7.5

Planning for IBM WebSphere MQ

Note

Before using this information and the product it supports, read the information in “Notices” on page 153.

This edition applies to version 7 release 5 of IBM® WebSphere® MQ and to all subsequent releases and modifications until otherwise indicated in new editions.

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

Contents

Planning.....5
IBM MQ and IBM MQ Appliance on premises considerations for GDPR readiness..... 5
Designing an IBM WebSphere MQ architecture.....14
Single queue manager architectures.....15
Single queue manager architectures.....15
Single queue manager architectures.....15
Single queue manager architectures.....15
Multiple queue manager architectures.....16
Point-to-point messaging.....19
Introduction to publish/subscribe.....20
Handling undelivered messages with the dead-letter queue handler..... 106
Planning for multiple installations..... 115
Choosing a primary installation..... 117
Planning your storage and performance requirements.....121
Disk space requirements..... 122
Planning file system support..... 123
IBM WebSphere MQ and UNIX System V IPC resources.....147
Shared memory on AIX.....148
Setting UNIX Process Priority values.....148
Planning your IBM WebSphere MQ client environment on HP Integrity NonStop Server.....149
Preparing the HP Integrity NonStop Server environment.....149
IBM WebSphere MQ and HP NonStop TMF.....149
Using HP NonStop TMF.....150

Notices.....153
Programming interface information.....154
Trademarks.....154

Planning

When planning your IBM WebSphere MQ environment, you must consider the IBM WebSphere MQ architecture that you want to configure, resource requirements, the need for logging, and backup facilities. Use the links in this topic to plan the environment where IBM WebSphere MQ runs.

Before you plan your IBM WebSphere MQ environment, familiarize yourself with the basic IBM WebSphere MQ concepts, see the topics in Technical overview.

Related concepts
Availability, recovery and restart
Related tasks
Migrating
Installing
Configuring
Administering WebSphere MQ
Making sure that messages are not lost (logging)

IBM MQ and IBM MQ Appliance on premises considerations for GDPR readiness

For PID(s):

- 5724-H72 IBM MQ
- 5655-AV9 IBM MQ Advanced for z/OS®
- 5655-AV1 IBM MQ Advanced for z/OS, Value Unit Edition
- 5655-AM9 IBM MQ Advanced Message Security for z/OS
- 5725-Z09 IBM MQ Appliance M2001
- 5725-S14 IBM MQ Appliance M2000
- 5655-MQ9 IBM MQ for z/OS
- 5655-VU9 IBM MQ for z/OS Value Unit Edition
- 5639-L92 IBM MQ Internet Pass-Thru
- 5655-MF9 IBM MQ Managed File Transfer for z/OS
- 5655-ADV IBM WebSphere MQ Advanced for z/OS
- 5655-AMS IBM WebSphere MQ Advanced Message Security for z/OS
- 5724-R10 IBM WebSphere MQ File Transfer Edition for Multiplatforms
- 5724-A39 IBM WebSphere MQ for HP NonStop Server
- 5724-A38 IBM WebSphere MQ for HP OpenVMS
- 5655-W97 IBM WebSphere MQ for z/OS
- 5655-VU8 IBM WebSphere MQ for z/OS Value Unit Edition
- 5655-VUE IBM WebSphere MQ for z/OS Value Unit Edition
- 5725-C79 IBM WebSphere MQ Hypervisor Edition for Red Hat Enterprise Linux® for x86
- 5725-F22 IBM WebSphere MQ Hypervisor for AIX®
- 5655-MFT IBM WebSphere MQ Managed File Transfer for z/OS

Notice:

This document is intended to help you in your preparations for GDPR readiness. It provides information about features of IBM MQ that you can configure, and aspects of the product's use, that you should consider to help your organization with GDPR readiness. This information is not an exhaustive list, due to the many ways that clients can choose and configure features, and the large variety of ways that the product can be used in itself and with third-party applications and systems.

Clients are responsible for ensuring their own compliance with various laws and regulations, including the European Union General Data Protection Regulation. Clients are solely responsible for obtaining advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulations that may affect the clients' business and any actions the clients may need to take to comply with such laws and regulations.

The products, services, and other capabilities described herein are not suitable for all client situations and may have restricted availability. IBM does not provide legal, accounting, or auditing advice or represent or warrant that its services or products will ensure that clients are in compliance with any law or regulation.

Table of Contents

1. GDPR
2. Product Configuration for GDPR
3. Data Life Cycle
4. Data Collection
5. Data Storage
6. Data Access
7. Data Processing
8. Data Deletion
9. Data Monitoring
10. Capability for Restricting Use of Personal Data
11. File handling

GDPR

General Data Protection Regulation (GDPR) has been adopted by the European Union ("EU") and applies from May 25, 2018.

Why is GDPR important?

GDPR establishes a stronger data protection regulatory framework for processing of personal data of individuals. GDPR brings:

- New and enhanced rights for individuals
- Widened definition of personal data
- New obligations for processors
- Potential for significant financial penalties for non-compliance
- Compulsory data breach notification

Read more about GDPR:

- EU GDPR Information Portal
- ibm.com/GDPR website

Product Configuration - considerations for GDPR Readiness

The following sections provide considerations for configuring IBM MQ to help your organization with GDPR readiness.

Data Life Cycle

IBM MQ is a transactional message oriented middleware product that enables applications to asynchronously exchange application provided data. IBM MQ supports a range of messaging APIs, protocols and bridges for the purpose of connecting applications. As such, IBM MQ may be used to exchange many forms of data, some of which could potentially be subject to GDPR. There are several third-party products with which IBM MQ might exchange data. Some of these are IBM-owned, but many others are provided by other technology suppliers. The Software Product Compatibility Reports website provides lists of the associated software. For considerations regarding the GDPR readiness of a third-party product, you should consult that product's documentation. IBM MQ administrators control the way in which IBM MQ interacts with data passing through it, by the definition of queues, topics and subscriptions.

What types of data flow through IBM MQ?

As IBM MQ provides asynchronous messaging service for application data, there is no one definitive answer to this question because use cases vary through application deployment. Application message data is persisted in queue files (pagesets or the Coupling Facility on z/OS), logs and archives and the message may itself contain data that is governed by GDPR. Application provided message data may also be included in files collected for problem determination purposes such as error logs, trace files and FFSTs. On z/OS application provided message data may also be included in address space or Coupling Facility dumps.

The following are some typical examples of personal data that may be exchanged using IBM MQ:

- Employees of the customer (for example; IBM MQ might be used to connect the customer's payroll or HR systems)
- The customer's own clients' personal data (for example; IBM MQ might be used by a customer to exchange data between applications that relates to their clients, such as taking sales leads and storing data inside their CRM system).
- The customer's own clients' sensitive personal data (for example; IBM MQ might be used within industry contexts that require personal data to be exchanged, such as HL7-based healthcare records when integrating clinical applications).

In addition to application provided message data, IBM MQ processes the following types of data:

- Authentication Credentials (such as username and passwords, API keys, etc.)
- Technically Identifiable Personal Information (such as device IDs, usage based identifiers, IP address, etc. - when linked to an individual)

Personal data used for online contact with IBM

IBM MQ clients can submit online comments/feedback/requests to contact IBM about IBM MQ subjects in a variety of ways, primarily:

- Public comments area on pages in the IBM MQ area on IBM Developer
- Public comments area on pages of IBM MQ product information in IBM Documentation
- Public comments in the IBM Support Forums
- Public comments in the IBM RFE Community on IBM Developer

Typically, only the client name and email address are used, to enable personal replies for the subject of the contact, and the use of personal data conforms to the IBM Online Privacy Statement.

Data Collection

IBM MQ can be used to collect personal data. When assessing your use of IBM MQ and your needs to meet with the demands of GDPR, you should consider the types of personal data which in your circumstances are passing through IBM. You may wish to consider aspects such as:

- How does data arrive into your queue managers? (Across which protocols? Is the data encrypted? Is the data signed?)
- How is data sent out from your queue managers? (Across which protocols? Is the data encrypted? Is the data signed?)
- How is data stored as it passes through a queue manager? (Any messaging application has the potential to write message data to stateful media, even if a message is non-persistent. Are you aware of how messaging features could potentially expose aspects of the application message data passing through the product?)
- How are credentials collected and stored where needed by IBM MQ to access third-party applications?

IBM MQ may need to communicate with other systems and services which require authentication, for example LDAP. Where needed, authentication data (userids, passwords) is configured and stored by IBM MQ for its use in such communications. Wherever possible, you should avoid using personal credentials for IBM MQ authentication. Consider the protection of the storage used for authentication data. (See Data Storage below.)

Data Storage

When message data travels through queue managers, IBM MQ will persist (perhaps multiple copies of) that data directly to stateful media. IBM MQ users may want to consider securing message data whilst it is at rest.

The following items highlight areas where IBM MQ persists application provided data, which users may wish to consider when ensuring compliance with GDPR.

- Application Message Queues:

  IBM MQ provides message queues to allow asynchronous data exchange between applications. Non-persistent and persistent messages stored on a queue are written to stateful media.

- File Transfer Agent Queues:

  IBM MQ Managed File Transfer utilizes message queues to co-ordinate the reliable transfer of file data, files that contain personal data and records of transfers are stored on these queues.

- Transmission Queues:

  To transfer messages reliably between queue managers, messages are stored temporarily on transmission queues.

- Dead-Letter Queues:

  There are some circumstances where messages cannot be put to a destination queue and are stored on a dead-letter queue if the queue manager has one configured.

- Backout Queues:

  JMS and XMS messaging interfaces provide a capability that allows poison messages to be moved to a backout queue after a number of backouts have occured to allow other valid messages to be processed.

- AMS Error Queue:

  IBM MQ Advanced Message Security will move messages that don't comply with a security policy to the SYSTEM.PROTECTION.ERROR.QUEUE error queue in a similar way to dead-letter queuing.

- Retained Publications:

  IBM MQ provides a retained publication feature to allow subscribing applications to recall a previous publication.

Read more:

- Logging: Making sure that messages are not lost
- MFT Agent queue settings
- Defining a transmission queue
- Using the dead-letter queue
- Handling poison messages in IBM MQ classes for JMS
- AMS error handling
- Retained publications

The following items highlight areas where IBM MQ may indirectly persist application provided data which users may also wish to consider when ensuring compliance with GDPR.

- Trace route messaging:

  IBM MQ provides trace route capabilities, which record the route a message takes between applications. The event messages generated may include technically identifiable personal information such as IP addresses.

- Application activity trace:

  IBM MQ provides application activity trace, which record the messaging API activities of applications and channels, application activity trace can record the contents of application provided message data to event messages.

- Service trace:

  IBM MQ provides service trace features, which records the internal code paths through which message data flows. As part of these features, IBM MQ can record the contents of application provided message data to trace files stored on disk.

- Queue manager events:

  IBM MQ can generate event messages that could include personal data, such as authority, command and configuration events.

Read more:

- Trace-route messaging
- Using trace
- Event monitoring
- Queue manager events

To protect access to copies of the application provided message data consider the following actions:

- Restrict privileged user access to IBM MQ data in the filesystem, for example restricting user membership of the 'mqm' group on UNIX platforms.
- Restrict application access to IBM MQ data via dedicated queues and access control. Where appropriate avoid unnecessary sharing of resources such as queues between applications and provide granular access control to queue and topic resources.
- Use IBM MQ Advanced Message Security to provide end-to-end signing and/or encryption of message data.
- Use file- or volume-level encryption to protect the contents of the directory used to store trace logs.
- After uploading service trace to IBM, you can delete your service trace files and FFST data if you are concerned about the contents potentially containing personal data.

Read more:

- Privileged users
- Planning file system support on Multiplatforms

An IBM MQ administrator may configure a queue manager with credentials (username and password, API keys, etc.) for 3rd party services such as LDAP, IBM Cloud® Product Insights, Salesforce, etc. This data is generally stored in the queue manager data directory protected through file system permissions.

When an IBM MQ queue manager is created, the data directory is set up with group-based access control such that IBM MQ can read the configuration files and use the credentials to connect to these systems. IBM MQ administrators are considered privileged users and are members of this group so have read access to the files. Some files are obfuscated but they are not encrypted. For this reason, to fully protect access to credentials, you should consider the following actions:

- Restrict privileged user access to IBM MQ data, for example restricting membership of the 'mqm' group on UNIX platforms.
- Use file- or volume-level encryption to protect the contents of the queue manager data directory.
- Encrypt backups of the production configuration directory and store them with appropriate access controls.
- Consider providing audit trails for authentication failure, access control and configuration changes with security, command and configuration events.

Read more:

- Securing IBM MQ

Data Access

IBM MQ queue manager data can be accessed through the following product interfaces, some of which are designed for access through a remote connection, and others for access through a local connection.

- IBM MQ Console [Only Remote]
- IBM MQ REST API [Only Remote]
- MQI [Local and Remote]
- JMS [Local and Remote]
- XMS [Local and Remote]
- IBM MQ Telemetry (MQTT) [Only Remote]
- IBM MQ Light (AMQP) [Only Remote]
- IBM MQ IMS bridge [Only Local]
- IBM MQ CICS bridge [Only Local]
- IBM MQ bridge for HTTP [Only Remote]
- IBM MQ MFT Protocol bridges [Only Remote]
- IBM MQ Connect:Direct bridges [Only Remote]
- IBM MQ Bridge to Salesforce [Only Remote]
- IBM MQ Bridge to Blockchain [Only Remote]
- IBM MQ MQAI [Local and Remote]
- IBM MQ PCF commands [Local and Remote]
- IBM MQ MQSC commands [Local and Remote]
- IBM MQ Explorer [Local and Remote]

The interfaces are designed to allow users to make changes to an IBM MQ queue manager and messages stored on it. Administration and messaging operations are secured such that there are three stages involved when a request is made;

- Authentication
- Role mapping
- Authorization

Authentication:

If the message or administrative operation was requested from a local connection, the source of this connection is a running process on the same system. The user running the process must have passed any authentication steps provided by the operating system. The user name of the owner of the process from which the connection was made is asserted as the identity. For example, this could be the name of the user running the shell from which an application has been started. The possible forms of authentication for local connections are:

1. Asserted user name (local OS)
2. Optional username and password (OS, LDAP or custom 3rd party repositories)

If the administrative action was requested from a remote connection, then communications with IBM MQ are made through a network interface. The following forms of identity may be presented for authentication via network connections;

1. Asserted user name (from remote OS)
2. Username and password (OS, LDAP or custom 3rd party repositories)
3. Source network address (such as IP address)
4. X.509 Digital Certificate (mutual SSL/TLS authentication)
5. Security tokens (such as LTPA2 token)
6. Other custom security (capability provided by 3rd party exits)

Role mapping:

In the role mapping stage, the credentials that were provided in the authentication stage may be mapped to an alternate user identifier. Provided the mapped user identifier is permitted to proceed (for example administrative users may be blocked by channel authentication rules), then the mapped user id is carried forward into the final stage when authorizing activities against IBM MQ resources.

Authorization:

IBM MQ provides the ability for different users to have different authorities against different messaging resources such as queues, topics and other queue manager objects.

Logging activity:

Some users of IBM MQ may need to create an audit record of access to MQ resources. Examples of desirable audit logs might include configuration changes that contain information about the change in addition to who requested it.

The following sources of information are available to implement this requirement:

1. An IBM MQ queue manager can be configured to produce command events when an admin command has been run successfully.
2. An IBM MQ queue manager can be configured to produce configuration events when a queue manager resource is created, altered or deleted.
3. An IBM MQ queue manager can be configured to produce an authority event when an authorization check fails for a resource.
4. Error messages indicating failed authorization checks are written to the queue manager error logs.
5. The IBM MQ Web console will write audit messages to its logs when authentication, authorization checks fail or when queue managers are created, started, stopped or deleted.

When considering these kind of solutions, IBM MQ users might want to give consideration to the following points:

- Event messages are non-persistent so when a queue manager restarts the information is lost. Any event monitors should be configured to constantly consume any available messages and transfer the content to persistent media.
- IBM MQ privileged users have sufficient privileges to disabled events, clear logs or delete queue managers.

For more information about securing access to IBM MQ data and providing an audit trail refer to the following topics:

- IBM MQ security mechanisms
- Configuration events
- Command events
- Error logs

Data Processing

Encryption using a Public Key Infrastructure:

You can secure network connections to IBM MQ to use TLS, which can also provide mutual authentication of the initiating side of the connection.

Using the PKI security facilities that are provided by transport mechanisms is the first step towards securing data processing with IBM MQ. However, without enabling further security features, the behavior of a consuming application is to process all messages delivered to it without validating the origin of the message or whether it was altered whilst in transit.

Users of IBM MQ that are licensed to use Advanced Message Security (AMS) capabilities can control the way in which applications process personal data held in messages, through the definition and configuration of security policies. Security policies allow digital signing and/or encryption to be applied to message data between applications.

It is possible to use security policies to require and validate a digital signature when consuming messages to ensure messages are authentic. AMS encryption provides a method by which message data is converted from a readable form to an encoded version that can only be decoded by another application if it is the intended recipient or the message and has access to the correct decryption key.

For more information about using SSL and certificates to secure your network connections, refer to the following topics in the IBM MQ V9 product documentation:

- Configuring TLS security for IBM MQ
- AMS Overview

Data Deletion

IBM MQ provides commands and user interface actions to delete data which has been provided to the product. This enables users of IBM MQ with facilities to delete data which relates to particular individuals, should this be required.

- Areas of IBM MQ behavior to consider for complying with GDPR Client Data deletion
  - Delete message data stored on an application queue by:
    - Removing individual messages using messaging API or tooling or by using message expiry.
    - Specifying that messages are non-persistent, held on a queue where non-persistent message class is normal and restarting the queue manager.
    - Administratively clearing the queue.
    - Deleting the queue.
  - Delete retained publication data stored on a topic by:
    - Specifying that messages are non-persistent and restarting the queue manager.
    - Replacing the retained data with new data or by using message expiry.
    - Administratively clearing the topic string.
  - Delete data stored on a queue manager by deleting the whole queue manager.
  - Delete data stored by the Service trace commands by deleting the files in the trace directory.
  - Delete FFST data stored by deleting the files in the errors directory.

  - Delete address space and Coupling Facility dumps (on z/OS).
  - Delete archive, backup or other copies of such data.
- Areas of IBM MQ behavior to consider for complying with GDPR Account Data deletion
  - You can delete account data and preferences stored by IBM MQ for connecting to queue managers and 3rd party services by deleting (including archive, backup or otherwise replicated copies thereof):
    - Queue manager authentication information objects that store credentials.
    - Queue manager authority records that reference user identifiers.
    - Queue manager channel authentication rules that map or block specific IP addresses, certificate DN's or user identifiers.
    - Credentials files used by IBM MQ Managed File Transfer Agent, Logger and MQ Explorer MFT Plugin for authentication with queue manager and file servers.
    - X.509 digital certificates that represent or contain information about an individual from keystores which may be used by SSL/TLS connections or IBM MQ Advanced Message Security (AMS).
    - Individual user accounts from IBM MQ Appliance, including reference to those accounts in system log files.
    - IBM MQ Explorer workspace metadata and Eclipse settings.
    - IBM MQ Explorer password store as specified in the Password Preferences.
    - IBM MQ Console and mqweb server configuration files.
    - Salesforce connection data configuration files.
    - blockchain connection data configuration files.
    - IBM Cloud Product Insights connection data under ReportingService stanza in qm.ini and APIKeyFile.

Read more:

- Configuring the IBM MQ Bridge to Salesforce
- Configuring IBM MQ for use with blockchain
- MFT and IBM MQ connection authentication
- Mapping credentials for a file server by using the ProtocolBridgeCredentials.xml file
- Configuring IBM MQ Console users and roles

Data Monitoring

IBM MQ provides a range of monitoring features that users can exploit to gain a better understanding of how applications and queue managers are performing.

IBM MQ also provides a number a features that help manage queue manager error logs.

Read more:

- Monitoring your IBM MQ network
- Diagnostic message services
- QMErrorLog service

IBM MQ provides a feature that enables users to publish information to an IBM Cloud Product Insights service, so that the IBM MQ user can view queue manager startup and usage information.

Read more:

- Configuring IBM MQ for use with IBM Cloud Product Insights service in IBM Cloud

Capability for Restricting Use of Personal Data

Using the facilities summarized in this document, IBM MQ enables an end-user to restrict usage of their personal data.

IBM MQ message queues should not be used as a permanent data store in the same way as a database, which is particularly true when handling application data that is subject to GDPR.

Unlike a database where data may be found through a search query, it can be difficult to find message data unless you know the queue, message and correlation identifiers of a message.

Provided messages containing an individual's data can be readily identified and located, it is possible using standard IBM MQ messaging features to access or modify message data.

File handling

1. IBM MQ Managed File Transfer does not perform malware scanning on files transferred. Files are transferred as-is and an integrity check is performed to ensure the file data is not modified during transfer. The source and destination checksums are published as part of transfer status publication. It is recommended that end users implement malware scanning as appropriate for their environment before MFT transfers the file and after MFT delivers a file to a remote end point.
2. IBM MQ Managed File Transfer does not take actions based on MIME type or file extension. MFT reads the files and transfers the bytes exactly as read from the input file.

Designing an IBM WebSphere MQ architecture

Find out about the different architectures that IBM WebSphere MQ supports for point-to-point and publish/subscribe messaging styles.

Before you plan your IBM WebSphere MQ architecture, familiarize yourself with the basic IBM WebSphere MQ concepts, see the topics in IBM WebSphere MQ Technical overview.

IBM WebSphere MQ architectures range from simple architectures using a single queue manager, to more complex networks of interconnected queue managers. Multiple queue managers are connected together using distributed queueing techniques. For more information about planning single queue manager and multiple queue manager architectures, see the following topics:

- “Architectures based on a single queue manager” on page 15
- “Architectures based on multiple queue managers” on page 16
- “Networks and Network Planning” on page 16
- WebSphere MQ distributed-messaging techniques

If you need multiple queue managers that are logically related and need to share data and applications they can be grouped together in a cluster. Using clusters can enable queue managers to communicate with each other without the need to set up extra channel definitions or remote queue definitions, simplifying their configuration and management. For more information about using clusters, see How clusters work.

Related concepts

“Planning” on page 5
When planning your IBM WebSphere MQ environment, you must consider the IBM WebSphere MQ architecture that you want to configure, resource requirements, the need for logging, and backup facilities. Use the links in this topic to plan the environment where IBM WebSphere MQ runs.

Related tasks

Configuring

Architectures based on a single queue manager

The simplest IBM WebSphere MQ architectures involve the configuration and use of a single queue manager.

Before you plan your IBM WebSphere MQ architecture, familiarize yourself with the basic IBM WebSphere MQ concepts, see Introduction to IBM WebSphere MQ.

A number of possible architectures using a single queue manager are described in the following sections:

- "Single queue manager with local applications accessing a service" on page 15
- "Single queue manager with remote applications accessing a service as clients" on page 15
- "Single queue manager with a publish/subscribe configuration" on page 15

Single queue manager with local applications accessing a service

The first architecture based on a single queue manager is where the applications accessing a service are running on the same system as the applications providing the service. An IBM WebSphere MQ queue manager provides asynchronous intercommunication between the applications requesting the service and the applications providing the service. This means that communication between the applications can continue even if one of the applications is offline for an extended period of time.

Single queue manager with remote applications accessing a service as clients

The second architecture based on a single queue manager has the applications running remotely from the applications providing the service. The remote applications are running on different systems to the services. The applications connect as clients to the single queue manager. This means that access to a service can be provided to multiple systems through a single queue manager.

A limitation of this architecture is that a network connection must be available for an application to operate. The interaction between the application and the queue manager over the network connection is synchronous.

Single queue manager with a publish/subscribe configuration

An alternative architecture using a single queue manager is to use a publish/subscribe configuration. In publish/subscribe messaging, you can decouple the provider of information from the consumers of that information. This differs from the point to point messaging styles in the previously described architectures, where the applications must know information about the target application, for example the queue name to put messages on. Using IBM WebSphere MQ publish/subscribe, the sending application publishes a message with a specified topic based on the subject of the information. IBM WebSphere MQ handles the distribution of the message to applications that have registered an interest in that subject through a subscription. The receiving applications also do not need to know anything about the source of the messages to receive them. For more information about publish/subscribe messaging, see Introduction to WebSphere MQ publish/subscribe messaging. For an example of publish/subscribe messaging using a single queue manager, see Example of a single queue manager publish/subscribe configuration.

Related concepts

"Designing an IBM WebSphere MQ architecture" on page 14
Find out about the different architectures that IBM WebSphere MQ supports for point-to-point and publish/subscribe messaging styles.

Related information

Introduction to WebSphere MQ

Creating and managing queue managers

Architectures based on multiple queue managers

You can use distributed message queuing techniques to create an IBM WebSphere MQ architecture involving the configuration and use of multiple queue managers.

Before you plan your IBM WebSphere MQ architecture, familiarize yourself with the basic IBM WebSphere MQ concepts, see Introduction to IBM WebSphere MQ.

An IBM WebSphere MQ architecture can be changed, without alteration to applications that provide services, by adding additional queue managers.

Applications can be hosted on the same machine as a queue manager, and then gain asynchronous communication with a service hosted on another queue manager on another system. Alternatively, applications accessing a service can connect as clients to a queue manager that then provides asynchronous access to the service on another queue manager.

Routes that connect different queue managers and their queues are defined using distributed queuing techniques. The queue managers within the architecture are connected using channels. Channels are used to move messages automatically from one queue manager to another in one direction depending on the configuration of the queue managers.

For a high level overview of planning an IBM WebSphere MQ network, see “Networks and Network Planning” on page 16.

For information about how to plan channels for your IBM WebSphere MQ architecture, see WebSphere MQ distributed-messaging techniques.

Distributed queue management enables you to create and monitor the communication between queue managers. For more information about distributed queue management, see Introduction to distributed queue management.

Related concepts

Introduction to WebSphere MQ

“Designing an IBM WebSphere MQ architecture” on page 14

Find out about the different architectures that IBM WebSphere MQ supports for point-to-point and publish/subscribe messaging styles.

Related tasks

Creating and managing queue managers

Networks and Network Planning

WebSphere MQ sends and receives data between applications, and over networks using Queue Managers and Channels. Network planning involves defining requirements to create a framework for connecting these systems over a network.

Channels can be created between your system and any other system with which you need to have communications. Multi-hop channels can be created to connect to systems where you have no direct connections. The message channel connections described in the scenarios are shown as a network diagram in Figure 1 on page 17.

Channel and transmission queue names

Transmission queues can be given any name. But to avoid confusion, you can give them the same names as the destination queue manager names, or queue manager alias names, as appropriate. This associates the transmission queue with the route they use, giving a clear overview of parallel routes created through intermediate (multi-hopped) queue managers.

It is not so clear-cut for channel names. The channel names in Figure 1 on page 17 for QM2, for example, must be different for incoming and outgoing channels. All channel names can still contain their transmission queue names, but they must be qualified to make them unique.

