with metatopics 161
Windows 2000, installation on 22
Windows Help 209
Windows NT, installation on 22
writing applications 37

Sending your comments to IBM

MQSeries

Publish/Subscribe User's Guide

GC34-5269-06

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book. Please limit your comments to the information in this book and the way in which the information is presented.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

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

You can send your comments to IBM in any of the following ways:

- By mail, use the Readers' Comment Form
- By fax:
  - From outside the U.K., after your international access code use 44 1962 870229
  - From within the U.K., use 01962 870229
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever you use, ensure that you include:

- The publication number and title
- The page number or topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

Readers' Comments

MQSeries

Publish/Subscribe User's Guide

GC34-5269-06

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

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

Be sure to print your name and address below if you would like a reply.

Name	Address
Company or Organization	
Telephone	Email

MQSeries

MQSeries Publish/Subscribe User's Guide GC34-5269-06

You can send your comments POST FREE on this form from any one of these countries:

Australia
Belgium
Bermuda
Cyprus
Denmark
Finland
France
Germany
Greece
Hong Kong
Iceland
Israel
Italy
Luxembourg
Monaco
Netherlands
New Zealand
Norway
Portugal
Republic of Ireland
Singapore
Spain
Sweden
Switzerland
United Arab Emirates
United States of America

If your country is not listed here, your local IBM representative will be pleased to forward your comments to us. Or you can pay the postage and send the form direct to IBM (this includes mailing in the U.K.).

1 Cut along this line

2 Fold along this line

By air mail
Par avion

IBRS/CCRI NUMBER: PHQ - D/1348/SO

NE PAS AFFRANCHIR
NO STAMP REQUIRED

REPONSE PAYEE
GRANDE-BRETAGNE

IBM United Kingdom Laboratories
Information Development Department (MP095)
Hursley Park,
WINCHESTER, Hants
SO21 2ZZ United Kingdom

3 Fold along this line

From: Name ____________________
Company or Organization ____________________
Address ____________________
____
EMAIL ____________________
Telephone ____________________

1 Cut along this line

4 Fasten here with adhesive tape

Printed in U.S.A.

GC34-5269-06