For example, at QM2, there is a QM3 channel coming from QM1, and a QM3 channel going to QM3. To make the names unique, the first one might be named 'QM3_from_QM1', and the second might be named

'QM3_from_QM2'. In this way, the channel names show the transmission queue name in the first part of the name. The direction and adjacent queue manager name are shown in the second part of the name.

A table of suggested channel names for Figure 1 on page 17 is given in Table 1 on page 17.

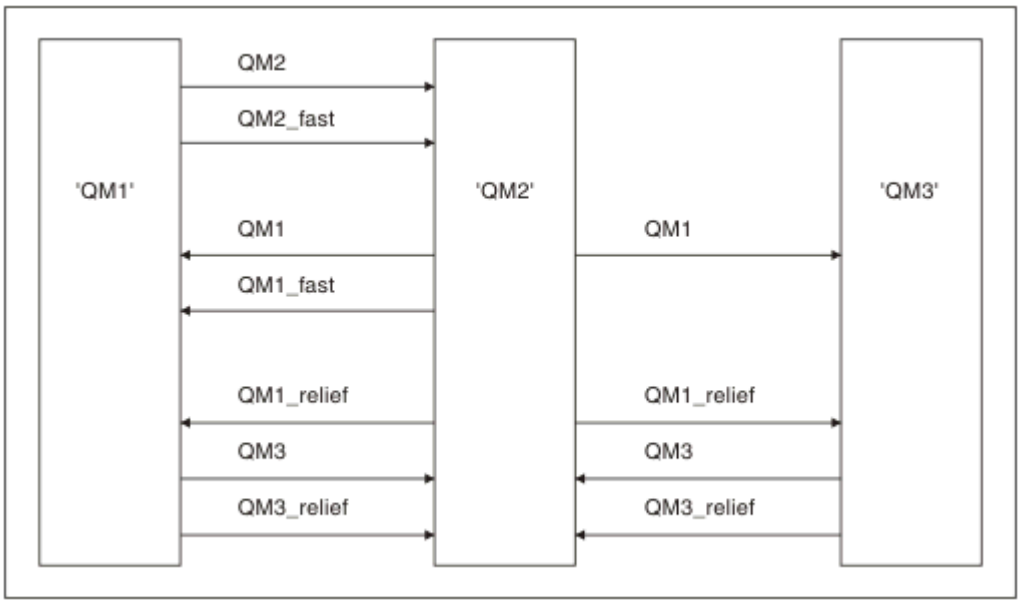


Figure 1. Network diagram showing all channels

Table 1. Example of channel names

Route name	Queue managers hosting channel	Transmission queue name	Suggested channel name
QM1	QM1 & QM2	QM1 (at QM2)	QM1.from.QM2
QM1	QM2 & QM3	QM1 (at QM3)	QM1.from.QM3
QM1_fast	QM1 & QM2	QM1_fast (at QM2)	QM1_fast.from.QM2
QM1_relief	QM1 & QM2	QM1_relief (at QM2)	QM1_relief.from.QM2
QM1_relief	QM2 & QM3	QM1_relief (at QM3)	QM1_relief.from.QM3
QM2	QM1 & QM2	QM2 (at QM1)	QM2.from.QM1
QM2_fast	QM1 & QM2	QM2_fast (at QM1)	QM2_fast.from.QM1
QM3	QM1 & QM2	QM3 (at QM1)	QM3.from.QM1
QM3	QM2 & QM3	QM3 (at QM2)	QM3.from.QM2
QM3_relief	QM1 & QM2	QM3_relief (at QM1)	QM3_relief.from.QM1
QM3_relief	QM2 & QM3	QM3_relief (at QM2)	QM3_relief.from.QM2

Note:

1. On WebSphere MQ for z/OS, queue manager names are limited to four characters.
2. Name all the channels in your network uniquely. As shown in Table 1 on page 17, including the source and target queue manager names in the channel name is a good way to do so.

Network planner

Creating a network assumes that there is another, higher level function of *network planner* whose plans are implemented by the other members of the team.

For widely used applications, it is more economical to think in terms of local access sites for the concentration of message traffic, using wide-band links between the local access sites, as shown in Figure 2 on page 18.

In this example there are two main systems and a number of satellite systems. Tthe actual configuration would depend on business considerations. There are two concentrator queue managers located at convenient centers. Each QM-concentrator has message channels to the local queue managers:

- QM-concentrator 1 has message channels to each of the three local queue managers, QM1, QM2, and QM3. The applications using these queue managers can communicate with each other through the QM-concentrators.
- QM-concentrator 2 has message channels to each of the three local queue managers, QM4, QM5, and QM6. The applications using these queue managers can communicate with each other through the QM-concentrators.
- The QM-concentrators have message channels between themselves thus allowing any application at a queue manager to exchange messages with any other application at another queue manager.

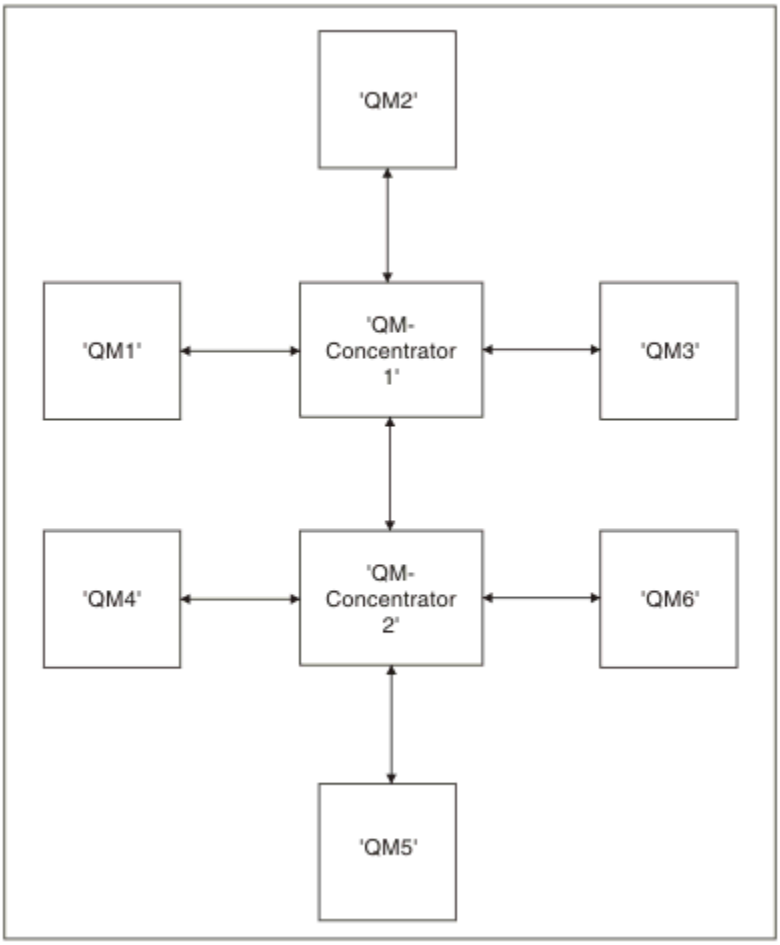


Figure 2. Network diagram showing QM-concentrators

Clustering

This topic provides guidance for planning and administering IBM WebSphere MQ clusters. This information is a guide based on testing and feedback from customers.

The following information assumes that the user has a basic understanding of IBM WebSphere MQ clusters. This information is not intended as a "one size fits all" solution, but is instead trying to share common approaches to common problems.

Clusters provide a mechanism for interconnecting queue managers in a way that simplifies the initial configuration required to set up the system and the ongoing management required. The larger the configuration, the greater the benefit.

Care is required in planning clustering systems to ensure that they function correctly and to ensure the levels of availability and responsiveness required by the system, especially for larger or more complex clustered systems.

A successful cluster setup is dependent on good planning and a thorough understanding of IBM WebSphere MQ fundamentals, such as good application management and network design. Ensure that you are familiar with the information in Concepts of intercommunication and How clusters work .

What are clusters and why are they used?

Clustering provides two key benefits:

- Clusters simplify the administration of IBM WebSphere MQ networks which usually require many object definitions for channels, transmit queues, and remote queues to be configured. This situation is especially true in large, potentially changing, networks where many queue managers need to be interconnected. This architecture is particularly hard to configure and actively maintain.
- Clusters can be used to distribute the workload of message traffic across queues and queue managers in the cluster. Such distribution allows the message workload of a single queue to be distributed across equivalent instances of that queue located on multiple queue managers. This distribution of the workload can be used to achieve greater resilience to system failures, and to improve the scaling performance of particularly active message flows in a system. In such an environment, each of the instances of the distributed queues have consuming applications processing the messages.

Related information

Clustering: Best practices

Point-to-point messaging

The simplest form of messaging in IBM WebSphere MQ is point-to-point messaging.

In point-to-point messaging, a sending application must know information about the receiving application before it can send a message to that application. For example, the sending application might need to know the name of the queue to which to send the information, and might also specify a queue manager name.

An alternative messaging style that you can use with IBM WebSphere MQ is publish/subscribe messaging. Publish/subscribe messaging allows you to decouple the provider of information, from the consumers of that information. The sending application and receiving application do not need to know anything about each other for the information to be sent and received. For more information about publish/subscribe messaging, see Introduction to WebSphere MQ publish/subscribe messaging.

Related information

Developing applications

WebSphere MQ messages

Introduction to IBM WebSphere MQ publish/subscribe messaging

Publish/subscribe messaging allows you to decouple the provider of information, from the consumers of that information. The sending application and receiving application do not need to know anything about each other for the information to be sent and received.

Before a point-to-point IBM WebSphere MQ application can send a message 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.

IBM WebSphere MQ publish/subscribe removes the need for your application to know anything about the target application. All the sending application has to do, is put a IBM WebSphere MQ message, containing the information that it wants, and assign it a topic, that denotes the subject of the information, and let IBM WebSphere MQ handle the distribution of that information. Similarly, the target application does not have to know anything about the source of the information it receives.

Figure 3 on page 20 shows the simplest publish/subscribe system. There is one publisher, one queue manager, and one subscriber. A subscription is sent from the subscriber to the queue manager, a publication is sent from the publisher to the queue manager, and the publication is then forwarded by the queue manager to the subscriber.

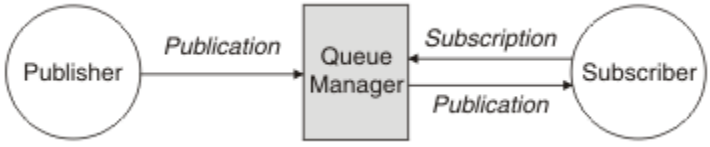


Figure 3. Simple publish/subscribe configuration

A typical publish/subscribe system has more than one publisher and more than one subscriber, and often, more than one queue manager. An application can be both a publisher and a subscriber.

Overview of publish/subscribe components

Publish/subscribe is the mechanism by which subscribers can receive information, in the form of messages, from publishers. The interactions between publishers and subscribers are controlled by queue managers, using standard WebSphere MQ facilities.

A typical publish/subscribe system has more than one publisher and more than one subscriber, and often, more than one queue manager. An application can be both a publisher and a subscriber.

The provider of information is called a *publisher*. Publishers supply information about a subject, without needing to know anything about the applications that are interested in that information. Publishers generate this information in the form of messages, called *publications* that they want to publish and define the topic of these messages.

The consumer of the information is called a *subscriber*. Subscribers create *subscriptions* that describe the topic that the subscriber is interested in. Thus, the subscription determines which publications are forwarded to the subscriber. Subscribers can make multiple subscriptions and can receive information from many different publishers.

Published information is sent in a WebSphere MQ message, and the subject of the information is identified by its *topic*. The publisher specifies the topic when it publishes the information, and the subscriber specifies the topics about which it wants to receive publications. The subscriber is sent information about only those topics it subscribes to.

It is the existence of topics that allows the providers and consumers of information to be decoupled in publish/subscribe messaging by removing the need to include a specific destination in each message as is required in point-to-point messaging.

Interactions between publishers and subscribers are all controlled by a queue manager. The queue manager receives messages from publishers, and subscriptions from subscribers (to a range of topics).

The queue manager's job is to route the published messages to the subscribers that have registered an interest in the topic of the messages.

Standard WebSphere MQ facilities are used to distribute messages, so your applications can use all the features that are available to existing WebSphere MQ 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 queue manager publish/subscribe configuration

Figure 4 on page 21 illustrates a basic single queue manager publish/subscribe configuration. The example shows the configuration for a news service, where information is available from publishers about several topics:

- 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 queue manager 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.

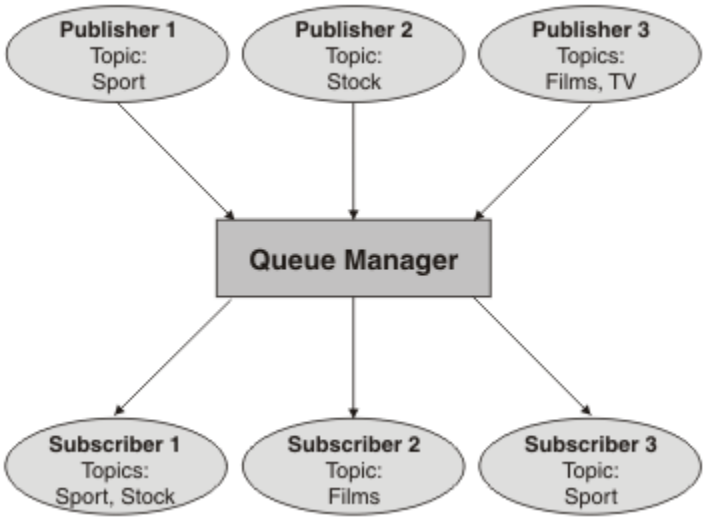


Figure 4. Single queue manager publish/subscribe example

Publishers and publications

In WebSphere MQ publish/subscribe a publisher is an application that makes information about a specified topic available to a queue manager in the form of a standard WebSphere MQ message called a publication. A publisher can publish information about more than one topic.

Publishers use the MQPUT verb to put a message to a previously opened topic, this message is a publication. The local queue manager then routes the publication to any subscribers who have subscriptions to the topic of the publication. A published message can be consumed by more than one subscriber.

In addition to distributing publications to all local subscribers that have appropriate subscriptions, a queue manager can also distribute the publication to any other queue managers connected to it, either directly or through a network of queue managers that have subscribers to the topic.

In a WebSphere MQ publish/subscribe network, a publishing application can also be a subscriber.

Publications under syncpoint

Publishers can issue MQPUT or MQPUT1 calls in syncpoint to include all messages delivered to subscribers in a unit of work. If the MQPMO_RETAIN option, or topic delivery options NPMSGDLV or PMSGDLV with values ALL or ALLDUR are specified, the queue manager uses internal MQPUT or MQPUT1 calls in syncpoint, within the scope of the publisher MQPUT or MQPUT1 call.

State and event information

Publications can be categorized as either state publications, such as the current price of a stock, or event publications, such as a trade in that stock.

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 because it is superseded by the new information.

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

If a publication contains state information, it is often published as a retained publication. A new subscriber typically wants the current state information immediately; the subscriber does not want to wait for an event that causes the information to be republished. Subscribers will automatically receive a topic's retained publication when it subscribes unless the subscriber uses the MQSO_PUBLICATIONS_ON_REQUEST or MQSO_NEW_PUBLICATIONS_ONLY options.

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 event is independent of other events.

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

Retained publications

By default, after a publication is sent to all interested subscribers it is discarded. However, a publisher can specify that a copy of a publication is retained so that it can be sent to future subscribers who register an interest in the topic.

Deleting publications after they have been sent to all interested subscribers is suitable for event information, but is not always suitable for state information. By retaining a message, new subscribers do not have to wait for information to be published again before they receive initial state information. For example, a subscriber with a subscription to a stock price would receive the current price straight away, without waiting for the stock price to change (and hence be republished).

The queue manager can retain only one publication for each topic, so the existing retained publication of a topic is deleted when a new retained publication arrives at the queue manager. However, the deletion of the existing publication might not occur synchronously with the arrival of the new retained publication. Therefore, wherever possible, have no more than one publisher sending retained publications on any topic.

Subscribers can specify that they do not want to receive retained publications by using the MQSO_NEW_PUBLICATIONS_ONLY subscription option. Existing subscribers can ask for duplicate copies of retained publications to be sent to them.

There are times when you might not want to retain publications, even for state information:

- If all subscriptions to a topic are made before any publications are made on that topic, and you do not expect, or do not allow, new subscriptions, there is no need to retain publications because they are delivered to the complete set of subscribers the first time they are published.
- If publications occur frequently, such as every second, a new subscriber (or a subscriber recovering from a failure) receives the current state almost immediately after their initial subscription, so there is no need to retain these publications.
- If the publications are large, you might end up needing a considerable amount of storage space to store the retained publication for each topic. In a multiple queue manager environment, retained publications are stored by all queue managers in the network that have a matching subscription.

When deciding whether to use retained publications, consider how subscribing applications recover from a failure. If the publisher does not use retained publications, the subscriber application might need to store its current state locally.

To ensure that a publication is retained, use the MQPMO_RETAIN put-message option. If this option is used and the publication cannot be retained, the message is not published and the call fails with MQRC_PUT_NOT_RETAINED.

If a message is a retained publication, this is indicated by the MQIsRetained message property. The persistence of a message is as it was when it was originally published.

Publications under syncpoint

In IBM WebSphere MQ publish/subscribe, syncpoint can be used by publishers or internally by the queue manager.

Publishers use syncpoint when they issue MQPUT/MQPUT1 calls with the MQPMO_SYNCPOINT option. All messages delivered to subscribers count towards the maximum number of uncommitted messages in a unit of work.The MAXUMSGS queue manager attribute specifies this limit. If the limit is reached then the publisher receives the 2024 (07E8) (RC2024): MQRC_SYNCPOINT_LIMIT_REACHED reason code.

When a publisher issues MQPUT/MQPUT1 calls using MQPMO_NO_SYNCPOINT with the MQPMO_RETAIN option, or topic delivery options NPMSGDLV/PMSGDLV with values ALL or ALLDUR, the queue manager uses internal syncpoints to guarantee that messages are delivered as requested. The publisher can receive the 2024 (07E8) (RC2024): MQRC_SYNCPOINT_LIMIT_REACHED reason code if the limit is reached within the scope of the publisher MQPUT/MQPUT1 call.

Subscribers and subscriptions

In WebSphere MQ publish/subscribe, a subscriber is an application that requests information about a specific topic from a queue manager in a publish/subscribe network. A subscriber can receive messages, about the same or different topics, from more than one publisher.

Subscriptions can be created manually using an MQSC command or by applications. These subscriptions are issued to the local queue manager and contain information about the publications the subscriber wants to receive:

- The topic the subscriber is interested in; this can resolve to multiple topics if wildcards are used.
- An optional selection string to be applied to published messages.
- A handle to a queue (known as the *subscriber queue*), on which selected publications should be placed, and the optional CorrelId.

The local queue manager stores subscription information and when it receives a publication, scans the information to determine whether there is a subscription that matches the publication's topic and selection string. For each matching subscription, the queue manager directs the publication to the subscriber's subscriber queue. The information that a queue manager stores about subscriptions can be viewed by using the DIS SUB and DIS SBSTATUS commands.

A subscription is deleted only when one of the following events occurs:

- The subscriber unsubscribes using the MQCLOSE call (if the subscription was made non-durably).

- The subscription expires.
- The subscription is deleted by the system administrator using the DELETE SUB command.
- The subscriber application ends (if the subscription was made non-durably).
- The queue manager is stopped or restarted (if the subscription was made non-durably).

When getting messages, use appropriate options on the MQGET call. If your application processes only messages for one subscription then, as a minimum, you should use `get-by-correlid`, as demonstrated in the C sample program `amqssbxa.c` and at unmanaged MQ subscriber. The **CorrelId** to use is returned from MQSUB in the MQSD.**SubCorrelId** field.

Managed queues and publish/subscribe

When you create a subscription you can choose to use managed queuing. If you use managed queueing a subscription queue is automatically created when you create a subscription. Managed queues are tidied up automatically in accordance with the durability of the subscription. Using managed queues means that you do not have to worry about creating queues to receive publications and any unconsumed publications are removed from subscriber queues automatically if a non-durable subscription connection is closed.

If an application has no need to use a particular queue as its subscriber queue, the destination for the publications it receives, it can make use of the *managed subscriptions* using the MQSO_MANAGED subscription option. If you create a managed subscription, the queue manager returns an object handle to the subscriber for a subscriber queue that the queue manager creates where publications will be received. The queue's object handle will be returned allowing you to browse, get or inquire on the queue (it is not possible to put to or set attributes of a managed queue unless you have been explicitly given access to temporary dynamic queues).

The durability of the subscription determines whether the managed queue remains when the subscribing application's connection to the queue manager is broken.

Managed subscriptions are particularly useful when used with non-durable subscriptions because when the application's connection is ended, unconsumed messages would otherwise remain on the subscriber queue taking up space in your queue manager indefinitely. If you are using a managed subscription, the managed queue will be a temporary dynamic queue and as such will be deleted along with any unconsumed messages when the connection is broken for any of the following reasons:

- MQCLOSE with MQCO_REMOVE_SUB is used and the managed Hobj is closed.
- a connection is lost to an application using a non-durable subscription (MQSO_NON_DURABLE).
- a subscription is removed because it has expired and the managed Hobj is closed.

Managed subscriptions can also be used with durable subscriptions but it is possible that you would want to leave unconsumed messages on the subscriber queue so that they can be retrieved when the connection is reopened. For this reason, managed queues for durable subscriptions take the form of a permanent dynamic queue and will remain when the subscribing application's connection to the queue manager is broken.

You can set an expiry on your subscription if you want to use permanent dynamic managed queue so that although the queue will still exist after the connection is broken, it will not continue to exist indefinitely.

If you delete the managed queue you will receive an error message.

The managed queues that are created are named with numbers at the end (timestamps) so that each is unique.

Subscription durability

Subscriptions can be configured to be durable or non-durable. Subscription durability determines what happens to subscriptions when subscribing applications disconnect from a queue manager.

Durable subscriptions

Durable subscriptions continue to exist when a subscribing application's connection to the queue manager is closed. If a subscription is durable, when the subscribing application disconnects, the subscription remains in place and can be used by the subscribing application when it reconnects

requesting the subscription again using the SubName that was returned when the subscription was created.

When subscribing durably, a subscription name (SubName) is required. Subscription names must be unique within a queue manager so that it can be used to identify a subscription. This means of identification is necessary when specifying a subscription you want to resume, if you have either deliberately closed the handle to the subscription (using the MQCO_KEEP_SUB option) or have been disconnected from the queue manager. You can resume an existing subscription by using the MQSUB call with the MQSO_RESUME option. Subscription names are also displayed if you use the DISPLAY SBSTATUS command with SUBTYPE ALL or ADMIN.

When an application no longer requires a durable subscription it can be removed using the MQCLOSE function call with the MQCO_REMOVE_SUB option or it can be deleted manually use the MQSC command DELETE SUB.

Whether durable subscriptions can be made to a topic can be controlled using the **DURSUB** topic attribute.

On return from an MQSUB call using the MQSO_RESUME option, subscription expiry is set to the original expiry of the subscription and not the remaining expiry time.

A queue manager continues to send publications to satisfy a durable subscription even if that subscriber application is not connected. This leads to a build up of messages on the subscriber queue. The easiest way to avoid this problem is to use a non-durable subscription wherever appropriate. However, where it is necessary to use durable subscriptions, a build up of messages can be avoided if the subscriber subscribes using the Retained publications option. A subscriber can then control when it receives publications by using the MQSUBRQ call.

Non-durable subscriptions

Non-durable subscriptions exist only as long as the subscribing application's connection to the queue manager remains open. The subscription is removed when the subscribing application disconnects from the queue manager either deliberately or by loss of connection. When the connection is closed, the information about the subscription is removed from the queue manager, and is no longer shown if you display subscriptions using the DISPLAY SBSTATUS command. No more messages are put to the subscriber queue.

What happens to any unconsumed publications on the subscriber queue for non-durable subscriptions is determined as follows.

- If a subscribing application is using a managed destination, any publications that have not been consumed are automatically removed.
- If the subscribing application provides a handle to its own subscriber queue when it subscribes, unconsumed messages are not removed automatically. It is the responsibility of the application to clear the queue if that is appropriate. If the queue is shared by more than one subscriber, or other point-to-point applications, it might not be appropriate to clear the queue completely.

Although not required for non durable subscriptions, a subscription name if provided, is used by the queue manager. Subscription names must be unique within the queue manager so that it can be used to identify a subscription.

Selection strings

A *selection string* is an expression that is applied to a publication to determine whether it matches a subscription. Selection strings can include wildcard characters.

When you subscribe, in addition to specifying a topic, you can specify a selection string to select publications according to their message properties.

Topics

A topic is the subject of the information that is published in a publish/subscribe message.

Messages in point-to-point systems are sent to a specific destination address. Messages in subject-based publish/subscribe systems are sent to subscribers based on the subject that describes the contents of the message. In content-based systems, messages are sent to subscribers based on the contents of the message itself.

The IBM WebSphere MQ publish/subscribe system is a subject-based publish/subscribe system. A publisher creates a message, and publishes it with a topic string that best fits the subject of the publication. To receive publications, a subscriber creates a subscription with a pattern matching topic string to select publication topics. The queue manager delivers publications to subscribers that have subscriptions that match the publication topic, and are authorized to receive the publications. The article, "Topic strings" on page 27, describes the syntax of topic strings that identify the subject of a publication. Subscribers also create topic strings to select which topics to receive. The topic strings that subscribers create can contain either of two alternative wildcard schemes to pattern match against the topic strings in publications. Pattern matching is described in "Wildcard schemes" on page 27.

In subject-based publish/subscribe, publishers, or administrators, are responsible for classifying subjects into topics. Typically subjects are organized hierarchically, into topic trees, using the `'/'` character to create subtopics in the topic string. See "Topic trees" on page 33 for examples of topic trees. Topics are nodes in the topic tree. Topics can be leaf-nodes with no further subtopics, or intermediate nodes with subtopics.

In parallel with organizing subjects into a hierarchical topic tree, you can associate topics with administrative topic objects. You assign attributes to a topic, such as whether the topic is distributed in a cluster, by associating it with an administrative topic object. The association is made by naming the topic using the `TOPICSTR` attribute of the administrative topic object. If you do not explicitly associate an administrative topic object to a topic, the topic inherits the attributes of its closest ancestor in the topic tree that you *have* associated with an administrative topic object. If you have not defined any parent topics at all, it inherits from `SYSTEM.BASE.TOPIC`. Administrative topic objects are described in "Administrative topic objects" on page 36.

Note: Even if you inherit all the attributes of a topic from `SYSTEM.BASE.TOPIC`, define a root topic for your topics that directly inherits from `SYSTEM.BASE.TOPIC`. For example, in the topic space of US states, `USA/Alabama` `USA/Alaska`, and so on, `USA` is the root topic. The main purpose of the root topic is to create discrete, non-overlapping topic spaces to avoid publications matching the wrong subscriptions. It also means you can change the attributes of your root topic to affect your whole topic space. For example, you might set the name for the **CLUSTER** attribute.

When you refer to a topic as a publisher or subscriber, you have a choice of supplying a topic string, referring to a topic object or you can do both, in which case the topic string you supply defines a subtopic of the topic object. The queue manager identifies the topic by appending the topic string to the topic string prefix named in the topic object, inserting an additional `'/'` in between the two topic strings, for example, *topic string/object string*. "Combining topic strings" on page 31 describes this further. The resulting topic string is used to identify the topic and associate it with an administrative topic object. The administrative topic object is not necessarily the same topic object as the topic object corresponding to the master topic.

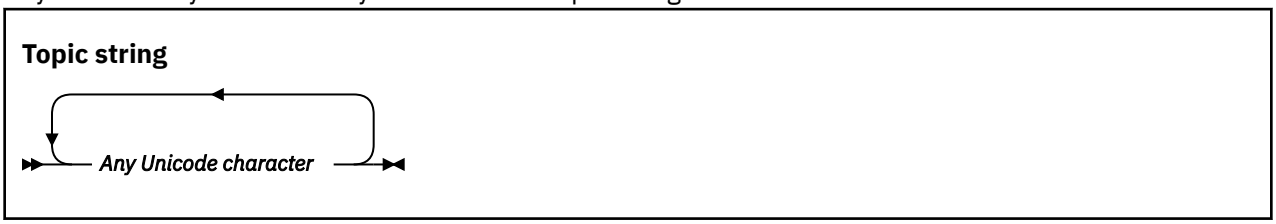
In content based publish/subscribe, you define what messages you want to receive by providing selection strings that search the contents of every message. WebSphere MQ provides an intermediate form of content based publish/subscribe using message selectors that scan message properties rather than the full content of the message, see Selectors. The archetypal use of message selectors is to subscribe to a topic and then qualify the selection with a numeric property. The selector enables you to specify you are interested in values only in a certain range; something you cannot do using either character or topic-based wildcards. If you do need to filter based on the full content of the message, you need to use WebSphere Message Broker.

Topic strings

Label information you publish as a topic using a topic string. Subscribe to groups of topics using either character or topic based wildcard topic strings.

Topics

A *topic string* is a character string that identifies the topic of a publish/subscribe message. You can use any characters you like when you construct a topic string.



Three characters have special meaning in version 7 publish/subscribe. They are allowed anywhere in a topic string, but use them with caution. The use of the special characters is explained in "Topic-based wildcard scheme" on page 28.

A forward slash (/)

The topic level separator. Use the `'/'` character to structure the topic into a topic tree.

Avoid empty topic levels, `'//'`, if you can. These correspond to nodes in the topic hierarchy with no topic string. A leading or trailing `'/'` in a topic string corresponds to a leading or trailing empty node and should be avoided too.

The hash sign (#)

Used in combination with `'/'` to construct a multilevel wildcard in subscriptions. Take care using `'#'` adjacent to `'/'` in topic strings used to name published topics. "Examples of topic strings" on page 27 shows a sensible use of `'#'`.

The strings `'.../#/...'`, `'#/...'` and `'.../#'` have a special meaning in subscription topic strings. The strings match all topics at one or more levels in the topic hierarchy. Thus if you created a topic with one of those sequences, you could not subscribe to it, without also subscribing to all topics at multiple levels in the topic hierarchy.

The plus sign (+)

Used in combination with `'/'` to construct a single-level wildcard in subscriptions. Take care using `'+'` adjacent to `'/'` in topic strings used to name published topics.

The strings `'.../+/...'`, `'+/...'` and `'.../+'` have a special meaning in subscription topic strings. The strings match all topics at one level in the topic hierarchy. Thus if you created a topic with one of those sequences, you could not subscribe to it, without also subscribing to all topics at one level in the topic hierarchy.

Examples of topic strings

```
IBM/Business Area#/Results
IBM/Diversity/%African American
```

Wildcard schemes

There are two wildcard schemes used to subscribe to multiple topics. The choice of scheme is a subscription option.

MQSO_WILDCARD_TOPIC

Select topics to subscribe to using the topic-based wildcard scheme.

This is the default if no wildcard schema is explicitly selected.

MQSO_WILDCARD_CHAR

Select topics to subscribe to using the character-based wildcard scheme.

Set either scheme by specifying the **wschema** parameter on the DEFINE SUB command. For more information, see DEFINE SUB .

Note: Subscriptions that were created before WebSphere MQ Version 7.0 always use the character-based wildcard scheme.

Examples

```
IBM/+/Results
#/Results
IBM/Software/Results
IBM/*ware/Results
```

Topic-based wildcard scheme
Topic-based wildcards allow subscribers to subscribe to more than one topic at a time.

Topic-based wildcards are a powerful feature of the topic system in WebSphere MQ publish/subscribe. The multilevel wildcard and single level wildcard can be used for subscriptions, but they cannot be used within a topic by the publisher of a message.

The topic-based wildcard scheme allows you to select publications grouped by topic level. You can choose for *each level in the topic hierarchy*, whether the string in the subscription for that topic level must exactly match the string in the publication or not. For example the subscription, `IBM/+/Results` selects all the topics,

```
IBM/Software/Results
IBM/Services/Results
IBM/Hardware/Results
```

There are two types of wildcard.

Multilevel wildcard

- The multilevel wildcard is used in subscriptions. When used in a publication it is treated as a literal.
- The multilevel wildcard character `'#'` is used to match any number of levels within a topic. For example, using the example topic tree, if you subscribe to `'USA/Alaska/#'`, you receive messages on topics `'USA/Alaska'` and `'USA/Alaska/Juneau'`.
- The multilevel wildcard can represent zero or more levels. Therefore, `'USA/#'` can also match the singular `'USA'`, where `'#'` represents zero levels. The topic level separator is meaningless in this context, because there are no levels to separate.
- The multilevel wildcard is only effective when specified on its own or next to the topic level separator character. Therefore, `'#'` and `'USA/#'` are valid topics where the `'#'` character is treated as a wildcard. However, although `'USA#'` is also a valid topic string, the `'#'` character is not regarded as a wildcard and does not have any special meaning. See “When topic-based wildcards are not wild” on page 30 for more information.

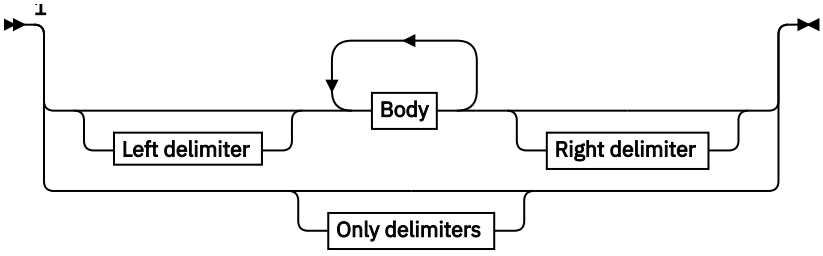
Single level wildcard

- The single wildcard is used in subscriptions. When used in a publication it is treated as a literal.
- The single-level wildcard character `'+'` matches one, and only one, topic level. For example, `'USA/+'` matches `'USA/Alabama'`, but not `'USA/Alabama/Auburn'`. Because the single-level wildcard matches only a single level, `'USA/+'` does not match `'USA'`.
- The single-level wildcard can be used at any level in the topic tree, and in conjunction with the multilevel wildcard. The single-level wildcard must be specified next to the topic level separator, except when it is specified on its own. Therefore, `'+'` and `'USA/+'` are valid topics where the `'+'` character is treated as a wildcard. However, although `'USA+'` is also a valid topic string, the `'+'` character is not regarded as a wildcard and does not have any special meaning. See “When topic-based wildcards are not wild” on page 30 for more information.

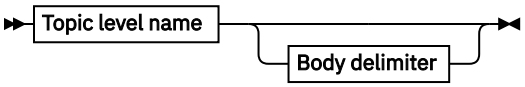
The syntax for the topic-based wildcard scheme has no escape characters. Whether `'#'` and `'+'` are treated as wildcards or not depends on their context. See “When topic-based wildcards are not wild” on page 30 for more information.

Note: The beginning and end of a topic string is treated in a special way. Using `'$'` to denote the end of the string, then `'$#/...'` is a multilevel wildcard, and `'$/#/..'`. is an empty node at the root, followed by a multilevel wildcard.

Topic-based wildcard string

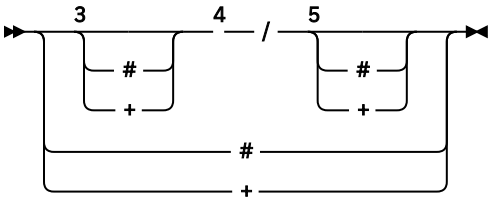


Body



Topic level name

Only delimiters



Notes:

[1] A null or zero length topic string is invalid

[2] You are advised not to use any of `*, ?, %` in level name strings for compatibility between char based and topic-based wildcard schemes.

[3] These cases are equivalent to the *left delimiter* pattern.

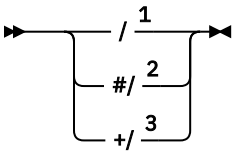
[4] / with no wildcards matches a single empty topic.

[5] These cases are equivalent to the *right delimiter* pattern.

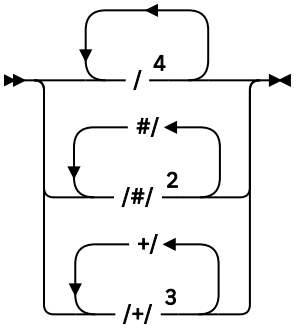
[6] Match every topic.

[7] Match every topic where there is only one level.

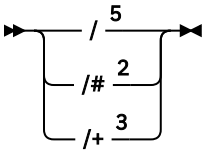
Left delimiter



Body delimiter



Right delimiter



Notes:

[1] The topic string starts with an empty topic
[2] Matches zero or more levels. Multiple multi-level match strings have the same effect as one multi-level match string.
[3] Matches exactly one level.
[4] // is an empty topic - a topic object with no topic string.
[5] The topic string ends with an empty topic

When topic-based wildcards are not wild

The wildcard characters '+' and '#' have no special meaning when they are mixed with other characters (including themselves) in a topic level.

This means that topics that contain '+' or '#' together with other characters in a topic level can be published.

For example, consider the following two topics:

1. level0/level1/+/level4/#
2. level0/level1/#+/level4/level#

In the first example, the characters '+' and '#' are treated as wildcards and are therefore not valid in a topic string that is to be published to but are valid in a subscription.

In the second example, the characters '+' and '#' are not treated as wildcards and therefore the topic string can be both published and subscribed to.

Examples

```
IBM/+/Results
#/Results
IBM/Software/Results
```

Character-based wildcard scheme
The character-based wildcard scheme allows you to select topics based on traditional character matching.

You can select all topics at multiple levels in a topic hierarchy using the string '*'. Using '*' in the character-based wildcard scheme is equivalent to using the topic-based wildcard string '#'

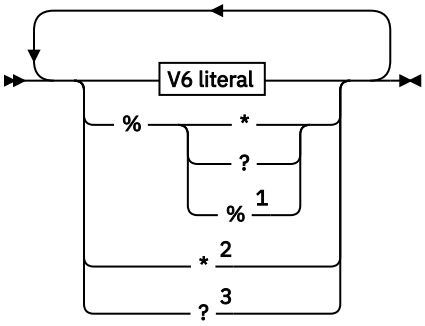
'*x*/*/*y*' is equivalent to '*x*/#/*y*' in the topic-based scheme, and selects all topics in the topic hierarchy between levels '*x* and *y*', where '*x*' and '*y*' are topic names that are not in the set of levels returned by the wildcard.

'/+/' in the topic-based scheme has no exact equivalent in the character-based scheme. 'IBM/*/Results' would also select 'IBM/Patents/Software/Results'. Only if the set of topic names at each level of the hierarchy are unique, can you always construct queries with the two schemes that yield identical matches.

Used in a general way, '*' and '?' in the character-based scheme have no equivalents in the topic-based scheme. The topic-based scheme does not perform partial matching using wildcards. The character based wildcard subscription 'IBM/*ware/Results' has no topic-based equivalent.

Note: Matches using character wildcard subscriptions are slower than matches using topic-based subscriptions.

Character-based wildcard string



V6 literal

Notes:

1. Means "Escape the following character", so that it is treated as a literal. '%' must be followed by either '*', '?' or '%'. See "Examples of topic strings" on page 27.
2. Means "Match zero or more characters" in a subscription.
3. Means "Match exactly one character" in a subscription.

Examples

```
IBM/*/Results
IBM/*ware/Results
```

Combining topic strings

When creating subscriptions, or opening topics so you can publish messages to them, the topic string can be formed by combining two separate sub-topic strings, or "subtopics". One subtopic is provided by the application or administrative command as a topic string, and the other is the topic string associated with a topic object. You can use either subtopic as the topic string on its own, or combine them to form a new topic name.

For example, when you define a subscription using the MQSC command **DEFINE SUB**, the command can take either **TOPICSTR** (topic string) or **TOPICOBJ** (topic object) as an attribute, or both together. If only **TOPICOBJ** is provided, the topic string associated with that topic object is used as the topic string. If only **TOPICSTR** is provided, that is used as the topic string. If both are provided, they are concatenated to form a single topic string in the form **TOPICOBJ**/**TOPICSTR**, where the **TOPICOBJ** configured topic string is always first and the two parts of the string are always separated by a "/" character.

Similarly, in an MQI program the full topic name is created by MQOPEN. It is composed of two fields used in publish/subscribe MQI calls, in the order listed:

1. The **TOPICSTR** attribute of the topic object, named in the **ObjectName** field.
2. The **ObjectString** parameter defining the subtopic provided by the application.

The resulting topic string is returned in the **ResObjectString** parameter.

These fields are considered to be present if the first character of each field is not a blank or null character, and the field length is greater than zero. If only one of the fields is present, it is used unchanged as the topic name. If neither field has a value, the call fails with reason code MQRC_UNKNOWN_OBJECT_NAME, or MQRC_TOPIC_STRING_ERROR if the full topic name is not valid.

If both fields are present, a "/" character is inserted between the two elements of the resultant combined topic name.

Table 2 on page 32 shows examples of topic string concatenation:

Table 2. Topic string concatenation examples

TOPICSTR of the topic object	Topic string provided by application or DEFINE SUB command	Full topic name	Comment
Football/Scores	' '	Football/Scores	The TOPICSTR of the topic object is used alone.
' '	Football/Scores	Football/Scores	The ObjectString/ TOPICSTR is used alone.
Football	Scores	Football/Scores	A "/" character is added at the concatenation point.
Football	/Scores	Football//Scores	An 'empty node' is produced between the two strings. This is different to "Football/ Scores".
/Football	Scores	/Football/Scores	The topic starts with an 'empty node'. This is different to "Football/ Scores".

The "/" character is considered as a special character, providing structure to the full topic name in “Topic trees” on page 33. The "/" character must not be used for any other reason, because the structure of the topic tree is affected. The topic "/Football" is not the same as the topic "Football".

Note: If you use a topic object when creating a subscription, the value of the topic object topic string is fixed in the subscription at define time. Any subsequent change to the topic object does not affect the topic string that the subscription is defined to.

Wildcard characters in topic strings

The following wildcard characters are special characters:

- plus sign (+)
- number sign (#)
- asterisk (*)
- question mark (?)

Wildcard characters only have special meaning when used by a subscription. These characters are not considered as invalid when used elsewhere, however you must ensure you understand how they are used and you might prefer not to use these characters in your topic strings when publishing or defining topic objects.

If you publish on a topic string with # or + mixed in with other characters (including themselves) within a topic level, the topic string can be subscribed to with either wildcard scheme.

If you publish on a topic string with # or + as the only character between two / characters, the topic string cannot be subscribed to explicitly by an application using the wildcard scheme MQSO_WILDCARD_TOPIC. This situation results in the application getting more publications than expected.

You should not use a wildcard character in the topic string of a defined topic object. If you do this, the character is treated as a literal character when the object is used by a publisher, and as a wildcard character when used by a subscription. This can lead to confusion.

Example code snippet

This code snippet, extracted from the example program Example 2: Publisher to a variable topic, combines a topic object with a variable topic string:

```
MQOD     td = {MQOD_DEFAULT};  /* Object Descriptor            */
td.ObjectType = MQOT_TOPIC;    /* Object is a topic            */
td.Version = MQOD_VERSION_4;   /* Descriptor needs to be V4    */
strncpy(td.ObjectName, topicName,  MQ_TOPIC_NAME_LENGTH);
td.ObjectString.VSPtr = topicString;
td.ObjectString.VSLength = (MQLONG)strlen(topicString);
td.ResObjectString.VSPtr = resTopicStr;
td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
```

Topic trees

Each topic that you define is an element, or node, in the topic tree. The topic tree can either be empty to start with or contain topics that have been defined previously using MQSC or PCF commands. You can define a new topic either by using the create topic commands or by specifying the topic for the first time in a publication or subscription.

Although you can use any character string to define a topic's topic string, it is advisable to choose a topic string that fits into a hierarchical tree structure. Thoughtful design of topic stings and topic trees can help you with the following operations:

- Subscribing to multiple topics.
- Establishing security policies.

Although you can construct a topic tree as a flat, linear structure, it is better to build a topic tree in a hierarchical structure with one or more root topics. For more information about security planning and topics, see Publish/subscribe security.

Figure 5 on page 33 shows an example of a topic tree with one root topic.

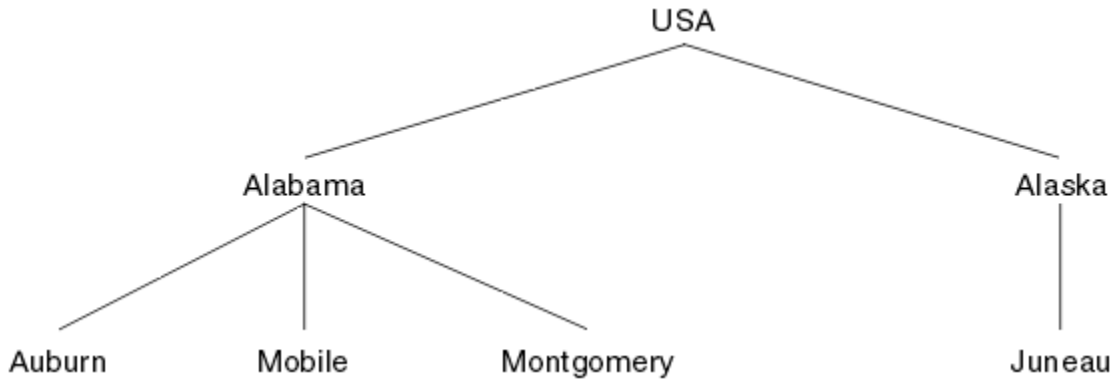


Figure 5. Example of a topic tree

Each character string in the figure represents a node in the topic tree. A complete topic string is created by aggregating nodes from one or more levels in the topic tree. Levels are separated by the "/" character. The format of a fully specified topic string is: "root/level2/level3".

The valid topics in the topic tree shown in Figure 5 on page 33 are:

"USA"
"USA/Alabama"
"USA/Alaska"
"USA/Alabama/Auburn"
"USA/Alabama/Mobile"
"USA/Alabama/Montgomery"
"USA/Alaska/Juneau"

When you design topic strings and topic trees, remember that the queue manager does not interpret, or attempt to derive meaning from, the topic string itself. It simply uses the topic string to send selected messages to subscribers of that topic.

The following principles apply to the construction and content of a topic tree:

- There is no limit to the number of levels in a topic tree.
- There is no limit to the length of the name of a level in a topic tree.
- There can be any number of "root" nodes; that is, there can be any number of topic trees.

Reducing the number of unwanted topics in the topic tree
The performance of a publish/subscribe system is improved by reducing the number of unwanted topics in the topic tree. What is an unwanted topic and how do you remove them?

You can create large numbers of topics without affecting performance adversely. However, some ways of using publish/subscribe result in continually expanding topic trees. An exceptionally large number of topics are created once and never used again. The growing number of topics might become a performance problem.

How can you avoid designs that lead to a large and growing number of unwanted topics? What can you do to help the queue manager remove unwanted topics from the topic tree?

The queue manager recognizes an unwanted topic because it has been unused for 30 minutes. The queue manager removes unused topics from the topic tree for you. The 30 minute duration can be changed by altering the queue manager attribute, **TREELIFE**. You can help the queue manager to remove unwanted topics by making sure that the topic appears to the queue manager to be unused. The section, “What is an unused topic?” on page 34 explains what an unused topic is.

A programmer, designing any application, and especially designing a long running application, considers its resource usage: how much resource the program requires, are there any unbounded demands, and any resource leaks? Topics are a resource that publish/subscribe programs use. Scrutinize the use of topics just like any other resource a program uses.

What is an unused topic?

Before defining what an unused topic is, what exactly counts as a topic?

When a topic string, such as `USA/Alabama/Auburn` , is converted into a topic, the topic is added to the topic tree. Additional topic nodes, and their corresponding topics, are created in the tree, if necessary. The topic string `USA/Alabama/Auburn` is converted into a tree with three topics.

- `USA`
- `USA/Alabama`
- `USA/Alabama/Auburn`

To display all the topics in the topic tree, use the **runmqsc** command `DISPLAY TPSTATUS('#') TYPE(TOPIC)` .

An unused topic in the topic tree has the following properties.

It is not associated with a topic object

An administrative topic object has a topic string that associates it with a topic. When you define the topic object `Alabama` , if the topic, `USA/Alabama`, it is to be associated with does not exist, the topic is created from the topic string. If the topic does exist, the topic object and the topic are associated together using the topic string.

It does not have a retained publication

A topic with a retained publication results from a publisher putting a message to a topic with the option `MQPMO_RETAIN` .

Use the **runmqsc** command `DISPLAY TPSTATUS('USA/Alabama') RETAINED` to check if `USA/Alabama` has a retained publication. The response is `YES` or `NO`.

Use the **runmqsc** command `CLEAR TOPICSTR('USA/Alabama') CLTRTYPE(RETAINED)` to remove a retained publication from `USA/Alabama` .

It has no child topics

`USA/Alabama/Auburn` is a topic with no child topics. `USA/Alabama/Auburn` is the direct child topic of `USA/Alabama`.

Display the direct children of `USA/Alabama` with the **runmqsc** command `DISPLAY TPSTATUS('USA/Alabama/+')` .

There are no active publishers to the node

An active publisher to a node is an application that has the topic open for output.

For example, an application opens the topic object named **Alabama** with open options `MQOO_OUTPUT`.

To display active publishers to `USA/Alabama` and all its children, use the **runmqsc** command `DISPLAY TPSTATUS('USA/Alabama/#') TYPE(PUB) ACTCONN`.

There are no active subscribers to the node

An active subscriber can either be a durable subscription, or an application that has registered a subscription to a topic with `MQSUB`, and not closed it.

To display active subscriptions to `USA/Alabama`, use the **runmqsc** command `DISPLAY TPSTATUS('USA/Alabama') TYPE(SUB) ACTCONN`.

To display active subscriptions to `USA/Alabama` and all its children, use the **runmqsc** command `DISPLAY TPSTATUS('USA/Alabama/#') TYPE(SUB) ACTCONN`.

Reducing the number of topics in a topic tree

In summary, there are a number of ways to reduce the number of topics in a topic tree.

Modify TREELIFE

An unused topic has a lifetime of 30 minutes by default. You can make the lifetime of an unused topic smaller.

For example, The **runmqsc** command, `ALTER QMGR TREELIFE(900)`, reduces lifetime of an unused topic from 30 minutes to 15 minutes.

Exceptionally, restart the queue manager

When the queue manager is restarted, the topic tree is reinitialized from topic objects, nodes with retained publications, and durable subscriptions. Topics that had been created by the operation of publisher and subscriber programs are eliminated.

Use the **runmqsc** command `DISPLAY TPSTATUS('#') TYPE(TOPIC)` periodically to list all topics and check if the number is growing.

As a last resort, if the growth in unwanted topics has been the cause of performance problems in the past, restart the queue manager.

Administrative topic objects

Using an administrative topic object, you can assign specific, non-default attributes to topics.

Figure 6 on page 36 shows how a high-level topic of `Sport` divided into separate topics covering different sports can be visualized as a topic tree:

Figure 6. Visualization of a topic tree

Figure 7 on page 36 shows how the topic tree can be divided further, to separate different types of information about each sport:

Figure 7. Extended topic tree

To create the topic tree illustrated, no administrative topic objects need be defined. Each of the nodes in this tree are defined by a topic string created in a publish or subscribe operation. Each topic in the tree inherits its attributes from its parent. Attributes are inherited from the parent topic object, because by default all attributes are set to ASPARENT. In this example, every topic has the same attributes as the `Sport` topic. The `Sport` topic has no administrative topic object, and inherits its attributes from SYSTEM.BASE.TOPIC.

Note, that it is not good practice to give authorities for non-mqm users at the root node of the topic tree, which is SYSTEM.BASE.TOPIC, because the authorities are inherited but cannot be restricted. Therefore, by giving authorities at this level, you are giving authorities to the whole tree. You should give the authority at a lower topic level in the hierarchy.

Administrative topic objects can be used to define specific attributes for particular nodes in the topic tree. In the following example, the administrative topic object is defined to set the durable subscriptions property DURSUBof the soccer topic to the value NO:

```
DEFINE TOPIC(FOOTBALL.EUROPEAN)
       TOPICSTR('Sport/Soccer')
       DURSUB(NO)
       DESCR('Administrative topic object to disallow durable subscriptions')
```

The topic tree can now be visualized as:

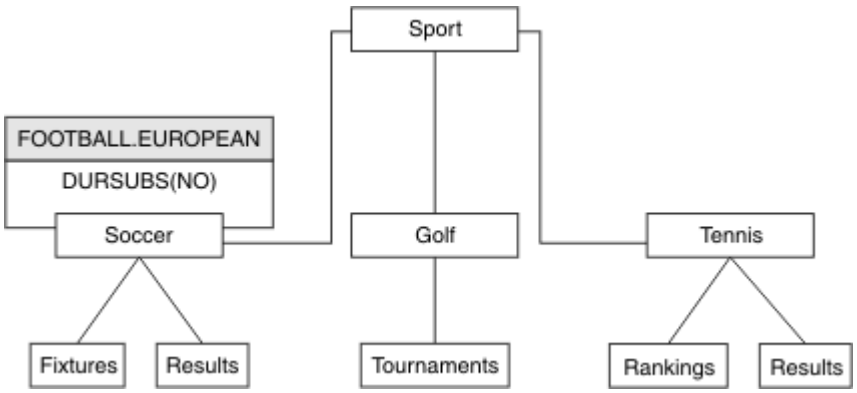


Figure 8. Visualization of an administrative topic object associated with the Sport/Soccer topic

Any applications subscribing to topics beneath Soccer in the tree can still use the topic strings they used before the administrative topic object was added. However, an application can now be written to subscribe using the object name `FOOTBALL.EUROPEAN` , instead of the string `/Sport/Soccer`. For example, to subscribe to `/Sport/Soccer/Results`, an application can specify `MQSD.ObjectName` as `FOOTBALL.EUROPEAN` and `MQSD.ObjectString` as `Results` .

With this feature, you can hide part of the topic tree from application developers. Define an administrative topic object at a particular node in the topic tree, then application developers can define their own topics as children of the node. Developers must know about the parent topic, but not about any other nodes in the parent tree.

Inheriting attributes

If a topic tree has many administrative topic objects, each administrative topic object, by default, inherits its attributes from its closest parent administrative topic. The previous example has been extended in Figure 9 on page 37:

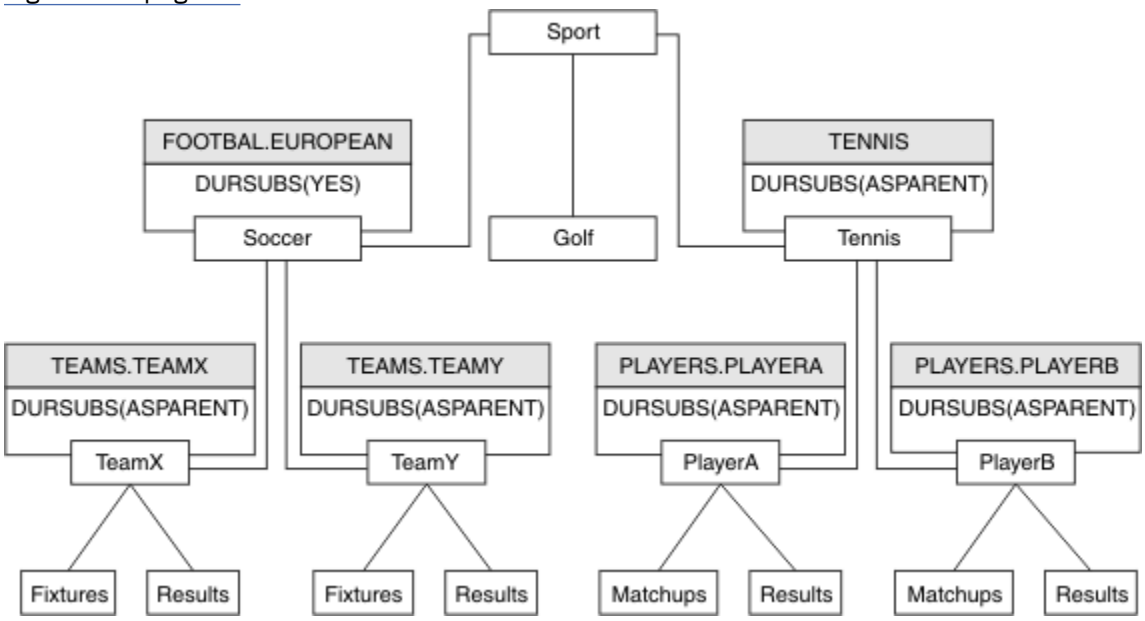


Figure 9. Topic tree with several administrative topic objects

For example use inheritance to give all the child topics of `/Sport/Soccer` the property that subscriptions are non-durable. Change the DURSUB attribute of `FOOTBALL.EUROPEAN` to `NO`.

This attribute can be set using the following command:

```
ALTER TOPIC(FOOTBALL.EUROPEAN) DURSUB(NO)
```

All the administrative topic objects of child topics of Sport/Soccer have the property DURSUB set to the default value ASPARENT. After changing the DURSUB property value of FOOTBALL.EUROPEAN to NO, the child topics of Sport/Soccer inherit the DURSUB property value NO . All child topics of Sport/Tennis inherit the value of DURSUB from SYSTEM.BASE.TOPIC object. SYSTEM.BASE.TOPIC has the value of YES.

Trying to make a durable subscription to the topic Sport/Soccer/TeamX/Results would now fail; however, trying to make a durable subscription to Sport/Tennis/PlayerB/Results would succeed.

Controlling wildcard usage with the WILDCARD property

Use the MQSC **Topic** WILDCARD property or the equivalent PCF Topic WildcardOperation property to control the delivery of publications to subscriber applications that use wildcard topic string names. The WILDCARD property can have one of two possible values:

WILDCARD
The behavior of wildcard subscriptions with respect to this topic.

PASSTHRU
Subscriptions made to a wildcarded topic less specific than the topic string at this topic object receive publications made to this topic and to topic strings more specific than this topic.

BLOCK
Subscriptions made to a wildcarded topic less specific than the topic string at this topic object do not receive publications made to this topic or to topic strings more specific than this topic.

The value of this attribute is used when subscriptions are defined. If you alter this attribute, the set of topics covered by existing subscriptions is not affected by the modification. This scenario applies also if the topology is changed when topic objects are created or deleted; the set of topics matching subscriptions created following the modification of the WILDCARD attribute is created using the modified topology. If you want to force the matching set of topics to be re-evaluated for existing subscriptions, you must restart the queue manager.

In the example, “Example: Create the Sport publish/subscribe cluster” on page 42, you can follow the steps to create the topic tree structure shown in Figure 10 on page 38.

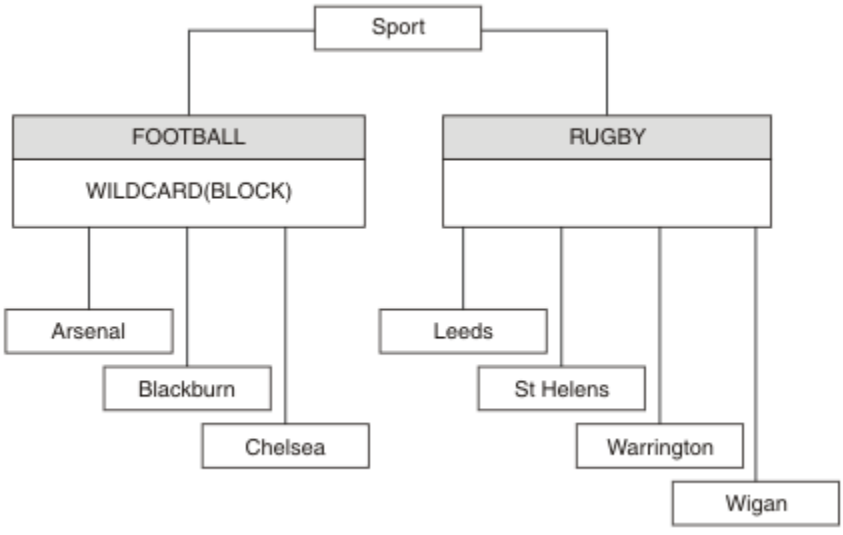


Figure 10. A topic tree that uses the WILDCARD property, BLOCK

A subscriber using the wildcard topic string # receives all publications to the Sport topic and the Sport/Rugby subtree. The subscriber receives no publications to the Sport/Football subtree, because the WILDCARD property value of the Sport/Football topic is BLOCK.

PASSTHRU is the default setting. You can set the WILDCARD property value PASSTHRU to nodes in the Sport tree. If the nodes do not have the WILDCARD property value BLOCK, setting PASSTHRU does not alter the behavior observed by subscribers to nodes in the Sports tree.

In the example, create subscriptions to see how the wildcard setting affects the publications that are delivered; see Figure 14 on page 43. Run the publish command in Figure 17 on page 44 to create some publications.

```
pub QMA
```

Figure 11. Publish to QMA

The results are shown in Table 3 on page 39. Notice how setting the WILDCARD property value BLOCK, prevents subscriptions with wildcards from receiving publications to topics within the scope of the wildcard.

Table 3. Publications received on QMA

Subscription	Topic string	Publications received	Notes
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	All publications to Football subtree blocked by WILDCARD(BLOCK) on Sports/Football
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) on Sports/Football prevents wildcard subscription on Arsenal
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	Default WILDCARD on Sports/Rugby does not prevent wildcard subscription on Leeds.

Note:

Suppose a subscription has a wildcard that matches a topic object with the WILDCARD property value BLOCK. If the subscription also has a topic string to the right of the matching wildcard, the subscription never receives a publication. The set of publications that are not blocked are publications to topics that are parents of the blocked wildcard. Publications to topics that are children of the topic with the BLOCK property value are blocked by the wildcard. Therefore subscription topic strings that include a topic to the right of the wildcard never receive any publications to match.

Setting the WILDCARD property value to BLOCK does not mean you cannot subscribe using a topic string that includes wildcards. Such a subscription is normal. The subscription has an explicit topic that matches the topic with a topic object having a WILDCARD property value BLOCK. It uses wildcards for topics that are parents or children of the topic with the WILDCARD property value BLOCK. In the example in Figure 10 on page 38, a subscription such as Sports/Football/# can receive publications.

Wildcards and cluster topics

Cluster topic definitions are propagated to every queue manager in a cluster. A subscription to a cluster topic at one queue manager in a cluster results in the queue manager creating proxy subscriptions. A proxy subscription is created at every other queue manager in the cluster. Subscriptions using topics strings containing wildcards, combined with cluster topics, can give hard to predict behavior. The behavior is explained in the following example.

In the cluster set up for the example, “Example: Create the Sport publish/subscribe cluster” on page 42, QMB has the same set of subscriptions as QMA, yet QMB received no publications after the publisher published to QMA, see Figure 11 on page 39. Although the Sports/Football and Sports/Rugby topics are cluster topics, the subscriptions defined in fullsubs.tst do not reference a cluster topic. No proxy

subscriptions are propagated from QMB to QMA. Without proxy subscriptions, no publications to QMA are forwarded to QMB .

Some of the subscriptions, such as Sports/#/Leeds , might seem to reference a cluster topic, in this case Sports/Rugby . The Sports/#/Leeds subscription actually resolves to the topic object SYSTEM.BASE.TOPIC.

The rule for resolving the topic object referenced by a subscription such as, Sports/#/Leeds is as follows. Truncate the topic string to the first wildcard. Scan left through the topic string looking for the first topic that has an associated administrative topic object. The topic object might specify a cluster name, or define a local topic object. In the example, Sports/#/Leeds, the topic string after truncation is Sports, which has no topic object, and so Sports/#/Leeds inherits from SYSTEM.BASE.TOPIC, which is a local topic object.

To see how subscribing to clustered topics can change the way wildcard propagation works, run the batch script, upsubs.bat. The script clears the subscription queues, and adds the cluster topic subscriptions in fullsubs.tst. Run puba.bat again to create a batch of publications; see Figure 11 on page 39.

Table 4 on page 40 shows the result of adding two new subscriptions to the same queue manager that the publications were published on. The result is as expected, the new subscriptions receive one publication each, and the numbers of publications received by the other subscriptions are unchanged. The unexpected results occur on the other cluster queue manager; see Table 5 on page 41.

Table 4. Publications received on QMA

Subscription	Topic string	Publications received	Notes
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	All publications to Football subtree blocked by WILDCARD(BLOCK) on Sports/Football
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) on Sports/Football prevents wildcard subscription on Arsenal
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	Default WILDCARD on Sports/Rugby does not prevent wildcard subscription on Leeds.
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal receives a publication because the subscription does not have a wildcard.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds would receive a publication in any event.

Table 5 on page 41 shows the results of adding the two new subscriptions on QMB and publishing on QMA. Recall that QMB received no publications without these two new subscriptions. As expected, the two new subscriptions receive publications, because Sports/FootBall and Sports/Rugby are both cluster topics. QMB forwarded proxy subscriptions for Sports/Football/Arsenal and Sports/Rugby/Leeds to QMA, which then sent the publications to QMB.

The unexpected result is that the two subscriptions Sports/# and Sports/#/Leeds that previously received no publications, now receive publications. The reason is that the Sports/Football/Arsenal and Sports/Rugby/Leeds publications forwarded to QMB for the other subscriptions are now available for any subscriber attached to QMB . Consequently the subscriptions to the local topics Sports/# and Sports/#/Leeds receive the Sports/Rugby/Leeds publication. Sports/#/Arsenal continues not to receive a publication, because Sports/Football has its WILDCARD property value set to BLOCK.

Table 5. Publications received on QMB			
Subscription	**Topic string**	**Publications received**	**Notes**
SPORTS	*Sports/#*	*Sports/Rugby/Leeds*	*All publications to Football subtree blocked by WILDCARD(BLOCK) on Sports/Football*
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) on Sports/Football prevents wildcard subscription on Arsenal
SLEEDS	*Sports/#/Leeds*	*Sports/Rugby/Leeds*	*Default WILDCARD on Sports/Rugby does not prevent wildcard subscription on Leeds .*
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal receives a publication because the subscription does not have a wildcard.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds would receive a publication in any event.

In most applications, it is undesirable for one subscription to influence the behavior of another subscription. One important use of the WILDCARD property with the value BLOCK is to make the subscriptions to the same topic string containing wildcards behave uniformly. Whether the subscription is on the same queue manager as the publisher, or a different one, the results of the subscription are the same.

Wildcards and streams

WebSphere MQ Version 6 streams are mapped to topics by WebSphere MQ Version 7; see “Streams and topics” on page 44 . In the default mapping, which is performed by **strmqbrk** in Version 7, all the topics in the stream Sports would be mapped to the topic Sports. All the topics in the stream Business would be mapped to the topic Business.

A subscription in WebSphere MQ Version 6 to * in the Sports stream receives all publications in the Sports tree, and no publications in the Business tree. The same subscription in version 7 would receive all the publications in the Sports tree and all the publications in the Business tree. To block this behavior, when streams are migrated to version 7, **strmqbrk** sets the WILDCARD property. It sets it to the value BLOCK for each of the top-level topics that are migrated from a stream. The WILDCARD property of Sports and Business is set to the value BLOCK by the conversion from the version 6 streams called Sports and Business.

For a new application written to the publish/subscribe API, the effect is that a subscription to * receives no publications. To receive all the Sports publications you must subscribe to Sports/*, or Sports/# , and similarly for Business publications.

The behavior of an existing queued publish/subscribe application does not change when the publish/subscribe broker is migrated to WebSphere MQ Version 7. The **StreamName** property in the **Publish**, **Register Publisher**, or **Subscriber** commands is mapped to the name of the topic the stream has been migrated to.

Wildcards and subscription points

WebSphere Message Broker subscriptions points are mapped to topics by WebSphere MQ Version 7; see “Subscription points and topics” on page 47. In the default mapping, which is performed by **migmqbrk** in Version 7, all the topics in the subscription point Sports would be mapped to the topic Sports. All the topics in the subscription point Business would be mapped to the topic Business.

A subscription on WebSphere Message Broker Version 6 to * in the Sports subscription point receives all publications in the Sports tree, and no publications in the Business tree. The same subscription in version 7 would receive all the publications in the Sports tree and all the publications in the Business tree. To block this behavior, when subscription points are migrated to version 7, **migmqbrk** sets the WILDCARD property. It sets it to the value BLOCK for each of the top-level topics that are migrated from a subscription point. The WILDCARD property of Sports and Business is set to the value BLOCK by the conversion from the WebSphere Message Broker subscription points called Sports and Business .

For a new application written to the publish/subscribe API, the effect of the migration is that a subscription to * receives no publications. To receive all the Sports publications you must subscribe to Sports/*, or Sports/# , and similarly for Business publications.

The behavior of an existing queued publish/subscribe application does not change when the publish/subscribe broker is migrated to WebSphere MQ Version 7. The **SubPoint** property in the **Publish**, **Register Publisher**, or **Subscriber** commands is mapped to the name of the topic the subscription has been migrated to.

Example: Create the Sport publish/subscribe cluster

The steps that follow create a cluster, CL1, with four queue managers: two full repositories, CL1A and CL1B , and two partial repositories, QMA and QMB. The full repositories are used to hold only cluster definitions. QMA is designated the cluster topic host. Durable subscriptions are defined on both QMA and QMB.

Note: The example is coded for Windows. You must recode Create qmgrs.bat and create pub.bat to configure and test the example on other platforms.

1. Create the script files.
   a. Create topics.tst
   b. Create wildsubs.tst
   c. Create fullsubs.tst
   d. Create qmgrs.bat
   e. create pub.bat
2. Run Create qmgrs.bat to create the configuration.

```
qmgrs
```

Create the topics in Figure 10 on page 38. The script in figure 5 creates the cluster topics Sports/Football and Sports/Rugby.

Note: The REPLACE option does not replace the TOPICSTR properties of a topic. TOPICSTR is a property that is usefully varied in the example to test different topic trees. To change topics, delete the topic first.

```
DELETE TOPIC ('Sports')
DELETE TOPIC ('Football')
DELETE TOPIC ('Arsenal')
DELETE TOPIC ('Blackburn')
DELETE TOPIC ('Chelsea')
DELETE TOPIC ('Rugby')
DELETE TOPIC ('Leeds')
DELETE TOPIC ('Wigan')
DELETE TOPIC ('Warrington')
DELETE TOPIC ('St. Helens')

DEFINE TOPIC ('Sports')     TOPICSTR('Sports')
DEFINE TOPIC ('Football')   TOPICSTR('Sports/Football') CLUSTER(CL1) WILDCARD(BLOCK)
DEFINE TOPIC ('Arsenal')    TOPICSTR('Sports/Football/Arsenal')
DEFINE TOPIC ('Blackburn')  TOPICSTR('Sports/Football/Blackburn')
DEFINE TOPIC ('Chelsea')    TOPICSTR('Sports/Football/Chelsea')
DEFINE TOPIC ('Rugby')      TOPICSTR('Sports/Rugby')    CLUSTER(CL1)
DEFINE TOPIC ('Leeds')      TOPICSTR('Sports/Rugby/Leeds')
DEFINE TOPIC ('Wigan')      TOPICSTR('Sports/Rugby/Wigan')
DEFINE TOPIC ('Warrington') TOPICSTR('Sports/Rugby/Warrington')
DEFINE TOPIC ('St. Helens') TOPICSTR('Sports/Rugby/St. Helens')
```

Figure 12. Delete and create topics: topics.tst

Note: Delete the topics, as REPLACE does not replace topic strings.

Create subscriptions with wildcards. The wildcards corresponding the topics with topic objects in Figure 10 on page 38. Create a queue for each subscription. The queues are cleared and the subscriptions deleted when the script is run or rerun.

Note: The REPLACE option does not replace TOPICOBJ or TOPICSTR properties of a subscription. TOPICOBJ or TOPICSTR are the properties that are usefully varied in the example to test different subscriptions. To change them, delete the subscription first.

```
DEFINE QLOCAL(QSPORTS)   REPLACE
DEFINE QLOCAL(QSARSENAL) REPLACE
DEFINE QLOCAL(QSLEEDS)   REPLACE
CLEAR  QLOCAL(QSPORTS)
CLEAR  QLOCAL(QSARSENAL)
CLEAR  QLOCAL(QSLEEDS)

DELETE SUB   (SPORTS)
DELETE SUB   (SARSENAL)
DELETE SUB   (SLEEDS)
DEFINE SUB   (SPORTS)   TOPICSTR('Sports/#')         DEST(QSPORTS)
DEFINE SUB   (SARSENAL) TOPICSTR('Sports/+/Arsenal') DEST(QSARSENAL)
DEFINE SUB   (SLEEDS)   TOPICSTR('Sports/+/Leeds')   DEST(QSLEEDS)
```

Figure 13. Create wildcard subscriptions: wildsubs.tst

Create subscriptions that reference the cluster topic objects.

Note:

The delimiter, /, is automatically inserted between the topic string referenced by TOPICOBJ, and the topic string defined by TOPICSTR.

The definition, `DEFINE SUB(FARSENAL) TOPICSTR('Sports/Football/Arsenal') DEST(QFARSENAL)` creates the same subscription. TOPICOBJ is used as a quick way to reference topic string you have already defined. The subscription, when created, no longer refers to the topic object.

```
DEFINE QLOCAL(QFARSENAL) REPLACE
DEFINE QLOCAL(QRLEEDS)   REPLACE
CLEAR  QLOCAL(QFARSENAL)
CLEAR  QLOCAL(QRLEEDS)

DELETE SUB   (FARSENAL)
DELETE SUB   (RLEEDS)
DEFINE SUB   (FARSENAL) TOPICOBJ('Football') TOPICSTR('Arsenal') DEST(QFARSENAL)
DEFINE SUB   (RLEEDS)   TOPICOBJ('Rugby')    TOPICSTR('Leeds')   DEST(QRLEEDS)
```

Figure 14. Delete and create subscriptions: fullsubs.tst

Create a cluster with two repositories. Create two partial repositories for publishing and subscribing. Rerun the script to delete everything and start again. The script also creates the topic hierarchy, and the initial wildcard subscriptions.

Note:

On other platforms, write a similar script, or type all the commands. Using a script makes it quick to delete everything and start again with an identical configuration.

```
@echo off
set port.CL1B=1421
set port.CLIA=1420
for %%A in (CL1A CL1B QMA QMB) do call :createQM %%A
call :configureQM CL1A CL1B %port.CL1B% full
call :configureQM CL1B CL1A %port.CL1A% full
for %%A in (QMA QMB) do call :configureQM %%A CL1A %port.CL1A% partial
for %%A in (topics.tst wildsubs.tst) do runmqsc QMA < %%A
for %%A in (wildsubs.tst) do runmqsc QMB < %%A
goto:eof

:createQM
echo Configure Queue manager %1
endmqm -p %1
for %%B in (dlt crt str) do %%Bmqm %1
goto:eof

:configureQM
if %1==CL1A set p=1420
if %1==CL1B set p=1421
if %1==QMA  set p=1422
if %1==QMB  set p=1423
echo configure %1 on port %p% connected to repository %2 on port %3 as %4 repository
echo DEFINE LISTENER(LST%1) TRPTYPE(TCP) PORT(%p%) CONTROL(QMGR) REPLACE | runmqsc %1
echo START LISTENER(LST%1) | runmqsc %1
if full==%4 echo ALTER  QMGR REPOS(CL1) DEADQ(SYSTEM.DEAD.LETTER.QUEUE) | runmqsc %1
echo DEFINE CHANNEL(TO.%2) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('LOCALHOST(%3)') CLUSTER(CL1)
REPLACE | runmqsc %1
echo DEFINE CHANNEL(TO.%1) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('LOCALHOST(%p%)')
CLUSTER(CL1) REPLACE | runmqsc %1
goto:eof
```

Figure 15. Create queue managers: qmgrs.bat

Update the configuration by adding the subscriptions to the cluster topics.

```
@echo off
for %%A in (QMA QMB) do runmqsc %%A < wildsubs.tst
for %%A in (QMA QMB) do runmqsc %%A < upsubs.tst
```

Figure 16. Update subscriptions: upsubs.bat

Run `pub.bat`, with a queue manager as a parameter, to publish messages containing the publication topic string. `Pub.bat` uses the sample program **amqspub**.

```
@echo off
@rem Provide queue manager name as a parameter
set S=Sports
set S=6 Sports/Football Sports/Football/Arsenal
set S=6 Sports/Rugby Sports/Rugby/Leeds
for %%B in (6) do echo %%B | amqspub %%B %1
```

Figure 17. Publish: pub.bat

Streams and topics

Queued publish/subscribe has the concept of a publication stream that does not exist in the integrated publish/subscribe model. In queued publish/subscribe, streams provide a way of separating the flow of information for different topics. In IBM WebSphere MQ Version 6.0, a stream is implemented as a queue, defined at each broker that supports the stream. Each queue has the same name (the name of the stream). From IBM WebSphere MQ Version 7.0 onwards, a stream is implemented as a top-level topic that can be mapped to a different topic identifier administratively.

The default stream SYSTEM.BROKER.DEFAULT.STREAM is set up automatically for all brokers and queue managers on a network, and no additional configuration is required to use the default stream. Think of the default stream as an unnamed default topic space. Topics published to the default stream are immediately available to all connected Version 6.0 brokers, and all queue managers from Version 7.0 onwards, with queued publish/subscribe enabled. Named streams are like separate, named, topic spaces. The named stream must be defined on each broker where it is used.

When you define a topic, the topic is available to Version 6.0 publish/subscribe brokers, and to publishers and subscribers running on a later version of IBM WebSphere MQ, with no special configuration.

If the publishers and subscribers are on different queue managers, then after the brokers are connected in the same broker hierarchy, no further configuration is required for the publications, and subscriptions to flow between them. The same interoperability works in reverse, too.

Named streams

A solution designer, working with the queued publish/subscribe programming model, might decide to place all sports publications into a named stream called Sport. In Version 6.0 a stream is often replicated automatically onto other brokers that use the model queue, SYSTEM.BROKER.MODEL.STREAM. However, for the stream to be available to a queue manager that runs on Version 7.0 onwards with queued publish/subscribe enabled, the stream must be added manually.

If you migrate a queue manager from Version 6.0, running the command **strmqbrk** migrates Version 6.0 named streams to topics. The stream Sport is mapped to the topic Sport. This is not applicable to z/OS.

Queued publish/subscribe applications that subscribe to Soccer/Results on stream Sport work without change. Integrated publish/subscribe applications that subscribe to the topic Sport using MQSUB, and supplying the topic string Soccer/Results receive the same publications too.

When the topic Soccer/Result is created by **strmqbrk**, it is defined as a child of the topic Sport, with the topic string Sport. A subscription to Soccer/Results is realized as a subscription to Sport/Soccer/Results, and so publications to the Sport stream are mapped to different place in topic space to publications to a different stream, such as Business .

There are scenarios for which the automatic migration performed by **strmqbrk** is not the answer, and you need to manually add streams. The task of adding a stream is described in the topic Adding a stream. You might need to add streams manually for three reasons.

1. You continue to maintain publish/subscribe applications on version 6 queue managers, which interoperate with newly written publish/subscribe applications running on later queue managers.
2. You continue to develop your queued publish/subscribe applications that are running on later version queue managers, rather than migrate the applications to the integrated publish/subscribe MQI interface.
3. The default mapping of streams to topics leads to a "collision" in topic space, and publications on a stream have the same topic string as publications from elsewhere.

Authorities

By default, at the root of the topic tree there are multiple topic objects: SYSTEM.BASE.TOPIC, SYSTEM.BROKER.DEFAULT.STREAM, and SYSTEM.BROKER.DEFAULT.SUBPOINT . Authorities (for example, for publishing or subscribing) are determined by the authorities on the SYSTEM.BASE.TOPIC; any authorities on SYSTEM.BROKER.DEFAULT.STREAM or SYSTEM.BROKER.DEFAULT.SUBPOINT are ignored. If either of SYSTEM.BROKER.DEFAULT.STREAM or SYSTEM.BROKER.DEFAULT.SUBPOINT are deleted and re-created with a non-empty topic string, authorities defined on those objects are used in the same way as a normal topic object.

Mapping between streams and topics

A queued publish/subscribe stream is mimicked in Version 7.0 onwards by creating a queue, and giving it the same name as the stream. Sometimes the queue is called the stream queue, because that is how

it appears to queued publish/subscribe applications. The queue is identified to the publish/subscribe engine by adding it to the special namelist called SYSTEM.QPUBSUB.QUEUE.NAMELIST. You can add as many streams as you need, by adding additional special queues to the namelist. Finally you need to add topics, with the same names as the streams, and the same topic strings as the stream name, so you can publish and subscribe to the topics.

However, in exceptional circumstances, you can give the topics corresponding to the streams any topic strings you choose when you define the topics. The purpose of the topic string is to give the topic a unique name in the topic space. Typically the stream name serves that purpose perfectly. Sometimes, a stream name and an existing topic name collide. To resolve the problem, can choose another topic string for the topic associated with the stream. Choose any topic string, ensuring it is unique.

The topic string defined in the topic definition is prefixed in the normal way to the topic string provided by publishers and subscribers using the MQOPEN or MQSUB MQI calls. Applications referring to topics using topic objects are not affected by the choice of prefix topic string - which is why you can choose any topic string that keeps the publications unique in the topic space.

The remapping of different streams onto different topics relies on the prefixes used for the topic strings being unique, to separate one set of topics completely from another. You must define a universal topic naming convention that is rigidly adhered to for the mapping to work. In Version 7.0, if topic strings collided you might use streams to separate the topic spaces. From Version 7.0 onwards, you use the prefixing mechanism to remap a topic string to another place in topic space.

Note: When you delete a stream, delete all the subscriptions on the stream first. This action is most important if any of the subscriptions originate from other brokers in the broker hierarchy.

Example

In Figure 18 on page 47, topic 'Sport' has the topic string 'xyz' resulting in publications that originate from stream 'Sport' being prefixed with the string 'xyz' in the version 7 queue manager topic space. Publishing or subscribing in version 7 to the topic 'Sport' prefixes 'xyz' to the topic string. If the publication flows to a version 6 subscriber, the prefix 'xyz' is removed from the publication and it is placed in the 'Sport' stream. Conversely, when a publication flows from version 6 to version 7, from the 'Sport' stream to the 'Sport' topic, the prefix 'xyz' is added to the topic string.

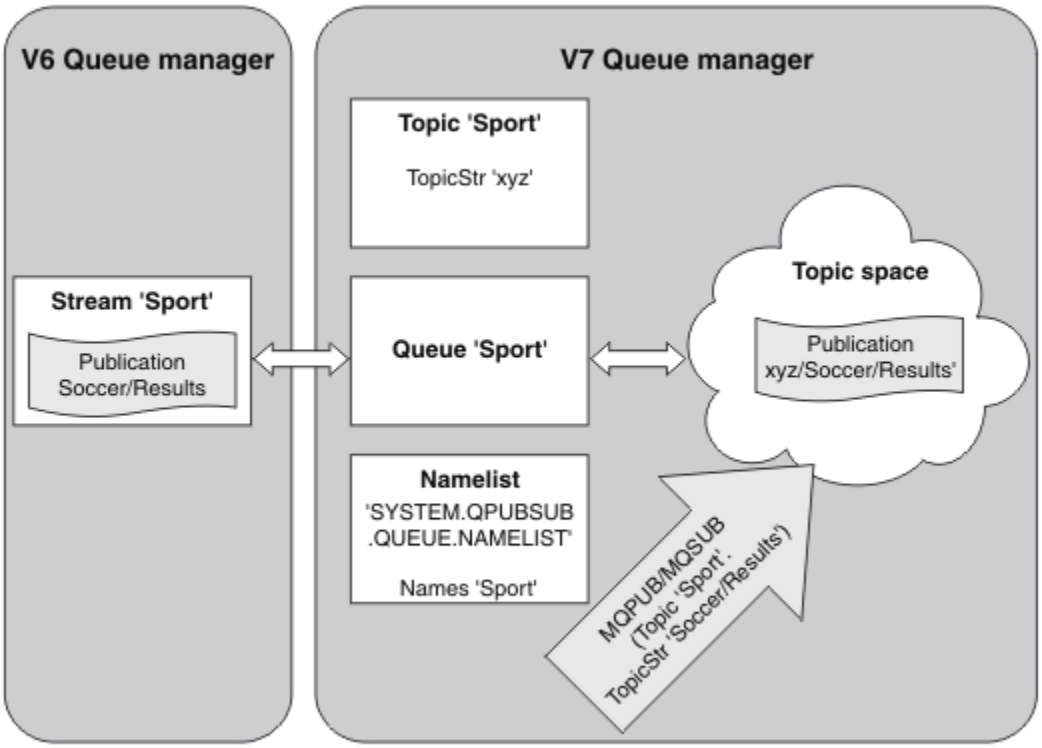


Figure 18. Version 6 streams that coexist with version 7 topics

Subscription points and topics

A subscription point used to request publications from a particular set of publication nodes in WebSphere MQ Event Broker and Message Broker. Named subscription points are emulated by topics and topic objects.

The WebSphere MQ Event Broker V6.0 to WebSphere MQ V7.0.1 migration procedure, **migmbbrk**, converts named subscription points into topics and topic objects. A subscription point is automatically migrated if it has a retained publication, or a registered subscriber. **migmbbrk** creates topic objects from named subscription points. The name of the subscription point becomes the name of the topic object, and the topic string itself. The topic object is added to `SYSTEM.QPUBSUB.SUBPOINT.NAMELIST`.

If a topic object with the same name exists **migmbbrk** does one of two things.

1. If the topic object has a different topic string, or if the subscription point name is longer than an object name, **migmbbrk** creates a topic object with a generated name.
2. If the topic object has the same topic string, **migmbbrk** adds the existing object to the namelist.

To add subscription points manually, see Adding a subscription point.

Subscription points in WebSphere MQ Event Broker

Publication nodes are used in a WebSphere MQ Event and Message Broker message flow to filter and transmit messages to subscribers. Publishers typically do not set subscription points on publication nodes. Subscribers register an interest in a particular set of topics and typically do not specify subscription points either.

A subscription point is a way of selecting which publication nodes forward messages to a subscription. The subscriber qualifies their interest in a set of topics with the name of a subscription point.

Assign a name to the **Subscription point** property of publication node to set its subscription point name.

The subscription point property controls whether a publication to a topic is forwarded to subscribers to the same topic. Publications from publication nodes with a named subscription point are forwarded only to subscribers to the same subscription point. Publications from publication nodes without a named subscription point, the default, are forwarded only to subscribers that have not named a subscription point.

Nodes with a named subscription point send `Publish` command messages in `MQRFH2` format, with the **SubPoint** property set. Subscriptions to a named subscription point must set **SubPoint** property in the `MQRFH2` `Register subscriber` command message.

Subscription points in WebSphere MQ

WebSphere MQ maps subscription points to different topic spaces within the WebSphere MQ topic tree. Topics in command messages without a subscription point are mapped unchanged to the root of the WebSphere MQ topic tree and inherit properties from `SYSTEM.BASE.TOPIC`.

Command messages with a subscription point are processing using the list of topic objects in `SYSTEM.QPUBSUB.SUBPOINT.NAMELIST`. The subscription point name in the command message is matched against the topic string for each of the topic objects in the list. If a match is found, then the subscription point name is prepended, as a topic node, to the topic string. The topic inherits its properties from the associated topic object found in `SYSTEM.QPUBSUB.SUBPOINT.NAMELIST`.

The effect of using subscription points is to create a separate topic space for each subscription point. The topic space is rooted in a topic that has the same name as the subscription point. Topics in each topic space inherit their properties from the topic object with the same name as the subscription point.

Any properties not set in the matching topic object are inherited, in the normal fashion, from `SYSTEM.BASE.TOPIC`.

Existing queued publish/subscribe applications, using `MQRFH2` message headers, continue to work by setting the **SubPoint** property in the `Publish` or `Register subscriber` command messages. The subscription point is combined with the topic string in the command message and the resulting topic processed like any other.

A new WebSphere MQ V7 application is unaffected by subscription points. If it uses a topic that inherits from one of the matching topic objects, it interoperates with a queued application using the matching subscription point.

Example

An existing WebSphere MQ Event Broker publish/subscribe application in a collective uses subscription points to publish share prices in different currencies. The dollar spot price of the IBM stock is published using the subscription point `USD`, and the topic `NYSE/IBM/SPOT`. The sterling price is published using the same topic, and the subscription point `GBP`.

The migration procedure on WebSphere MQ creates two topic objects, `GBP` and `USD`, with the corresponding topic strings `'GBP'` and `'USD'`.

Existing publishers to the topic `NYSE/IBM/SPOT`, migrated to run on WebSphere MQ, that use the subscription point `USD` create publications on the topic `USD/NYSE/IBM/SPOT`. Similarly existing subscribers to `NYSE/IBM/SPOT`, using the subscription point `USD` create subscriptions to `USD/NYSE/IBM/SPOT`.

Subscribe to the dollar spot price in a version 7 publish/subscribe program by calling `MQSUB`. Create a subscription using the `USD` topic object and the topic string `'NYSE/IBM/SPOT'`, as illustrated in the 'C' code fragment.

```
strncpy(sd.ObjectName, "USD", MQ_TOPIC_NAME_LENGTH);
sd.ObjectString.VSPtr = "NYSE/IBM/SPOT";
sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
```

Consider whether your WebSphere MQ Event Broker applications in the collective always used the subscriptions points `USD` and `GBP`. If they did, create the `USD` and `GBP` topic objects only once, as

cluster topics on the cluster topic host. You do not need to carry out step ../com.ibm.mq.mig.doc/q007670_.dita#q007670_/clusterstep of the migration procedure to change SYSTEM.BASE.TOPIC, on every queue manager in the cluster, to a cluster topic. Instead, carry out these steps:

1. Set the CLUSTER attribute of the USD and GBP topic objects on the cluster topic host.
2. Delete all the copies of the USD and GBP topic objects on other queue managers in the cluster.
3. Make sure that USD and GBP are defined in SYSTEM.QPUBSUB.SUBPOINT.NAMELIST on every queue manager in the cluster.

Distributed publish/subscribe

This section contains information about how publish/subscribe messaging can be performed between queue managers, and the two different queue manager topologies that can be used to connect queue managers, clusters and hierarchies.

Queue managers can communicate with other queue managers in your WebSphere MQ publish/subscribe system, so that subscribers can subscribe to one queue manager and receive messages that were initially published to another queue manager. This is illustrated in Figure 19 on page 49.

Figure 19 on page 49 shows a publish/subscribe system with two queue managers.

- Queue manager 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 queue manager, 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 queue manager from the publisher. This is possible because the queue managers are linked to each other.

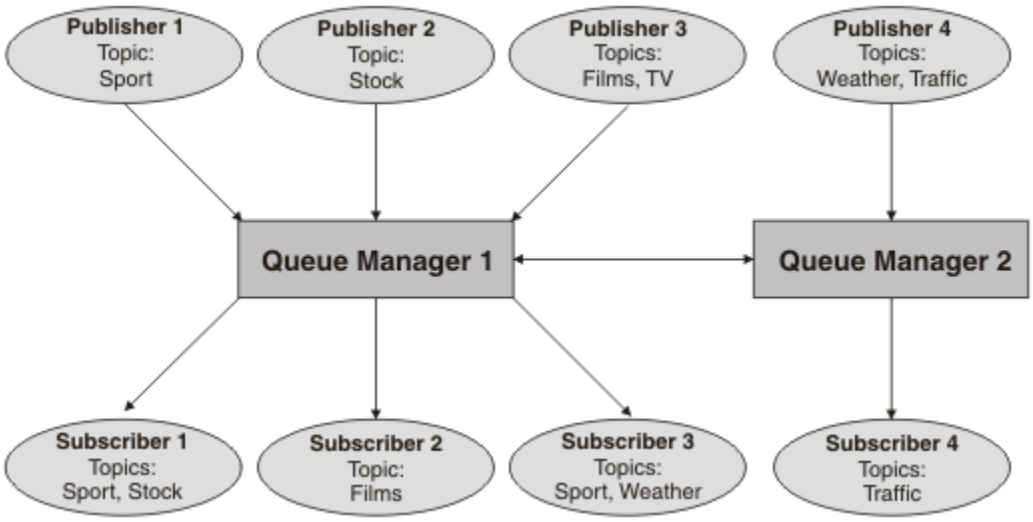


Figure 19. Publish/subscribe example with two queue managers

How does distributed publish/subscribe work?

WebSphere MQ publish/subscribe uses proxy subscriptions to ensure that subscribers can receive messages that are published to remote queue managers.

Distributed publish/subscribe uses the same components as distributed queuing to connect networks of queue managers and consequently, the applications that connect to those queue managers. To find out more about messaging between queue managers and the components involved making connections between queue managers see the *Intercommunication* documentation.

Subscribers need not do anything beyond the standard subscription operation in a distributed publish/subscribe system. When a subscription is made on a queue manager, the queue manager manages the

process by which the subscription is propagated to connected queue managers. Proxy subscriptions flow to all queue managers in the network. They are created to ensure that publications get routed back to the queue manager where the original subscription was created; see Figure 20 on page 50.

A publication is propagated to a remote queue manager only if a subscription to that topic exists on that remote queue manager.

A queue manager consolidates all the subscriptions that are created on it, whether from local applications or from remote queue managers. It creates proxy subscriptions for the topics of the subscriptions with its neighbors, unless a subscription exists; see Figure 21 on page 50.

When an application publishes information, the receiving queue manager forwards it to any applications that have valid subscriptions on remote queue managers. It might forward it through one or more intermediate queue managers; see Figure 22 on page 51.

Subscriber 1 registers a subscription for a particular topic on the Asia queue manager (1). The subscription for this topic is forwarded to all other queue managers in the network (2,3,4).

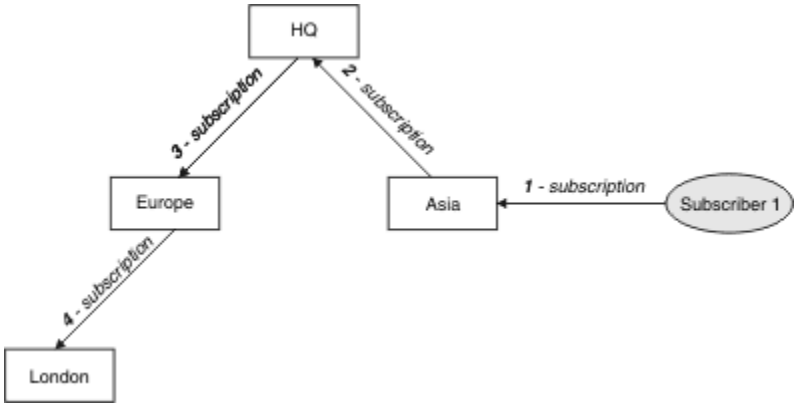


Figure 20. Propagation of subscriptions through a queue manager network

Subscriber 2 registers a subscription, to the same topic as in Figure 20 on page 50, on the HQ queue manager (5). The subscription for this topic is forwarded to the Asia queue manager, so that it is aware that subscriptions exist elsewhere on the network (6). The subscription is not forwarded to the Europe queue manager, because a subscription for this topic has already been registered; see step 3 in Figure 20 on page 50.

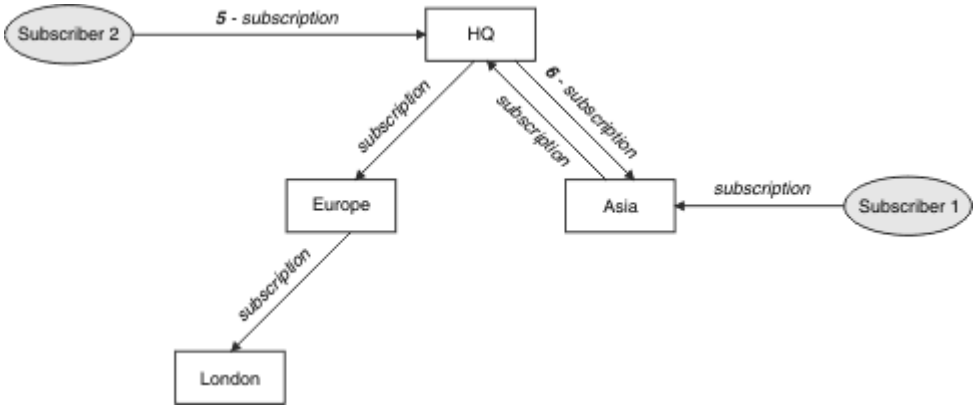


Figure 21. Multiple subscriptions

A publisher sends a publication, on the same topic as in Figure 21 on page 50, to the Europe queue manager (7). A subscription for this topic exists from HQ to Europe, so the publication is forwarded to the HQ queue manager (8). However, no subscription exists from London to Europe (only from Europe to London), so the publication is not forwarded to the London queue manager. The HQ queue manager sends the publication directly to subscriber 2 and to the Asia queue manager (9). The publication is forwarded to subscriber 1 from Asia (10).

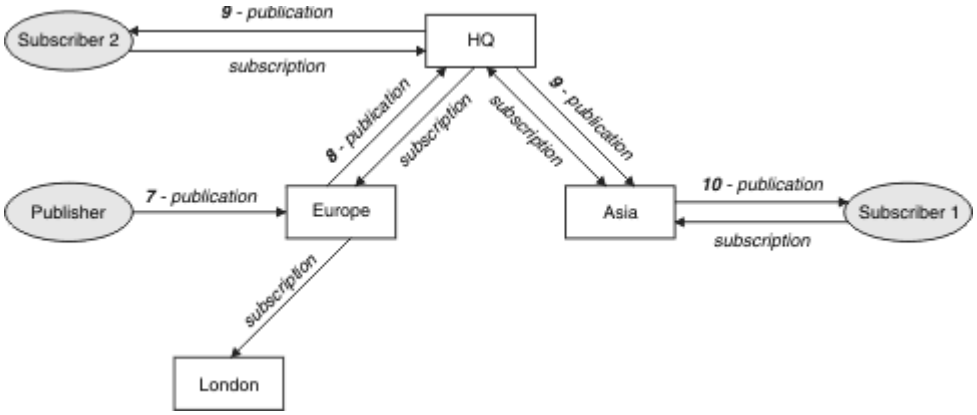


Figure 22. Propagation of publications through a queue manager network

When a queue manager sends any publications or subscriptions to another queue manager, it sets its own user ID in the message. If you are using a publish/subscribe hierarchy, and if the incoming channel is set up to put messages with the authority of the user ID in the message, then you must authorize the user ID of the sending queue manager; see "Using default user IDs with a queue manager hierarchy" on page 104. If you are using publish/subscribe clusters, authorization is handled by the cluster.

Because of the interconnected nature of publish/subscribe queue managers, it takes time for proxy subscriptions to propagate around all nodes in the network. Remote publications do not necessarily start being subscribed to immediately. You can eliminate the subscription delay by using the **Topic** attribute PROXYSUB with the value FORCE as described in "More on routing mechanisms" on page 52.

The subscription operation finishes when the proxy subscription has been put on the appropriate transmission queue for each directly connected queue manager. The subscription operation does not wait for the propagation of the proxy subscription to the rest of the topology.

Proxy subscriptions are associated with the queue manager name that created them. If queue managers in the hierarchy have the same name, it can result in publications failing to reach them. To avoid this problem, as with point-to-point messaging, give queue managers unique names, especially if they are directly or indirectly connected in a WebSphere MQ network.

Within a distributed publish/subscribe network the flow of publications and subscriptions can be controlled, and if appropriate, restricted, using publication and subscription scope.

Proxy subscription aggregation and publication aggregation
Distributed publish/subscribe publications and proxy subscriptions are aggregated to minimize the quantity of messages passing between publish/subscribe queue managers.

A proxy subscription is a subscription made by one queue manager for topics published on another queue manager. You do not create proxy subscriptions explicitly, the queue manager does so on your behalf; see "How does distributed publish/subscribe work?" on page 49.

You can connect queue managers together into a publish/subscribe hierarchy, or into a publish/subscribe cluster. Proxy subscriptions flow between the connected queue managers. Proxy subscriptions cause publications to a topic created by a publisher connected to one queue manager to be received by subscribers to that topic connected to other queue managers; see "Publish/subscribe topologies" on page 61.

A proxy subscription flows between queue managers for each individual topic string that is subscribed to by a subscription.

You can restrict the flow of proxy subscriptions and publications between connected queue managers using the **Topic** attributes PUBSCOPE and SUBSCOPE. You can also restrict the flow of proxy subscriptions containing wildcards by setting the **Topic** attribute WILDCARD to BLOCK; see "Wildcard rules" on page 54.

Proxy subscriptions are flowed between queue managers asynchronously to the creation of subscriptions. You can reduce the latency of waiting for a proxy subscription to be propagated to all the connected queue managers, by setting the **Topic** attribute PROXYSUB to FORCE on the topic, or a parent of a topic that will be subscribed to; see "More on routing mechanisms" on page 52.

Proxy subscription aggregation

Proxy subscriptions are aggregated using a duplicate elimination system. For a particular resolved topic string, a proxy subscription is sent on the first local subscription or received proxy subscription. Subsequent subscriptions to the same topic string make use of this existing proxy subscription.

The proxy subscription is canceled after the last local subscription or received proxy subscription is canceled.

In publish/subscribe topologies with many thousands of subscriptions to individual topic strings, or where the existence of those subscriptions may be rapidly changing, the overhead of the proxy subscription propagation must be considered. The individual proxy subscriptions can be consolidated through the use of the topic attribute **PROXYSUB** being set to FORCE. For more details about routing mechanisms and cluster topic performance, see "More on routing mechanisms" on page 52.

Publication aggregation

When there is more than one subscription to the same topic string on a queue manager, only a single copy of each publication matching that topic string is sent from other queue managers in the publish/subscribe topology. On arrival of the message, the local queue manager delivers a copy of the message to each matching subscription.

It is possible for more than one proxy subscription to match the topic string of a single publication when the proxy subscriptions contain wildcards. If a message is published on a queue manager that matches two or more proxy subscriptions created by a single connected queue manager, only one copy of the publication is forwarded to the remote queue manager to satisfy the multiple proxy subscriptions.

More on routing mechanisms
Publish everywhere is an alternative routing mechanism to individual proxy subscription-forwarding. Individual proxy subscription forwarding means that only publications that have a matching subscription on the topic string are sent to a remote messaging server. Publish everywhere, or broadcast, works by forwarding all publications that are published to a messaging server, to all other messaging servers in a distributed publish/subscribe network. The receiving messaging servers then deliver those publications that match local subscriptions.

Each mechanism has its merits, but also has limitations.

Individual proxy subscription forwarding
This mechanism results in the least amount of inter-queue manager publication traffic as only those publications that match subscriptions on a queue manager are sent.

However:

- Each individual topic string that is subscribed to results in a proxy subscription that is sent to all other queue managers in the publish/subscribe topology. This messaging overhead can be significant if there are many thousands of subscriptions to create or delete (for example, all non-durable subscriptions after a restart of a queue manager) or if the set of subscriptions is rapidly changing, and each is to a different topic string.

- Proxy subscriptions are flowed to other queue managers using asynchronous messaging, therefore, there is a delay between the creation of a subscription and proxy subscription creation, delivery, and processing by the other queue managers. Messages that are published at those queue managers in that interval are not delivered to the remote subscription.

Publish everywhere

With this mechanism:

- There is no per topic string proxy subscription overhead on the system which means rapid subscription creation, deletion, or change does not result in increased network load and processing.
- There is no delay between creating a subscription and publications being flowed to a queue manager as they are always flowed to all queue managers. Therefore, there is no window where publications are not delivered to newly created remote subscriptions.

However:

- All publications are sent to all queue managers in the publish/subscribe topology, potentially resulting in excessive network traffic where publications do not have matching subscriptions on each queue manager.

You might want to use the publish everywhere mechanism when you expect a publication to be subscribed to from a significant proportion of your queue managers in the cluster or hierarchy, or where the proxy subscription overheads are too great because of the frequency of subscription changes. This method of working might be more effective in these instances than in others where you experience increased messaging traffic when publications are sent to all queue managers, rather than to the queue managers with matching subscriptions.

A publish everywhere mechanism can be enabled in IBM WebSphere MQ distributed publish/subscribe topologies by setting the **PROXYSUB** attribute to FORCE for a high-level topic object.

For details about disabling individual proxy subscriptions, see “Disabling individual proxy subscriptions” on page 74.

When this forced proxy subscription is propagated throughout the topology, any new subscriptions immediately receive any publications from other connected queue managers, without suffering latency.

Care must be taken when you are configuring such a system. No topic objects below a topic with **PROXYSUB** set to FORCE must be in a different cluster or hierarchy stream to the node where **PROXYSUB** is set to FORCE. Similarly, lower topic objects must not set their **WILDCARD** attribute to BLOCK. In both cases this can result in published messages not flowing from one queue manager to another correctly.

Even when **PROXYSUB** is set to FORCE, a proxy subscription for each individual topic string that is subscribed to continues to be propagated. If the number and frequency of subscriptions is high enough to cause significant overhead to the system, they can be disabled for all topics on a queue manager. For details about disabling individual proxy subscriptions, see “Disabling individual proxy subscriptions” on page 74.

Multicast and subscription latency

Subscription latency and the PROXYSUB(FORCE) option can be used to maintain a proxy subscription.

For example, there is the potential problem of a proxy subscription from QM_B to QM_A being unmade after all the subscribers are disconnected. This situation might not be wanted if you require the multicast traffic to continue even when the unicast connection to the queue manager terminates. Multicast for WebSphere MQ maintains the proxy subscription for a short time, in case a new subscriber connects, by adding a few minutes latency to each proxy subscription so that they are not unmade the instant that the last subscriber terminates.

You can also use the PROXYSUB(FORCE) option on the topic to ensure that an outstanding proxy subscription is always outstanding. You must ensure that the messages flowing across the queues are required by at least one subscriber for most of the time that the subscription is active. If PROXYSUB(FORCE) is set, a proxy subscription might be sent before the first local subscription or

received proxy subscription, and will not be canceled even after the last local subscription or received proxy subscription is canceled.

If the subscription is still unmade, peer-to-peer communication can be used to ensure that message transfers continue; For more information, see High availability for multicast .

Wildcard rules
Wildcards in proxy subscriptions are converted to use topic wildcards.

If a subscription for a wildcard is received, it can be a character, as used by WebSphere MQ Version 6.0. It can also be a topic, as used by WebSphere Message Broker Version 6.0 and WebSphere MQ Version 7.0.

- Character wildcards use * to represent any character, including /.
- Topic wildcards use # to represent a portion of the topic space between / characters.

In WebSphere MQ Version 7.0, all proxy subscriptions are converted to use topic wildcards. If a character wildcard is found, it is replaced with a # character, back to the nearest /. For example, /aaa/bbb/c*d is converted to /aaa/bbb/#. The conversion results in remote queue managers sending slightly more publications than were explicitly subscribed to. The additional publications are filtered out by the local queue manager, when it delivers the publications to its local subscribers.

Controlling wildcard usage with the WILDCARD property

Use the MQSC **Topic** WILDCARD property or the equivalent PCF Topic `WildcardOperation` property to control the delivery of publications to subscriber applications that use wildcard topic string names. The WILDCARD property can have one of two possible values:

WILDCARD
The behavior of wildcard subscriptions with respect to this topic.

PASSTHRU
Subscriptions made to a wildcarded topic less specific than the topic string at this topic object receive publications made to this topic and to topic strings more specific than this topic.

BLOCK
Subscriptions made to a wildcarded topic less specific than the topic string at this topic object do not receive publications made to this topic or to topic strings more specific than this topic.

The value of this attribute is used when subscriptions are defined. If you alter this attribute, the set of topics covered by existing subscriptions is not affected by the modification. This scenario applies also if the topology is changed when topic objects are created or deleted; the set of topics matching subscriptions created following the modification of the WILDCARD attribute is created using the modified topology. If you want to force the matching set of topics to be re-evaluated for existing subscriptions, you must restart the queue manager.

In the example, “Example: Create the Sport publish/subscribe cluster” on page 42, you can follow the steps to create the topic tree structure shown in Figure 10 on page 38.

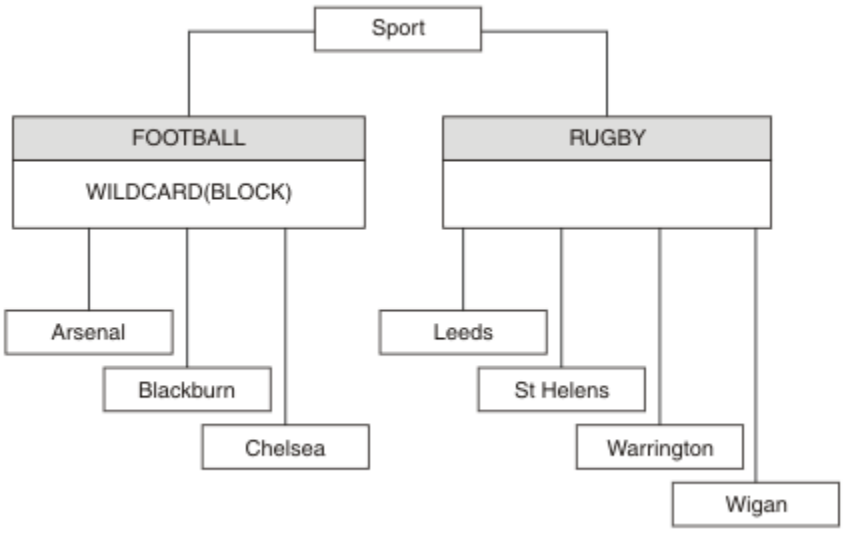


Figure 23. A topic tree that uses the WILDCARD property, BLOCK

A subscriber using the wildcard topic string `#` receives all publications to the `Sport` topic and the `Sport/Rugby` subtree. The subscriber receives no publications to the `Sport/Football` subtree, because the `WILDCARD` property value of the `Sport/Football` topic is `BLOCK`.

PASSTHRU is the default setting. You can set the `WILDCARD` property value PASSTHRU to nodes in the `Sport` tree. If the nodes do not have the `WILDCARD` property value `BLOCK`, setting PASSTHRU does not alter the behavior observed by subscribers to nodes in the `Sports` tree.

In the example, create subscriptions to see how the wildcard setting affects the publications that are delivered; see Figure 14 on page 43. Run the publish command in Figure 17 on page 44 to create some publications.

```
pub QMA
```

Figure 24. Publish to QMA

The results are shown in Table 3 on page 39. Notice how setting the `WILDCARD` property value `BLOCK`, prevents subscriptions with wildcards from receiving publications to topics within the scope of the wildcard.

Table 6. Publications received on QMA

Subscription	**Topic string**	**Publications received**	**Notes**

SPORTS	`Sports/#`	`Sports`
		`Sports/Rugby`
		`Sports/Rugby/Leeds`
			All publications to Football subtree blocked by `WILDCARD(BLOCK)` on `Sports/Football`

SARSENAL	`Sports/#/Arsenal`	-	`WILDCARD(BLOCK)` on `Sports/Football` prevents wildcard subscription on `Arsenal`
SLEEDS	`Sports/#/Leeds`	`Sports/Rugby/Leeds`	Default `WILDCARD` on `Sports/Rugby` does not prevent wildcard subscription on `Leeds`.

Note:

Suppose a subscription has a wildcard that matches a topic object with the WILDCARD property value BLOCK. If the subscription also has a topic string to the right of the matching wildcard, the subscription never receives a publication. The set of publications that are not blocked are publications to topics that are parents of the blocked wildcard. Publications to topics that are children of the topic with the BLOCK property value are blocked by the wildcard. Therefore subscription topic strings that include a topic to the right of the wildcard never receive any publications to match.

Setting the WILDCARD property value to BLOCK does not mean you cannot subscribe using a topic string that includes wildcards. Such a subscription is normal. The subscription has an explicit topic that matches the topic with a topic object having a WILDCARD property value BLOCK. It uses wildcards for topics that are parents or children of the topic with the WILDCARD property value BLOCK. In the example in Figure 10 on page 38, a subscription such as Sports/Football/# can receive publications.

Wildcards and cluster topics

Cluster topic definitions are propagated to every queue manager in a cluster. A subscription to a cluster topic at one queue manager in a cluster results in the queue manager creating proxy subscriptions. A proxy subscription is created at every other queue manager in the cluster. Subscriptions using topics strings containing wildcards, combined with cluster topics, can give hard to predict behavior. The behavior is explained in the following example.

In the cluster set up for the example, “Example: Create the Sport publish/subscribe cluster” on page 42, QMB has the same set of subscriptions as QMA, yet QMB received no publications after the publisher published to QMA, see Figure 11 on page 39. Although the Sports/Football and Sports/Rugby topics are cluster topics, the subscriptions defined in fullsubs.tst do not reference a cluster topic. No proxy subscriptions are propagated from QMB to QMA. Without proxy subscriptions, no publications to QMA are forwarded to QMB .

Some of the subscriptions, such as Sports/#/Leeds , might seem to reference a cluster topic, in this case Sports/Rugby . The Sports/#/Leeds subscription actually resolves to the topic object SYSTEM.BASE.TOPIC.

The rule for resolving the topic object referenced by a subscription such as, Sports/#/Leeds is as follows. Truncate the topic string to the first wildcard. Scan left through the topic string looking for the first topic that has an associated administrative topic object. The topic object might specify a cluster name, or define a local topic object. In the example, Sports/#/Leeds, the topic string after truncation is Sports, which has no topic object, and so Sports/#/Leeds inherits from SYSTEM.BASE.TOPIC, which is a local topic object.

To see how subscribing to clustered topics can change the way wildcard propagation works, run the batch script, upsubs.bat. The script clears the subscription queues, and adds the cluster topic subscriptions in fullsubs.tst. Run puba.bat again to create a batch of publications; see Figure 11 on page 39.

Table 4 on page 40 shows the result of adding two new subscriptions to the same queue manager that the publications were published on. The result is as expected, the new subscriptions receive one publication each, and the numbers of publications received by the other subscriptions are unchanged. The unexpected results occur on the other cluster queue manager; see Table 5 on page 41.

Table 7. Publications received on QMA

Subscription	Topic string	Publications received	Notes
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	All publications to Football subtree blocked by WILDCARD(BLOCK) on Sports/Football
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) on Sports/Football prevents wildcard subscription on Arsenal

Table 7. Publications received on QMA (continued)			
Subscription	**Topic string**	**Publications received**	**Notes**
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	Default WILDCARD on Sports/Rugby does not prevent wildcard subscription on Leeds.
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal receives a publication because the subscription does not have a wildcard.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds would receive a publication in any event.

Table 5 on page 41 shows the results of adding the two new subscriptions on QMB and publishing on QMA. Recall that QMB received no publications without these two new subscriptions. As expected, the two new subscriptions receive publications, because Sports/FootBall and Sports/Rugby are both cluster topics. QMB forwarded proxy subscriptions for Sports/Football/Arsenal and Sports/Rugby/Leeds to QMA, which then sent the publications to QMB.

The unexpected result is that the two subscriptions Sports/# and Sports/#/Leeds that previously received no publications, now receive publications. The reason is that the Sports/Football/Arsenal and Sports/Rugby/Leeds publications forwarded to QMB for the other subscriptions are now available for any subscriber attached to QMB . Consequently the subscriptions to the local topics Sports/# and Sports/#/Leeds receive the Sports/Rugby/Leeds publication. Sports/#/Arsenal continues not to receive a publication, because Sports/Football has its WILDCARD property value set to BLOCK.

Table 8. Publications received on QMB			
Subscription	**Topic string**	**Publications received**	**Notes**
SPORTS	*Sports/#*	*Sports/Rugby/Leeds*	*All publications to Football subtree blocked by WILDCARD(BLOCK) on Sports/Football*
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) on Sports/Football prevents wildcard subscription on Arsenal
SLEEDS	*Sports/#/Leeds*	*Sports/Rugby/Leeds*	*Default WILDCARD on Sports/Rugby does not prevent wildcard subscription on Leeds .*
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal receives a publication because the subscription does not have a wildcard.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds would receive a publication in any event.

In most applications, it is undesirable for one subscription to influence the behavior of another subscription. One important use of the WILDCARD property with the value BLOCK is to make the subscriptions to the same topic string containing wildcards behave uniformly. Whether the subscription is on the same queue manager as the publisher, or a different one, the results of the subscription are the same.

Wildcards and streams

WebSphere MQ Version 6 streams are mapped to topics by WebSphere MQ Version 7; see “Streams and topics” on page 44 . In the default mapping, which is performed by **strmqbrk** in Version 7, all the topics in the stream Sports would be mapped to the topic Sports. All the topics in the stream Business would be mapped to the topic Business.

A subscription in WebSphere MQ Version 6 to * in the Sports stream receives all publications in the Sports tree, and no publications in the Business tree. The same subscription in version 7 would receive all the publications in the Sports tree and all the publications in the Business tree. To block this behavior, when streams are migrated to version 7, **strmqbrk** sets the WILDCARD property. It sets it to the value BLOCK for each of the top-level topics that are migrated from a stream. The WILDCARD property of Sports and Business is set to the value BLOCK by the conversion from the version 6 streams called Sports and Business.

For a new application written to the publish/subscribe API, the effect is that a subscription to * receives no publications. To receive all the Sports publications you must subscribe to Sports/*, or Sports/# , and similarly for Business publications.

The behavior of an existing queued publish/subscribe application does not change when the publish/subscribe broker is migrated to WebSphere MQ Version 7. The **StreamName** property in the **Publish**, **Register Publisher**, or **Subscriber** commands is mapped to the name of the topic the stream has been migrated to.

Wildcards and subscription points

WebSphere Message Broker subscriptions points are mapped to topics by WebSphere MQ Version 7; see “Subscription points and topics” on page 47. In the default mapping, which is performed by **migmqbrk** in Version 7, all the topics in the subscription point Sports would be mapped to the topic Sports. All the topics in the subscription point Business would be mapped to the topic Business.

A subscription on WebSphere Message Broker Version 6 to * in the Sports subscription point receives all publications in the Sports tree, and no publications in the Business tree. The same subscription in version 7 would receive all the publications in the Sports tree and all the publications in the Business tree. To block this behavior, when subscription points are migrated to version 7, **migmqbrk** sets the WILDCARD property. It sets it to the value BLOCK for each of the top-level topics that are migrated from a subscription point. The WILDCARD property of Sports and Business is set to the value BLOCK by the conversion from the WebSphere Message Broker subscription points called Sports and Business .

For a new application written to the publish/subscribe API, the effect of the migration is that a subscription to * receives no publications. To receive all the Sports publications you must subscribe to Sports/*, or Sports/# , and similarly for Business publications.

The behavior of an existing queued publish/subscribe application does not change when the publish/subscribe broker is migrated to WebSphere MQ Version 7. The **SubPoint** property in the **Publish**, **Register Publisher**, or **Subscriber** commands is mapped to the name of the topic the subscription has been migrated to.

Example: Create the Sport publish/subscribe cluster

The steps that follow create a cluster, CL1, with four queue managers: two full repositories, CL1A and CL1B , and two partial repositories, QMA and QMB. The full repositories are used to hold only cluster definitions. QMA is designated the cluster topic host. Durable subscriptions are defined on both QMA and QMB.

Note: The example is coded for Windows. You must recode Create qmgrs.bat and create pub.bat to configure and test the example on other platforms.

1. Create the script files.
   a. Create topics.tst
   b. Create wildsubs.tst

c. Create fullsubs.tst

d. Create qmgrs.bat

e. create pub.bat

2. Run Create qmgrs.bat to create the configuration.

```
qmgrs
```

Create the topics in Figure 10 on page 38. The script in figure 5 creates the cluster topics `Sports/Football` and `Sports/Rugby`.

Note: The REPLACE option does not replace the TOPICSTR properties of a topic. TOPICSTR is a property that is usefully varied in the example to test different topic trees. To change topics, delete the topic first.

```
DELETE TOPIC ('Sports')
DELETE TOPIC ('Football')
DELETE TOPIC ('Arsenal')
DELETE TOPIC ('Blackburn')
DELETE TOPIC ('Chelsea')
DELETE TOPIC ('Rugby')
DELETE TOPIC ('Leeds')
DELETE TOPIC ('Wigan')
DELETE TOPIC ('Warrington')
DELETE TOPIC ('St. Helens')

DEFINE TOPIC ('Sports')     TOPICSTR('Sports')
DEFINE TOPIC ('Football')   TOPICSTR('Sports/Football') CLUSTER(CL1) WILDCARD(BLOCK)
DEFINE TOPIC ('Arsenal')    TOPICSTR('Sports/Football/Arsenal')
DEFINE TOPIC ('Blackburn')  TOPICSTR('Sports/Football/Blackburn')
DEFINE TOPIC ('Chelsea')    TOPICSTR('Sports/Football/Chelsea')
DEFINE TOPIC ('Rugby')      TOPICSTR('Sports/Rugby')    CLUSTER(CL1)
DEFINE TOPIC ('Leeds')      TOPICSTR('Sports/Rugby/Leeds')
DEFINE TOPIC ('Wigan')      TOPICSTR('Sports/Rugby/Wigan')
DEFINE TOPIC ('Warrington') TOPICSTR('Sports/Rugby/Warrington')
DEFINE TOPIC ('St. Helens') TOPICSTR('Sports/Rugby/St. Helens')
```

Figure 25. Delete and create topics: topics.tst

Note: Delete the topics, as REPLACE does not replace topic strings.

Create subscriptions with wildcards. The wildcards corresponding the topics with topic objects in Figure 10 on page 38. Create a queue for each subscription. The queues are cleared and the subscriptions deleted when the script is run or rerun.

Note: The REPLACE option does not replace TOPICOBJ or TOPICSTR properties of a subscription. TOPICOBJ or TOPICSTR are the properties that are usefully varied in the example to test different subscriptions. To change them, delete the subscription first.

```
DEFINE QLOCAL(QSPORTS)   REPLACE
DEFINE QLOCAL(QSARSENAL) REPLACE
DEFINE QLOCAL(QSLEEDS)   REPLACE
CLEAR  QLOCAL(QSPORTS)
CLEAR  QLOCAL(QSARSENAL)
CLEAR  QLOCAL(QSLEEDS)

DELETE SUB   (SPORTS)
DELETE SUB   (SARSENAL)
DELETE SUB   (SLEEDS)
DEFINE SUB   (SPORTS)   TOPICSTR('Sports/#')         DEST(QSPORTS)
DEFINE SUB   (SARSENAL) TOPICSTR('Sports/+/Arsenal') DEST(QSARSENAL)
DEFINE SUB   (SLEEDS)   TOPICSTR('Sports/+/Leeds')   DEST(QSLEEDS)
```

Figure 26. Create wildcard subscriptions: wildsubs.tst

Create subscriptions that reference the cluster topic objects.

Note:

The delimiter, `/`, is automatically inserted between the topic string referenced by TOPICOBJ, and the topic string defined by TOPICSTR.

The definition, `DEFINE SUB(FARSENAL) TOPICSTR('Sports/Football/Arsenal') DEST(QFARSENAL)` creates the same subscription. `TOPICOBJ` is used as a quick way to reference topic string you have already defined. The subscription, when created, no longer refers to the topic object.

```
DEFINE QLOCAL(QFARSENAL) REPLACE
DEFINE QLOCAL(QRLEEDS)   REPLACE
CLEAR  QLOCAL(QFARSENAL)
CLEAR  QLOCAL(QRLEEDS)

DELETE SUB   (FARSENAL)
DELETE SUB   (RLEEDS)
DEFINE SUB   (FARSENAL) TOPICOBJ('Football') TOPICSTR('Arsenal') DEST(QFARSENAL)
DEFINE SUB   (RLEEDS)   TOPICOBJ('Rugby')    TOPICSTR('Leeds')   DEST(QRLEEDS)
```

Figure 27. Delete and create subscriptions: fullsubs.tst

Create a cluster with two repositories. Create two partial repositories for publishing and subscribing. Rerun the script to delete everything and start again. The script also creates the topic hierarchy, and the initial wildcard subscriptions.

Note:

On other platforms, write a similar script, or type all the commands. Using a script makes it quick to delete everything and start again with an identical configuration.

```
@echo off
set port.CL1B=1421
set port.CLIA=1420
for %%A in (CL1A CL1B QMA QMB) do call :createQM %%A
call :configureQM CL1A CL1B %port.CL1B% full
call :configureQM CL1B CL1A %port.CL1A% full
for %%A in (QMA QMB) do call :configureQM %%A CL1A %port.CL1A% partial
for %%A in (topics.tst wildsubs.tst) do runmqsc QMA < %%A
for %%A in (wildsubs.tst) do runmqsc QMB < %%A
goto:eof

:createQM
echo Configure Queue manager %1
endmqm -p %1
for %%B in (dlt crt str) do %%Bmqm %1
goto:eof

:configureQM
if %1==CL1A set p=1420
if %1==CL1B set p=1421
if %1==QMA  set p=1422
if %1==QMB  set p=1423
echo configure %1 on port %p% connected to repository %2 on port %3 as %4 repository
echo DEFINE LISTENER(LST%1) TRPTYPE(TCP) PORT(%p%) CONTROL(QMGR) REPLACE | runmqsc %1
echo START LISTENER(LST%1) | runmqsc %1
if full==%4 echo ALTER  QMGR REPOS(CL1) DEADQ(SYSTEM.DEAD.LETTER.QUEUE) | runmqsc %1
echo DEFINE CHANNEL(TO.%2) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('LOCALHOST(%3)') CLUSTER(CL1)
REPLACE | runmqsc %1
echo DEFINE CHANNEL(TO.%1) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('LOCALHOST(%p%)')
CLUSTER(CL1) REPLACE | runmqsc %1
goto:eof
```

Figure 28. Create queue managers: qmgrs.bat

Update the configuration by adding the subscriptions to the cluster topics.

```
@echo off
for %%A in (QMA QMB) do runmqsc %%A < wildsubs.tst
for %%A in (QMA QMB) do runmqsc %%A < upsubs.tst
```

Figure 29. Update subscriptions: upsubs.bat

Run `pub.bat`, with a queue manager as a parameter, to publish messages containing the publication topic string. `Pub.bat` uses the sample program **amqspub**.

```
@echo off
@rem Provide queue manager name as a parameter
set S=Sports
set S=6 Sports/Football Sports/Football/Arsenal
set S=6 Sports/Rugby Sports/Rugby/Leeds
for %%B in (6) do echo %%B | amqspub %%B %1
```

Figure 30. Publish: pub.bat

Publish/subscribe topologies

A *publish/subscribe topology* consists of queue managers and the connections between them, that support publish/subscribe applications.

A publish/subscribe application can consist of a network of queue managers connected together. The queue managers can all be on the same physical system, or they can be distributed over several physical systems. By connecting queue managers together, publications can be received by an application using any queue manager in the network.

This provides the following benefits:

- Client applications can communicate with a nearby queue manager rather than with a distant queue manager, thereby getting better response times.
- By using more than one queue manager, more subscribers can be supported.

You can arrange queue managers that are doing publish/subscribe messaging in two different ways, clusters and hierarchies. For examples of a simple cluster and a simple hierarchy, see Figure 31 on page 61 and Figure 32 on page 62. For more information about these two topologies and to find out which is most appropriate for you, refer to the information in this section of the product documentation.

It is possible to use both topologies in combination by joining clusters together in a hierarchy.

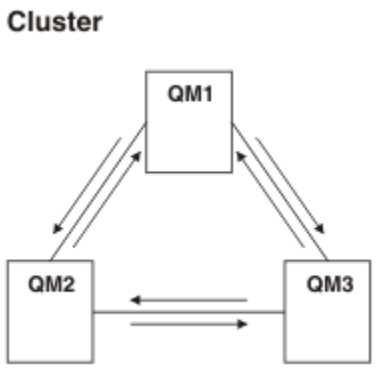


Figure 31. Simple publish/subscribe cluster

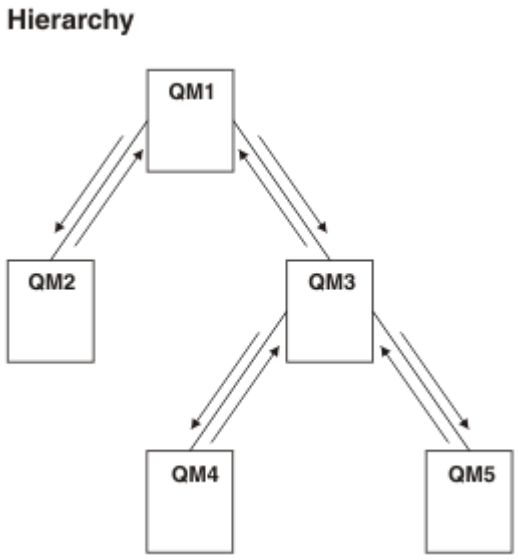
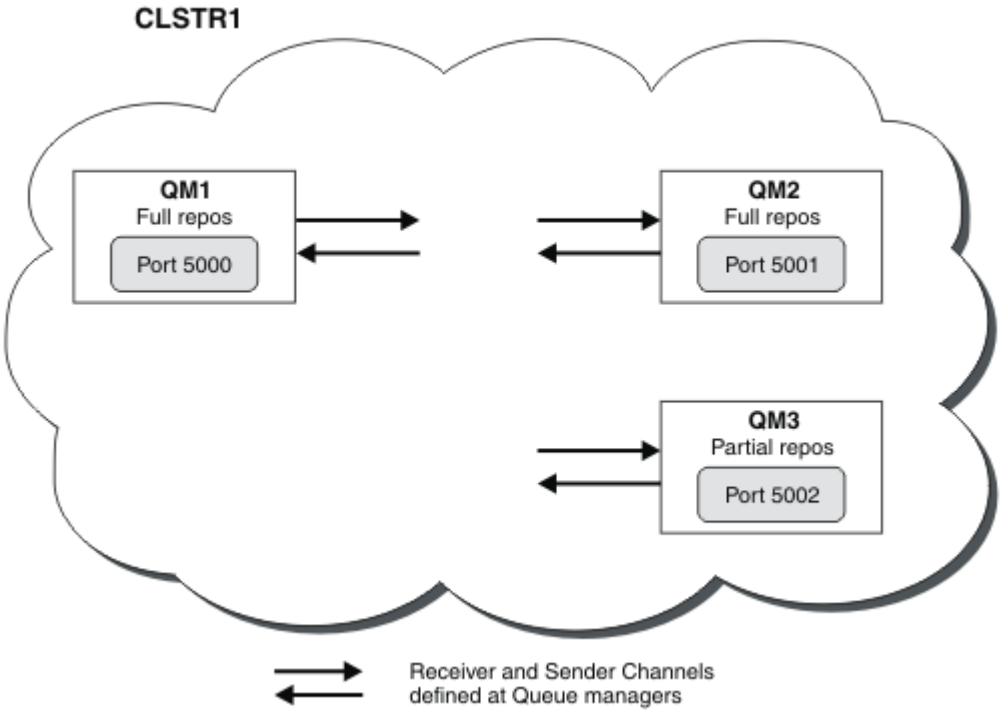


Figure 32. Simple publish/subscribe hierarchy

Setting up a publish/subscribe cluster: Scenario 1
Add two queue managers as full repositories to a cluster, and define the channels between them.

About this task

The following diagram has three queue managers; QM1 , QM2, and



QM3:

QM1 and QM2 are full repositories in a cluster, and QM3 is a partial repository.

Scenario 1 adds QM1 and QM2 to the cluster DEMO as full repositories.

Scenario 2 adds QM3 to the cluster DEMO as a partial repository.

These tasks require at least one command window.

Procedure

1. Set QM1 and QM2 as full repositories of the DEMO cluster:

```
alter QMGR REPOS(DEMO)
```

2. Define and start a listener for QM1:

```
define listener(QM1_LS) TRPTYPE(TCP) CONTROL(QMGR) PORT(5000)
start listener(QM1_LS)
```

3. Define and start a listener for QM2:

```
define listener(QM2_LS) TRPTYPE(TCP) CONTROL(QMGR) PORT(5001)
start listener(QM2_LS)
```

4. Define a receiver channel for QM1:

```
DEFINE CHANNEL(DEMO.QM1) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('$HOSTNAME(5000)')
CLUSTER(DEMO) DESCR('TCP Cluster-receiver channel for queue manager QM1')
```

5. Define a sender channel from QM1 to QM2:

```
DEFINE CHANNEL(DEMO.QM2) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('$HOSTNAME(5001)')
CLUSTER(DEMO) DESCR('TCP Cluster-sender channel from QM1 to  queue manager QM2')
```

6. Define a receiver channel for QM2:

```
DEFINE CHANNEL(DEMO.QM2) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('$HOSTNAME(5001)')
CLUSTER(DEMO) DESCR('TCP Cluster-receiver channel for queue manager QM2')
```

7. Define a sender channel from QM2 to QM1:

```
DEFINE CHANNEL(DEMO.QM1) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('$HOSTNAME(5000)')
CLUSTER(DEMO) DESCR('TCP Cluster-sender channel from qm2 to qm1')
```

8. Define the cluster topic `scores` on QM1:

```
define topic(scores) TOPICSTR(/football) CLUSTER(DEMO)
```

9. Verify the setup with the following commands:

```
display topic(scores) type(all) clusinfo
display clusqmgr(*)
display chstatus(*)
```

10. Test the setup using two command windows:
    a. Enter this command in the first command window:

```
/opt/mqm/samp/bin/amqspub /FOOTBALL/scores QM1
```

    b. Enter this command in the second command window:

```
/opt/mqm/samp/bin/amqssub /FOOTBALL/scores QM2
```

Related tasks

Managing WebSphere MQ clusters

Setting up a new cluster

Setting up a publish/subscribe cluster: Scenario 2

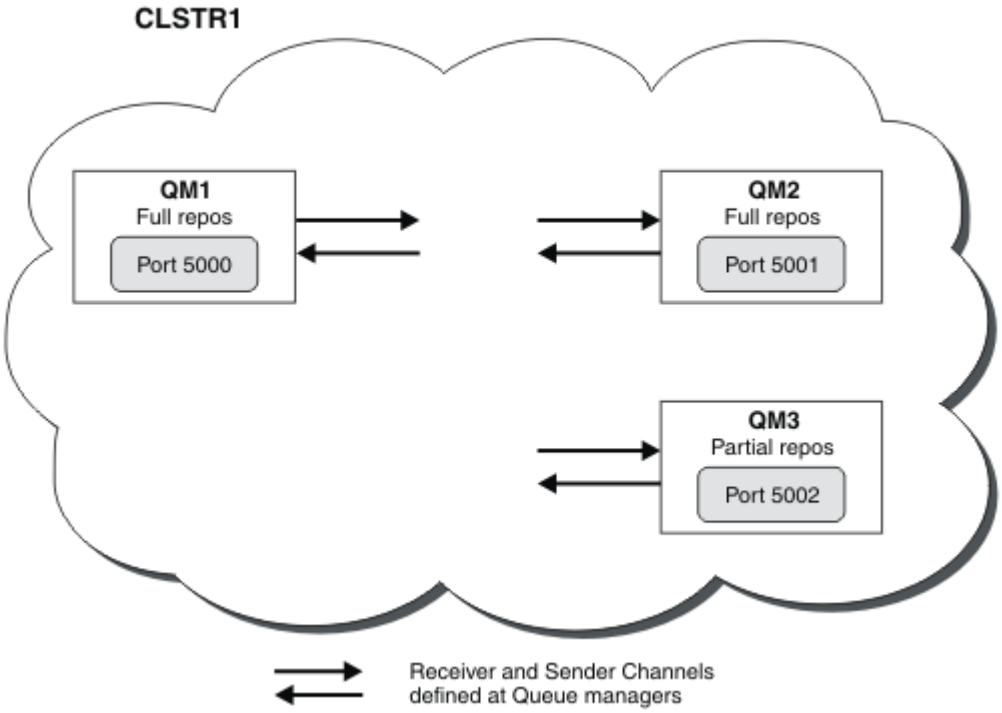
Add a third queue manager as a partial repository to the cluster.

Before you begin

You must have completed the task in “Setting up a publish/subscribe cluster: Scenario 1” on page 62 before completing this task.

About this task

The following diagram has 3 queue managers; QM1, QM2, and



QM3:

QM1 and QM2 are full repositories in a cluster, and QM3 is a partial repository.

Scenario 1 adds QM1 and QM2 to the cluster DEMO as full repositories.

Scenario 2 adds QM3 to the cluster DEMO as a partial repository.

These tasks require at least 1 command window.

Procedure

1. Define and start a listener for QM3:

```
define listener(QM3_LS) TRPTYPE(TCP) CONTROL(QMGR) PORT(5002)
start listener(QM3_LS)
```

2. Define a receiver channel for QM3:

```
DEFINE CHANNEL(DEMO.QM3) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('$HOSTNAME(5002)') CLUSTER
(DEMO) DESCR('TCP Cluster-receiver channel for queue manager QM3')
```

3. Define a sender channel from QM3 to QM1:

```
DEFINE CHANNEL(DEMO.QM1) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('$HOSTNAME(5000)')
CLUSTER(DEMO) DESCR('TCP Cluster-sender channel from qm3 to qm1')
```

4. Verify the setup with the following commands:

```
display topic(scores) type(all) clusinfo
display clusqmgr(*)
display chstatus(*)
```

5. Test the setup using 2 command windows:
   a. Enter this command in the first command window:

```
/opt/mqm/samp/bin/amqspub /FOOTBALL/scores QM2
```

   b. Enter this command in the second command window:

```
/opt/mqm/samp/bin/amqssub /FOOTBALL/scores QM3
```

Publish/subscribe clusters
A publish/subscribe cluster is a standard IBM WebSphere MQ cluster of interconnected queue managers where publications are automatically moved from publishing applications to subscriptions that exist on any of the queue managers in the cluster.

A cluster that is used for publish/subscribe messaging is no different from a standard IBM WebSphere MQ cluster. As such, the queue managers within the publish/subscribe cluster can exist on physically separate computers and each pair of queue managers is automatically connected together by cluster channels when necessary. For information about how to plan and configure an IBM WebSphere MQ cluster, see How clusters work.

A publish/subscribe cluster is created when a clustered topic object is defined, through setting the **CLUSTER** attribute on the topic configured by any queue manager in the cluster. The topic definition is propagated to all members of the cluster. You can publish and subscribe to the topic, and any topic string below it in the topic tree, on any queue manager in the cluster. Publications are automatically propagated to subscribers connected to other queue managers in the cluster.

Non-clustered publish/subscribe activity can also take place in a publish/subscribe cluster, by working with topic strings which do not fall under a clustered topic object. This arrangement differs from a publish/subscribe hierarchy, in which all subscriptions are propagated across the whole hierarchy. In both cases, finer control is available using Subscription and Publication scope.

Using clusters in a publish/subscribe topology provides the following benefits:

- Messages destined for a subscription on a specific queue manager in the same cluster are transported directly to that queue manager and do not need to pass through an intermediate queue manager. This improves performance and optimizes inter-queue manager publish/subscribe traffic, in comparison with a hierarchical topology.
- Because all queue managers are directly connected to each other, there is no single point of failure in this topology. If one queue manager is not available, subscriptions on other queue managers in the cluster are still able to receive messages from publishers on available queue managers.
- In a system that contains multiple separate clusters, for example where the clusters are geographically dispersed, it is possible to connect clusters into a hierarchy of clusters. This connection is created by joining a single queue manager in each cluster to enable the flow of publications and subscriptions through the network; see “Combine the topic spaces of multiple clusters” on page 96. You can also control which publications flow from one cluster to another; see “Combine and isolate topic spaces in multiple clusters” on page 97 .
- A subscribing application can connect to its nearest queue manager, to improve its own performance. The queue manager receives all messages that match the subscription registration of the client from all queue managers within the cluster.

  The performance of a client application is also improved for other services that are requested from this queue manager. A client application can use both publish/subscribe and point-to-point messaging.
- The number of clients and subscriptions for each queue manager can be reduced by adding more queue managers to the cluster to share workload. Publications automatically distribute to the clients on the new queue managers. For some usage patterns, this process can make a publish/subscribe cluster topology highly scalable.

Things to consider when using clusters in publish/subscribe:

- All queue managers in a publish/subscribe cluster are automatically made aware of all other queue managers in the cluster. This process is different for a point-to-point cluster, where only queue managers of interest to a queue manager are known of.
- Queue managers in a publish/subscribe cluster that host one or more subscriptions to a clustered topic, automatically create cluster sender channels to all other queue managers in the cluster. The queue managers also send information regarding the subscriptions to each of them, even when the receiving queue managers are not publishing messages on any clustered topics.

- The first subscription on a queue manager to a topic string under a clustered topic results in a message being sent to every other queue manager in the cluster. Similarly, the last subscription on a topic string to be deleted also results in a message. The more individual topic strings being used under a clustered topic, the more inter-queue manager communication occurs.

CAUTION:

For the reasons listed previously in this topic, the introduction of a clustered topic into a large IBM WebSphere MQ cluster (that is, a cluster that contains many queue managers) can immediately result in additional load on each queue manager in the cluster, and in some situations cause a reduction in performance. For more information, see “Cluster topic performance” on page 72.

Introducing publish/subscribe into a cluster of queue managers, especially an existing cluster, must be carefully planned to accommodate these reductions in performance.

Where it is known that a cluster cannot accommodate the reduction in performance of publish/subscribe, it is possible to disable the clustered publish/subscribe functionality in queue managers using the **PSCLUS** parameter. The **PSCLUS** parameter is primarily to stop the severe problems that can occur with the creation of a publish/subscribe cluster by accidentally or incorrectly defining a clustered topic. For more information about disabling this functionality, see “Inhibiting clustered publish/subscribe in a cluster” on page 71.

Publish/subscribe clustering: Best practices
This topic provides guidance for planning and administering IBM WebSphere MQ Publish/subscribe clusters. The information is based on testing and feedback from customers.

The following information assumes that the user has a basic understanding of IBM WebSphere MQ clusters, Publish/subscribe, and is familiar with the topics in “Distributed publish/subscribe” on page 49. This information is not intended as a "one size fits all" solution, but is instead trying to share common approaches to common problems.

Publish/Subscribe clusters

With a cluster, you have "any-to-any" direct connectivity between queue managers in the cluster when required. When a cluster is used for point-to-point messaging, each queue manager in the cluster knows only the information about other cluster resources, such as other queue managers in the cluster and clustered queues, when applications connecting to them request to use them; that is they work on a need-to-know basis.

A publish/subscribe cluster is a cluster of queue managers, with the usual CLUSSDR and CLUSRCVR channel definitions. However, a publish/subscribe cluster also contains at least one TOPIC object that is defined on at least one queue manager in the cluster where the topic object has identified a cluster name.

With a topic object defined in the cluster, an application that is connected to one queue manager in the cluster can subscribe to that topic, or any node in the topic tree below that topic, and receive publications on that topic from other queue managers in the cluster. This process is achieved by the creation of proxy subscriptions on all the other queue managers in the cluster identifying the queue manager where the subscription exists. So, when a publication to the topic in question happens on their queue manager, they know to forward it to other appropriate members of the cluster, and from there deliver it to the individual application subscriptions.

To achieve this delivery, every queue manager in the cluster needs to know the identity of every other queue manager in the cluster, as soon as a topic is added to a cluster. This knowledge is propagated by way of the cluster's full repository queue managers. Published messages on one queue manager are only sent to other queue managers in the cluster that are known to host subscriptions to the same topic. To achieve this process, when an application creates a subscription to a topic that is clustered, that queue manager must communicate directly with every other queue manager in the cluster, by way of cluster-sender channels to propagate the proxy subscriptions.

This process differs greatly from the limited need-to-know information and connectivity that is needed when you are using a cluster for point-to-point delivery. Therefore, the requirements on a publish/

subscribe cluster are different to the requirements on a point-to-point cluster (one without any clustered topics).

Using clustered topics makes extending the publish/subscribe domain between queue managers simple, but can lead to problems if the mechanics and implications are not understood, and considered in respect to the cluster being used for publish/subscribe. The following best practices are designed to help in this understanding and preparation.

In summary, the performance implications of clustered publish/subscribe can be detrimental to a large cluster, and needs to be carefully considered and understood before any attempt to use publish/subscribe in an existing cluster. For example, even the simple creation of a clustered topic object. It might be better to start with a small new cluster dedicated to publish/subscribe activity, and grow the cluster from there.

Designing a publish/subscribe topology

As previously described, there are capacity and performance considerations when you are using publish/subscribe in a cluster. Therefore, it is best practice to carefully consider the need for publish/subscribe across queue managers, and to limit it to only the number of queue managers that require it. After the minimal set of queue managers that need to publish and subscribe to a set of topics are identified, they can be made members of a cluster which contains only them and no other queue managers.

This is especially true in an established cluster already functioning well for point-to-point messaging. For this reason, when you are turning an existing large cluster into a publish/subscribe cluster, it is a better practice to initially create a separate cluster for the publish/subscribe work where the applications can be tried, rather than using the current cluster. It is possible to continue to use existing queue managers already in one or more point-to-point clusters, the subset of these queue managers need to be made members of the new publish/subscribe cluster. However, this new cluster must have separate queue managers that are configured as full repositories to isolate the additional load from the existing cluster full repositories.

Where you establish that a cluster is not to be used for publish/subscribe due to its size or current load, it is good practice to prevent this cluster unexpectedly being made into a publish/subscribe cluster by the simple creation of a clustered topic on any queue manager in the cluster. Use the **PSCLUS** queue manager property to achieve this design, for details, see Inhibiting clustered publish/subscribe in a cluster.

It is also important to choose carefully which topics are to be added to the cluster: The higher up the topic tree these topics are, the more widespread they become. For this reason, it is not recommended to put the topic root node into the cluster without considering the behavior that is seen. Make global topics obvious where possible, for example by using a high-level qualifier in the topic string: `/global` or `/cluster`.

How to size systems

Publish/subscribe clusters require many channels because the model is different to point-to-point messaging: There is the need for each queue manager to talk to all other queue managers in that cluster. The point-to-point model is an 'opt in' one, but publish/subscribe clusters have an indiscriminate nature with subscription fan-out. Therefore, the full repository queue managers, and any queue manager that hosts local subscriptions in a publish/subscribe cluster, must have the capacity to establish channels to every member of the cluster at the same time.

It is best to ensure that every queue manager in the publish/subscribe cluster can achieve this capacity, but it is acknowledged that queue managers that are known never to host subscriptions do not need to establish channels with every other queue manager, and therefore do not require this level of capacity.

However, care must be taken because an accidental subscription created on such a queue manager, or any attempt to manually resynchronize such a queue manager with the others in the cluster, results in all channels being started concurrently. See “Resynchronization of proxy subscriptions” on page 68 for more information.

Clustered publish/subscribe enables the delivery of published messages on one queue manager to be delivered to subscriptions on other queue managers. But as for point-to-point messaging, the cost of transmitting messages between queue managers can be detrimental to performance. Therefore, attempts

must be made to wherever possible to create subscriptions to topics on the same queue managers as where messages are being published.

Another consideration is the affect on performance on the system of propagating proxy subscriptions. Typically, a queue manager sends a proxy subscription message to every other queue manager in the cluster when the first subscription for a specific clustered topic string (not just a configured topic object) is created. If a publish/subscribe solution consists of many unique topic strings being subscribed to, or the topics are frequently being subscribed and unsubscribed from, a significant amount of proxy subscription traffic can be generated between all queue managers in a cluster, adversely affecting the overall performance of the system. See “Cluster topic performance” on page 72 for information about ways to reduce proxy subscription overhead.

Resynchronization of proxy subscriptions

Under normal circumstances, queue managers automatically ensure that the proxy subscriptions in the system correctly reflect the subscriptions on each queue manager in the cluster.

However, should the need arise, you can manually resynchronize a queue manager's local subscriptions with the proxy subscriptions that it propagated across the cluster using the REFRESH QMGR TYPE(PROXYSUB) command.

Note: Resynchronization temporarily creates a sudden additional proxy subscription load on the cluster, originating from the queue manager where the command is issued. For this reason, do not use it unless IBM WebSphere MQ service, IBM WebSphere MQ documentation, or error logging instructs you to do so.

An example of when resynchronization is required is when a queue manager cannot correctly propagate its proxy subscriptions, perhaps because a channel has stopped and all messages cannot be queued for transmission, or because operator error has caused messages to be incorrectly deleted from the `SYSTEM.CLUSTER.TRANSMIT.QUEUE` queue. In this situation, first rectify the original problem (for example by restarting the channel) then issue the **`REFRESH QMGR TYPE(PROXYSUB)`** command on the queue manager. Note that publications missed due to proxy subscriptions not being in place are not recovered for the affected subscriptions. This drawback must be taken into account.

Resynchronization requires the queue manager to start channels to all other queue managers in the cluster. Therefore the queue manager that you are refreshing must have enough capability to cope with communicating with every other queue manager in the cluster.

Cluster topics
Cluster topics are administrative topics with the **`cluster`** attribute defined. Information about cluster topics is pushed to all members of a cluster and combined with local topics to create a different topic space at each queue manager.

When you define a cluster topic on a queue manager, the cluster topic definition is sent to the full repository queue managers. The full repositories then propagate the cluster topic definition to all queue managers within the cluster, making the same cluster topic available to publishers and subscribers at any queue manager in the cluster. The queue manager on which you create a cluster topic is known as a cluster topic host. The cluster topic can be used by any queue manager in the cluster, but any modifications to a cluster topic must be made on the queue manager where that topic is defined (the host) at which point the modification is propagated to all members of the cluster via the full repositories.

At each queue manager a single topic name space is constructed from the local and cluster topic definitions that it is aware of. When an application subscribes to a topic that resolves to a clustered topic, IBM WebSphere MQ creates a proxy subscription and sends it directly from the queue manager where the subscription is created, to all the other members of the cluster. Unlike the clustered topic itself, proxy subscriptions do not flow via the full repository queue managers.

Messages published on a topic are sent to every subscription known to the queue manager that the publisher is connected to. If any of those subscriptions are proxy subscriptions, a copy of the published message is sent to the queue manager that originated the proxy subscription. The receiving queue manager then sends a copy of the message to every local subscription. This process ensures that the subscriber to a clustered topic receives publications from publishers connected to any of the queue

managers in the cluster and that the minimal number of published messages are propagated through the cluster.

If you have a clustered topic, and a local topic object, then the local topic takes precedence. See "Multiple cluster topic definitions" on page 70 for more information.

For more information about the commands to use to display cluster topics, see the following related links:

Wildcard subscriptions

Proxy subscriptions are created when local subscriptions are made to a topic string that resolves at, or below, a clustered topic object. If a wildcard subscription is made higher in the topic hierarchy than any cluster topic, it does not have proxy subscriptions sent around the cluster for the matching cluster topic, and therefore receives no publications from other members of the cluster. It does however receive publications from the local queue manager.

However, if another application subscribes to a topic string that resolves to or below the cluster topic, proxy subscriptions are generated and publications are propagated to this queue manager. On arrival the original, higher wildcard subscription is considered a legitimate recipient of those publications and receives a copy.

This behavior differs from locally published messages on the same topics. If this behavior is not required, setting **WILDCARD(BLOCK)** on the clustered topic makes the original wildcard not be considered a legitimate subscription and does not receive any publications (local, or from elsewhere in the cluster) on the cluster topic, or its subtopics.

Related concepts

Working with administrative topics

Working with subscriptions

Related reference

DISPLAY TOPIC

DISPLAY TPSTATUS

DISPLAY SUB

Cluster topic attributes

A good understanding of cluster topic attributes is needed to design and administer publish/subscribe clusters.

A topic object has a number of attributes that apply to multi-queue manager publish/subscribe topologies. When you are using an IBM WebSphere MQ cluster to create such a topology, these attributes have the following behavior.

PROXYSUB

- **PROXYSUB** is an attribute that controls when proxy subscriptions are made. For details about why you might want to change this attribute from the default value of FIRSTUSE, see "More on routing mechanisms" on page 52.
- In the same way as for other attributes of a clustered topic, the **PROXYSUB** attribute is propagated to every queue manager in the cluster, not just the queue manager that the topic was defined on. This instantly results in every queue manager in the cluster creating a wildcarded proxy subscription to every other queue manager. The result of this process is that every queue manager creates cluster sender channels to every other queue manager, and any messages that are published are sent to every queue manager.

PUBSCOBE and SUBSCOPE

PUBSCOPE and **SUBSCOPE** determine whether this queue manager propagates publications to queue managers in the topology (publish/subscribe cluster or hierarchy) or restricts the scope to just its local queue manager. You can do the equivalent job programmatically using MQPMO_SCOPE_QMGR / MQSO_SCOPE_QMGR.

- **PUBSCOPE** If a cluster topic object is defined with **PUBSCOPE(QMGR)**, the definition is shared with the cluster, but the scope of publications that are based on that topic is local only and they are not sent to other queue managers in the cluster.
- **SUBSCOPE** If a cluster topic object is defined with **SUBSCOPE(QMGR)**, the definition is shared with the cluster, but the scope of subscriptions that are based on that topic is local only, therefore no proxy subscriptions are sent to other queue managers in the cluster.

These two attributes are commonly used together to isolate a queue manager from interacting with other members of the cluster on particular topics. The queue manager neither publishes or receives publications on those topics to and from other members of the cluster. This situation does not prevent publication or subscription if topic objects are defined on subtopics.

Setting **SUBSCOPE** to QMGR on a local definition of a topic does not prevent other queue managers in the cluster from propagating their proxy subscriptions to the queue manager if they are using a clustered version of the topic, with **SUBSCOPE(ALL)**. However, if the local definition also sets **PUBSCOPE** to QMGR those proxy subscriptions are not sent publications from this queue manager.

Multiple cluster topic definitions
A local topic definition overrides a remotely defined cluster topic definition of the same name. Creation of multiple definitions of the same cluster topic on different queue managers in a cluster is also possible. Both of these scenarios require some caution however, the reasons are explained in this topic.

Just as for clustered queues, having multiple definitions of the same cluster topic object in a cluster introduces the possibility of different properties defined on each. It is not easy to determine which version of the topic definition is seen by each queue manager in the cluster and it is therefore hard to determine the expected behavior.

Where two or more cluster topic definitions, for a single topic string, have differing attributes or exist in more than one cluster, messages (AMQ5465 & AMQ5466) are written to the error log and the most recently received cluster topic definition is used.

The cluster topic host queue manager must not delete the topic definition, and remains in the cluster to ensure that the clustered topic continues to be known by all members of the cluster. It is not essential that this host queue manager is continually available because the cluster topic definition is cached by the full repository queue managers and by all other queue managers in their partial cluster repositories. This caching allows for at least 60 days of availability while the host queue manager is unavailable. For more information about this subject, see “Key roles for publish/subscribe cluster queue managers” on page 76.

Overriding a cluster topic definition locally

It might be necessary to override the behavior of a clustered topic on certain queue managers in the cluster. This override can be achieved by defining a local topic object to override a cluster topic object with the same topic string, and use it to publish only to locally connected subscribers.

Even when a local definition of a topic is created to override a clustered topic on a queue manager the queue manager continue to receive proxy subscriptions from other members of the cluster using the clustered topic definition. By default, messages published locally continue to be sent to the remote queue managers to honor the proxy subscriptions. If this arrangement is not required, specify **PUBSCOPE**(QMGR) on the local topic object to ensure publisher applications connected to this queue manager publish only to local subscribers.

Modifying a cluster topic definition

If you need to alter a cluster topic definition, modify it at the same queue manager it was defined on, the cluster topic host. Do not create a definition of the same cluster topic on a different queue manager in the cluster. Defining the topic again results in two cluster topic hosts for the same cluster topic.

Defining a cluster topic multiple times creates potentially conflicting definitions and the possibility that different queue managers use different definitions at different times.

Moving a cluster topic definition to a different queue manager in the cluster

You might need to move a cluster topic definition from one queue manager in the cluster to another, for example when decommissioning a queue manager from the cluster. To move a cluster topic definition to a different queue manager in the cluster without interrupting the flow of publications, you need to follow these steps. The example moves a definition from QM1 to QM2.

1. Create a duplicate of the cluster topic definition on QM2 with the same attributes as the definition of QM1.
2. Wait for the new definition to be propagated throughout the cluster by the full repository queue managers. The propagation can be determined by displaying the cluster topics on each cluster member using the **(DISPLAY CLUSTER)** command, and checking for a definition originating from QM2.
3. Delete the cluster topic definition from QM1.

After the original definition is deleted from QM1 it is possible to modify the definition on QM2 if required, without introducing a conflict in properties.

Replacing a cluster topic definition on a failed queue manager

In the previous scenario, it might not be possible to delete the definition from QM1 if QM1 is not available for a period of time. In this scenario it is acceptable to run with both definitions in existence.

If it then becomes a requirement to modify the clustered topic definition, it is possible to modify the version on QM2 in the knowledge that the QM2 definition is newer than the QM1 definition, and therefore prevails. However, during this period, errors are written to the error logs of the queue managers because there is a conflicting cluster topic definition. Resolve the error as soon as possible by removing the duplicate cluster topic definition from QM1 when it can be restarted.

Alternatively, if QM1 is never going to return to the cluster (for example, unexpected decommissioning following a catastrophic hardware failure), the RESET CLUSTER command can be used to forcibly eject the queue manager. **RESET CLUSTER** automatically deletes all topic objects hosted on the target queue manager.

Inhibiting clustered publish/subscribe in a cluster
Introducing publish/subscribe into a cluster of queue managers, especially an existing cluster, must be carefully planned to accommodate any reductions in performance.

The introduction of a clustered topic into a large IBM WebSphere MQ cluster (one that contains many queue managers) can immediately result in additional load on each queue manager in the cluster, and in some situations, a reduction in performance. Therefore the introduction of publish/subscribe must be carefully planned. See “Cluster topic performance” on page 72 for more information.

Where it is known that a cluster could not accommodate the overheads of publish/subscribe it is possible to disable the clustered publish/subscribe functionality in queue managers by setting the queue manager attribute **PSCLUS** to DISABLED.

Setting **PSCLUS** to DISABLED modifies three aspects of queue manager functionality:

- An administrator of this queue manager is no longer able to define a Topic object as clustered.
- Incoming topic definitions or proxy subscriptions from other queue managers are rejected (a warning message is logged to inform the administrator of incorrect configuration).
- Full repositories no longer share information about every queue manager with all other partial repositories automatically when they receive a topic definition.

Although **PSCLUS** is a parameter of each individual queue manager in a cluster, it is not intended to selectively disable publish/subscribe in a subset of queue managers in the cluster. Apart from anything else, this method would cause frequent error messages to be seen as proxy subscriptions and topic definitions would constantly be seen and rejected. Ideally, when using this option, consistently set all queue managers in the cluster to disabled. Where a queue manager participates in one or more publish

subscribe cluster or clusters, and also one or more traditional cluster or clusters, **PSCLUS** must be set to ENABLED on that queue manager. See the following information about disabling at the full repositories.

Importantly, setting **PSCLUS** to DISABLED on all full repository queue managers in the cluster prevents any clustered topic definition on an incorrectly configured partial repository from affecting other queue managers in the cluster. In such scenarios, the inconsistency is reported in the error logs of the full repository queue managers.

When overlapping a traditional point to point cluster with a publish subscribe cluster, it is important to use a separate set of full repositories in each. This arrangement allows topic definitions and 'all queue manager' information to be allowed to flow only in the publish/subscribe cluster.

There are some caveats on usage of this parameter which help to avoid inconsistent configurations. No clustered topic objects can exist in any cluster of which this queue manager is a member when modifying from ENABLED to DISABLED. Any such topics (even remotely defined ones) must be deleted before disabling this function

For more information about **PSCLUS**, see ALTER QMGR (PSCLUS).

Cluster topic performance
The performance characteristics of cluster topics require special consideration because they differ from the performance characteristics of cluster queues, and poorly considered usage can potentially be a source of performance problems in large or unbalanced clusters.

Reducing the effect of publish/subscribe on performance

There are two sources of workload on a queue manager in a cluster: directly handling messages for application programs, and handling messages and channels needed to manage the cluster. In a typical point-to-point cluster, the cluster system workload is largely limited to information explicitly requested by members of the cluster as required (see a comparison in “Performance characteristics of publish/subscribe clusters” on page 75). Therefore in anything other than a very large cluster, for example one which contains thousands of queue managers, you can largely discount the performance effect of managing the cluster when considering queue manager performance.

In a publish/subscribe cluster, information, such as clustered topics and proxy subscriptions, is pushed to all members of a cluster, irrespective of whether all cluster queue managers are actively participating in publish/subscribe messaging. This process can create a significant additional load on the system. Therefore you do need to consider the effect of cluster management on queue manager performance, both in its timing, and its size.

To reduce the effect of publish/subscribe cluster management on the performance of a cluster consider the following two suggestions:

1. Perform cluster, topic, and subscription updates at off-peak times of the day.
2. If you are considering adding publish/subscribe topics to an existing large cluster just because the cluster is already there, consider whether you can define a much smaller subset of queue managers involved in publish/subscribe and make that an "overlapping" cluster. This cluster is then the cluster where cluster topics are defined. Although some queue managers are now in two clusters, the overall effect of publish/subscribe is reduced:
   a. The size of the publish/subscribe cluster is smaller.
   b. Queue managers not in the publish/subscribe cluster are much less affected by the effect of cluster management traffic.

Balancing producers and consumers

An important concept in asynchronous messaging performance is *balance*. Unless message consumers are balanced with message producers, there is the danger that a backlog of unconsumed messages might build up and seriously affect the performance of multiple applications.

In a point-to-point messaging topology, the relationship between message consumers and message producers is readily understood. You can obtain estimates of message production and consumption,

queue by queue, channel by channel. If there is a lack of balance, the bottlenecks are readily identified and then remedied.

It is harder to work out whether publishers and subscribers are balanced in a publish/subscribe topology. Start from each subscription that resolves to a clustered topic, and work back to the queue managers having publishers on the topic. Calculate the number of publications flowing to each subscriber from each queue manager.

Each publication that matches a subscription on a remote queue manager in the cluster (based on proxy subscriptions) is put to the SYSTEM.CLUSTER.TRANSMIT.QUEUE . If multiple remote queue managers have proxy subscriptions for that publication, multiple copies of the message are put to the transmission queue, targeted for different cluster sender channels.

Those publications are targeted at the SYSTEM.INTER.QMGR.PUBS queue on the remote queue managers. Each queue manager processes messages arriving on that queue and deliver them to the correct subscriptions on that queue manager.

For this reason, monitor the load at the following points where bottlenecks might arise:

- The individual subscription queues themselves:
  - This bottleneck would imply that the subscribing application is not consuming the publications as quick as they are being published.
- The SYSTEM.INTER.QMGR.PUBS queue:
  - The queue manager is receiving publications from one or more remote queue managers faster than it can distribute them to the local subscriptions.
- The cluster channels between the publishing queue manager, the subscribing queue managers, and the cluster transmission queues .(SYSTEM.CLUSTER.TRANSMIT.QUEUE by default) on the publishing queue manager:
  - Either one or more cluster channel is not running, or messages are being published to the local queue manager faster than the channels can deliver them to the remote queue manager.
- If the publishing application is using a queued publish/subscribe interface, the SYSTEM.BROKER.DEFAULT.STREAM queue and any other stream queues listed in the SYSTEM.QPUBSUB.QUEUE.NAMELIST , and the SYSTEM.BROKER.DEFAULT.SUBPOINT queue and any other subpoint queues, as listed in the SYSTEM.QPUBSUB.SUBPOINT.NAMELIST, also require consideration:
  - Messages are being put by local publishing applications faster than the local queue manager can process the messages.

Subscription performance considerations

As previously described, when a subscription is made on a queue manager for a topic string that resolves to a clustered topic, that queue manager must ensure that every other queue manager in the cluster has a proxy subscription in place for the topic. To achieve this result, the queue manager creates and sends a proxy subscription message to every other queue manager in the cluster.

Using the default configuration the only time that creating a subscription to a clustered topic does not result in new proxy subscriptions being sent, is when there is already a subscription on the local queue manager to exactly the same topic string. In this situation, no additional proxy subscriptions are required because arriving publications are delivered to all matching subscriptions, not just the original subscription for the topic string.

For an alternative to the default configuration, see “Disabling individual proxy subscriptions” on page 74.

Subscription selectors are not taken into account, so two subscriptions to the same topic string, but with different selectors, still share proxy subscriptions. This situation can also mean that publications that match the topic string are propagated to the subscriber queue manager, even if the publication does not match the subscription's selector.

An equivalent message to the proxy subscription message is created and sent to all queue managers when the last subscription to a topic string is deleted from a queue manager. This process removes the proxy subscriptions from the remote queue managers.

For these reasons, the size of the cluster and frequency of subscriptions to different topic strings can exert a significant load on the cluster itself and must be considered when planning the cluster and the topics to be used by the publish/subscribe applications.

When considering the load on the system from the proxy subscription traffic, in addition to monitoring the queues listed in the "Balancing producers and consumers" on page 72 section, monitor the following queues.

- The SYSTEM.INTER.QMGR.FANREQ queue on the subscriber queue manager.
- The SYSTEM.INTER.QMGR.CONTROL queue on all other queue managers in the cluster.

Any significant message backlog on these queues implies that either the rate of subscription change is too great for the system or a queue manager is not correctly functioning in the cluster. Either due to having publish/subscribe support disabled (see **PSMODE** in ALTER QMGR) or a problem occurring that requires further investigation, at which point check the queue manager error logs.

Reducing proxy subscription traffic

If proxy subscription overhead is high, steps should be taken to reduce this. This might be possible through general topic consolidation or through changing to a broadcast model for inter-queue manager publications.

It is a general publish/subscribe recommendation that the use of topic strings is assessed to see whether they can be consolidated in a way to reduce the overall load on resources for the system. The use of many distinct, transient, topic strings introduces some level of management overhead on each queue manager in the system where publishers or subscriptions are attached. Reducing the number and transient nature of topic strings and therefore the publishers and subscriptions to them reduces the impact on the system.

One method for reducing proxy subscription traffic is to locate subscriptions to the same topic string on the same queue manager. This method allows this queue manager to send a single proxy subscription to the other queue managers, rather than having multiple queue managers sending proxy subscriptions, each for their own set of subscriptions on the same topic string. This practice also optimizes publication routing across the cluster.

Disabling individual proxy subscriptions

In some situations, where the set of distinct topic strings being subscribed to across a cluster is large and continually changing, it might be better to change from a subscription propagation model to a publication broadcast model. This preferred model is where every publication on any cluster topic is automatically sent to every queue manager in the cluster, irrespective of the existence of subscriptions on those queue managers.

The receiving queue managers can then deliver the messages to the local subscriptions that exist, or discard the message. In this model, there is no need for individual proxy subscriptions to be created and deleted based on the existence of subscriptions. When running in this mode it is probable that the published message resource load increases as all publications are sent to all queue managers. Therefore, the queue managers in the cluster must have the capacity to handle this additional load.

Enable a broadcast model using the following configuration steps:

1. Every queue manager hosting subscriptions must be configured to not send proxy subscriptions that match local subscriptions to clustered topics. This configuration requires the following tuning parameter to be set in each queue manager `qm.ini` file, prior to the definition of cluster topics or the creation of subscriptions in the cluster:

   ```
   TuningParameters:
       pscProxySubFlags=1
   ```

2. After the tuning parameter is set, all queue managers must be restarted.

3. After the queue managers are restarted the clustered topic/topics can be defined. Each cluster topic must set **PROXYSUB** to FORCE.

Reversing the behavior

To reverse the mode of operation described previously in "Disabling individual proxy subscriptions" on page 74, use the following steps:

1. Remove the tuning parameter from the `qm.ini` file for every queue manager.
2. Restart every queue manager.
3. Issue the **REFRESH QMGR TYPE(PROXYSUB)** command on every queue manager hosting subscriptions.
4. Set **PROXYSUB** to FIRSTUSE on the clustered topic or topics.

CAUTION: In both the enabling and reversing of this behavior, if all the steps are not completed in the documented order, correct flow of publications to subscriptions might not occur.

Note: Implication of setting PROXYSUB (to FORCE)

As described previously in this topic, the **PROXYSUB(FORCE)** topic attribute can reduce the proxy subscription traffic, but it must be used with caution. The **PROXYSUB(FORCE)** attribute is propagated to every queue manager in the cluster, not just the queue manager that the topic was defined on. This instantly results in every queue manager in the cluster creating a wildcarded proxy subscription to every other queue manager. The result of this process is that every queue manager creates cluster sender channels to every other queue manager, and any published messages are sent to every queue manager.

Setting this property in a large or busy cluster can result in additional load on system resources.

Performance characteristics of publish/subscribe clusters

It is important to consider how changing attributes of a publish/subscribe cluster, such as adding a queue manager, topic, or subscription to the cluster affects the performance of applications running in the cluster.

Compare a point-to-point cluster with a publish/subscribe cluster in respect of two management tasks.

First, a point to point cluster:

1. When a new cluster queue is defined, the destination information is pushed to the full repository queue managers, and only sent to other cluster members when they first reference a cluster queue (for example, an application attempts to open it). This information is then cached locally by the queue manager to remove the need to remotely retrieve the information each time the queue is accessed.
2. Adding a queue manager to a cluster does not directly affect the load on other queue managers. Information about the new queue manager is pushed to the full repositories, but channels to the new queue manager from other queue managers in the cluster are only created and started when traffic begins to flow to or from the new queue manager.

In short, the load on a queue manager in a point-to-point cluster is related to the message traffic it handles for application programs and is not directly related to the size of the cluster.

Second, a publish/subscribe cluster:

1. When a new cluster topic is defined, the information is pushed to the full repository queue managers, and from there directly to all members of the cluster immediately, causing channels to be started to each member of the cluster from the full repositories if not already started.
2. When a subscription is created to a cluster topic on a new topic string, the information is pushed directly from that queue manager to all other members of the cluster immediately, causing channels to be started to each member of the cluster from that queue manager if not already started.
3. When a new queue manager joins an existing cluster, information about all clustered topics is pushed to it from the full repository queue managers. The new queue manager then synchronizes knowledge of all subscriptions to cluster topics in the cluster with all members of the cluster, causing channels to be created and started to each member of the cluster from the new queue manager.

In summary, cluster management load at any queue manager in the cluster grows with the number of queue managers, clustered topics, and proxy subscriptions within the cluster, irrespective of the local use of those cluster topics on each queue manager.

Key roles for publish/subscribe cluster queue managers
Similar to point-to-point clusters, there are two key roles for queue managers in a publish/subscribe cluster; as full repository queue managers and as cluster topic hosts.

Full repository
A full repository queue manager has the role of pushing object definitions out to other members of a cluster; in the case of publish/subscribe clusters, pushing clustered topic object definitions out to other members of the cluster.

Cluster topic host
A cluster topic host is a queue manager where a clustered topic object is defined. You can define clustered topic objects on any queue manager in the publish/subscribe cluster. The cluster topic object is pushed to the full repository queue managers, which then push it out to all the other queue managers in the cluster where it is cached for use by publishers and subscribers running on any queue managers in the cluster.

Availability and management

You should define two full repositories in a cluster to maximize the availability of cluster topic definitions in the cluster.

As for queued messaging clusters, in publish/subscribe clusters that have just two highly available computers among many computers, it is good practice to define the highly available computers as full repositories.

In queued clusters, you can increase the availability and throughput of a cluster queue by *defining* the same cluster queue on multiple queue managers in the cluster. Messages are then workload balanced across them. In contrast, in publish/subscribe clusters, a clustered topic is *available* on all queue managers in the cluster but no workload balancing of publish/subscribe traffic is performed. Instead, separate subscriptions and publishers should be spread across different queue managers, to spread the publish/subscribe load. If the queue manager on which you defined the cluster topic becomes unavailable, the other queue managers continue to process publish/subscribe requests for the topic.

However, if the queue manager on which you defined the cluster topic object is never made available again, then eventually the cached topic objects on the other queue managers are deleted and the topic becomes unavailable. This process happens after at least 60 days (depending on when the topic definition was last refreshed) from when the topic definition became unavailable.

With the 60 day period to recover the queue manager on which you defined cluster topic objects, there is little need to take special measures to make a cluster topic host highly available. The 60 day period is sufficient to cater for technical problems; the 60 day period is likely to be exceeded only because of administrative errors. To mitigate that possibility, if the cluster topic host is unavailable, all members of the cluster write error log messages hourly that their cached cluster topic object was not refreshed. Respond to this message by making sure that the queue manager on which the cluster topic object is defined, is running.

You might adopt the practice of defining the same cluster topic object on other queue managers. Each definition results in an additional cluster topic object being pushed out to the other queue managers in the cluster, including the other cluster topic hosts. Now if a cluster topic host becomes unavailable for over 60 days, only its version of the cluster topic object is removed from the other hosts. The other versions of the cluster topic object remain. It is a requirement that all definitions for a specific topic in a cluster are identical, otherwise it is difficult to ascertain which topic definition is being used by a queue manager. The most recent copy on any host is always the cluster topic object that is used.

Weigh the added protection of multiple cluster topic definitions against increased administrative complexity: with increased complexity comes a greater chance of human error.

Unlike hosting a clustered queue, being the host queue manager for a clustered topic definition does not introduce any additional application message traffic. That traffic is limited to the queue managers where the subscriptions are created and the messages published. It is possible to host the clustered topic on a queue manager that is doing neither. This situation means that although it is not mandatory, it is often sensible to host the clustered topics on the full repository queue managers for the cluster as these queue managers might be provisioned with higher levels of availability, and have tighter administrative control on them. This arrangement reduces the possibility of incorrectly modifying or deleting the definitions or even the queue manager.

Overlapping cluster support, and publish/subscribe
With IBM WebSphere MQ clusters, a single queue manager can be a member of more than one cluster. This arrangement is known as overlapping clusters. Clustered topics in a publish/subscribe clusters behave differently to queues when clusters are overlapped in a queue manager. This behavior must be clearly understood when using clustered publish/subscribe with overlapping clusters.

Unlike for a queue, there is no ability to associate a topic definition with more than one cluster. Therefore, the scope of proxy subscriptions created in a cluster is limited to the single cluster in which the clustered topic is defined. However, each queue manager has a single topic tree which includes all local topics and any known clustered topics, from any cluster that they are a member of. For this reason it is possible to architect such a system where publish/subscribe behaviour can be hard to understand.

Integrating multiple publish/subscribe clusters

For point-to-point messages, a reason for making a single queue manager a member of more than one cluster is to create a cluster gateway between two clusters. For more information about this subject, see Overlapping clusters . This cluster gateway enables point-to-point messages originating in one cluster to be routed to queries in another cluster. Publish/subscribe clusters inherit the capability of being overlapped from traditional queue manager clusters. However, you cannot use this mechanism to route publications and subscriptions from one cluster to another.

Instead, to pass publications and subscriptions from queue managers in one cluster to another, you must link the queue managers together using a publish/subscribe hierarchy. This arrangement can be achieved by explicitly creating a parent-child hierarchical relationship between one queue manager in one cluster with another queue manager in the other cluster. This relationship enables the flow of all proxy subscriptions between the clusters, and thus any matching publications. For more information about this relationship, see “Publish/subscribe hierarchies” on page 79.

A way to limit which publications and subscriptions flow between clusters is to use a gateway queue manager that is in neither cluster; see “Combine and isolate topic spaces in multiple clusters” on page 97.

Overlapping clusters, single topic tree

Each queue manager has a single Topic tree that includes local topics and all known clustered topics. A further consideration with overlapping two clusters, both using publish/subscribe, is that it is possible for a queue manager in each cluster to define a clustered topic with the same name, or to define differently named clustered topics which have the same topic string. On the queue managers that are members of both clusters, conflicts arise when they are informed of the multiple cluster topic definitions, one for each cluster. A problem is reported by the queue manager but the queue manager continues to operate, using only the most recent clustered topic definition. Therefore the behavior becomes nondeterministic and cannot be relied upon.

For this reason, overlapping clusters that are using clustered publish/subscribe must consider their topic definition namespace to span all clusters, and name their topic objects and structure their topic strings accordingly. You can then use queue managers in the overlap to publish and subscribe to both clusters predictably.

In Figure 33 on page 78, T_B and T_C are topic definitions that do not overlap. A publisher connected to QM3, in the cluster overlap, is able to publish to both topics in their respective clusters. A subscriber connected to QM3 in the overlap is able to subscribe to topics in both clusters.

An alternative way of thinking about Figure 33 on page 78 is to consider the proxy subscriptions. An application connected to queue manager QM3, subscribing on a topic that resolves to topic object T_B (which exists only in CLUSTER 1) results in proxy subscriptions being sent from queue manager QM3 to queue managers QM1 and QM2 only. An application connected to queue manager QM3 subscribes on

a topic that resolves to topic object T_C (which exists only in the CLUSTER 2). The subscription results in proxy subscriptions being sent from queue manager QM3 to queue managers QM4 and QM5 only.

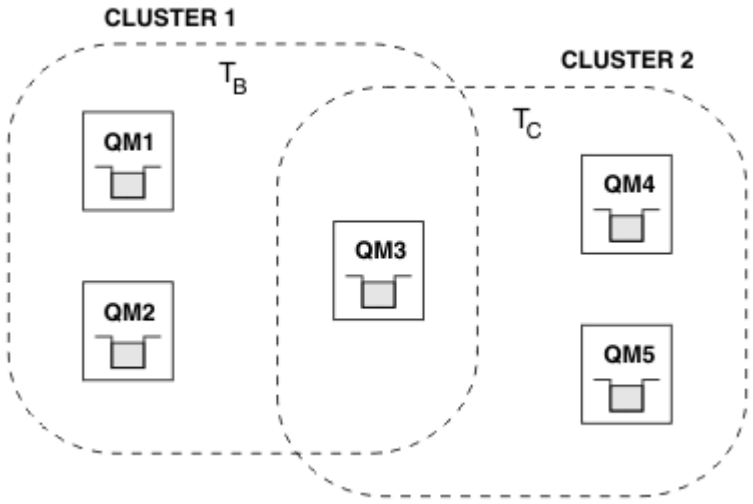


Figure 33. Overlapping clusters: two clusters each subscribing to different topics

Publishers and subscribers to queue managers that are not in the overlap can publish and subscribe to topics in their cluster only, for example a subscriber to a topic string on QM2 do not receive messages published to the same topic string published from QM5, irrespective of clustering the topics. To achieve this arrangement, a publish/subscribe hierarchy is required.

Overlapping clusters, wildcarded subscriptions

For the reasons in the previous section of this topic, care must be taken when using wildcards to subscribe to topics on a queue manager that is a member of multiple clusters.

In the previous example, assume that the two topic objects were configured as:

- T_B: Topic name `'Football'`, cluster `'CLUSTER1'`. topic string `'/Sport/Football'`
- T_C: Topic name `'Tennis'`, cluster `'CLUSTER2'`. topic string `'/Sport/Tennis'`

In this scenario, the two clustered topics are clearly separate, with no overlap in either the topic name or the topic string.

An application connected to QM3 can create a subscription to `'/Sport/Football'` and a subscription to `'/Sport/Tennis'`. They would then receive any publications from across the two clusters. However, as described in "Administrative topic objects" on page 36, if they were to subscribe to `'/Sport/#'`, with the intent of receiving publications on both `'/Sport/Football'` and `'/Sport/Tennis'`, this model is not recognized as a clustered topic in either cluster and therefore no proxy subscriptions would be created. They would then miss publications from other queue managers in either cluster.

As already described, it is not valid to create a clustered topic for `'/Sport/#'` in both CLUSTER 1 and CLUSTER 2, because these clustered topics would conflict and informational messages are written to the error logs to indicate this. However, it is 'allowed' to create such a topic in just one of the clusters, say CLUSTER 1. Now a subscription to `'/Sport/#'` in QM3 would result in proxy subscriptions being sent to the queue managers in CLUSTER 1 only, so still, publications to `'/Sport/Tennis'` from QM4 or QM5 would still fail to be received.

The only solution in this scenario is to continue to create two separate subscriptions.

REFRESH CLUSTER considerations for publish/subscribe clusters
Issuing the **REFRESH CLUSTER** command results in the queue manager temporarily discarding locally held information about a cluster, including any cluster topics and their associated proxy subscriptions.

The time taken from issuing the **REFRESH CLUSTER** command to the point that the queue manager regains a full knowledge of the necessary information for clustered publish/subscribe depends on the size of the cluster, the availability, and the responsiveness of the full repository queue managers.

During the refresh processing, disruption to publish/subscribe traffic in a publish/subscribe cluster occurs. For large clusters, use of the **REFRESH CLUSTER** command can disrupt the cluster while it is in progress, and again at 27 day intervals thereafter when the cluster objects automatically send status updates to all interested queue managers. See Refreshing in a large cluster can affect performance and availability of the cluster. For these reasons, the **REFRESH CLUSTER** command must be used in a publish/subscribe cluster only when under the guidance of your IBM Support Center.

The disruption to the cluster can appear externally as the following symptoms:

- Subscriptions to cluster topics on this queue manager are not receiving publications from publishers that are connected to other queue managers in the cluster.
- Messages that are published to cluster topics on this queue manager are not being propagated to subscriptions on other queue managers.
- Subscriptions to cluster topics on this queue manager created during this period are not consistently sending proxy subscriptions to other members of the cluster.
- Subscriptions to cluster topics on this queue manager deleted during this period are not consistently removing proxy subscriptions from other members of the cluster.
- 10-second pauses, or longer, in message delivery.
- **MQPUT** failures, for example, MQRC_PUBLICATION_FAILURE.
- Publications placed on the dead-letter queue with a reason of MQRC_UNKNOWN_REMOTE_Q_MGR

For these reasons publish/subscribe applications need to be quiesced before issuing the **REFRESH CLUSTER** command.

See also Usage Notes for **REFRESH CLUSTER** and Clustering: Using REFRESH CLUSTER best practices.

After a **REFRESH CLUSTER** command is issued on a queue manager in a publish/subscribe cluster, wait until all cluster queue managers and cluster topics have been successfully refreshed, then resynchronize proxy subscriptions as described in “Resynchronization of proxy subscriptions” on page 68. This arrangement requires cluster sender channels to be started from this queue manager to all other queue managers in the cluster. When all proxy subscriptions have been correctly resynchronized, restart your publish/subscribe applications.

If a **REFRESH CLUSTER** command is taking a long time to complete, monitor it by looking at the CURDEPTH of SYSTEM.CLUSTER.COMMAND.QUEUE .

Related concepts

Application issues seen when running REFRESH CLUSTER

Clustering: Using REFRESH CLUSTER best practices

Related reference

MQSC Commands reference: REFRESH CLUSTER

Publish/subscribe hierarchies
Queue managers can be grouped together in a hierarchy, where the hierarchy contains one or more queue managers that are directly connected. Queue managers are connected together using a connection-time parent and child relationship. When two queue managers are connected together for the first time, the child queue manager is connected to the parent queue manager.

When the parent and child queue managers are connected in a hierarchy there is no functional difference between them until you disconnect queue managers from the hierarchy.

Note: IBM WebSphere MQ hierarchical connections require that the queue manager attribute PSMODE is set to ENABLED.

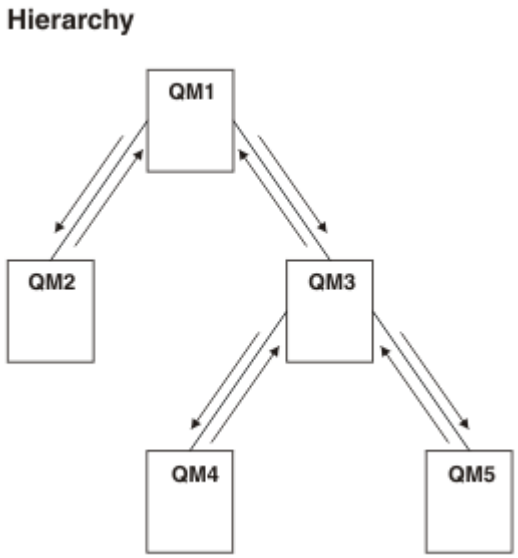


Figure 34. Simple publish/subscribe hierarchy

Connect a queue manager to a broker hierarchy
You can connect a local queue manager to a parent queue manager to modify a broker hierarchy.

Before you begin

1. Enable queued publish/subscribe mode. See Starting queued publish/subscribe.
2. This change is propagated to the parent queue manager using an IBM WebSphere MQ connection. There are two ways to establish the connection.
   - Connect the queue managers to an IBM WebSphere MQ cluster, see Adding a queue manager to a cluster
   - Establish a point-to-point channel connection using a transmission queue, or queue manager alias, with the same name as the parent queue manager. For more information about how to establish a point-to-point channel connection, see WebSphere MQ distributed-messaging techniques.

About this task

Use the `ALTER QMGR PARENT (PARENT_NAME)` runmqsc command to connect children to parents.

Distributed publish/subscribe is implemented by using queue manager clusters and clustered topic definitions. For interoperability with IBM WebSphere MQ Version 6.0 and WebSphere Message Broker Version 6.1 and WebSphere Event Broker Version 6.1 and earlier, you can also connect Version 7.1 or later queue managers to a broker hierarchy as long as queued publish/subscribe mode is enabled.

Procedure

ALTER QMGR PARENT(PARENT)

Example

The first example shows how to attach QM2 as a child of QM1, and then querying QM2 for its connection:

```
C:>runmqsc QM2
5724-H72 (C) Copyright IBM Corp. 1994, 2025. ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM2
alter qmgr parent(QM1)
     1 : alter qmgr parent(QM1)
```

```
AMQ8005: WebSphere MQ queue manager changed.
display pubsub all
     2 : display pubsub all
AMQ8723: Display pub/sub status details.
   QMNAME(QM2)                             TYPE(LOCAL)
   STATUS(ACTIVE)
AMQ8723: Display pub/sub status details.
   QMNAME(QM1)                             TYPE(PARENT)
   STATUS(ACTIVE)
```

The next example shows the result of querying QM1 for its connections:

```
C:\Documents and Settings\Admin>runmqsc QM1
5724-H72 (C) Copyright IBM Corp. 1994, 2025. ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM1.
display pubsub all
     2 : display pubsub all
AMQ8723: Display pub/sub status details.
   QMNAME(QM1)                             TYPE(LOCAL)
   STATUS(ACTIVE)
AMQ8723: Display pub/sub status details.
   QMNAME(QM2)                             TYPE(CHILD)
   STATUS(ACTIVE)
```

What to do next

You can define topics on one broker or queue manager that are available to publishers and subscribers on the connected queue managers. For more information, see Defining an administrative topic

Related concepts

Streams and topics

Introduction to WebSphere MQ publish/subscribe messaging

Related reference

DISPLAY PUBSUB

Disconnect a queue manager from a broker hierarchy

Disconnect a child queue manager from a parent queue manager in a broker hierarchy.

About this task

Use the **ALTER QMGR** command to disconnect a queue manager from a broker hierarchy. You can disconnect a queue manager in any order at any time.

The corresponding request to update the parent is sent when the connection between the queue managers is running.

Procedure

```
ALTER QMGR PARENT('')
```

Example

```
C:\Documents and Settings\Admin>runmqsc QM2
5724-H72 (C) Copyright IBM Corp. 1994, 2025. ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM2.
    1 : alter qmgr parent('')
AMQ8005: WebSphere MQ queue manager changed.
    2 : display pubsub type(child)
AMQ8147: WebSphere MQ object  not found.
display pubsub type(parent)
    3 : display pubsub type(parent)
AMQ8147: WebSphere MQ object  not found.
```

What to do next

You can delete any streams, queues and manually defined channels that are no longer needed.

Publish/subscribe hierarchy example: Scenario 1
Set up a publish/subscribe hierarchy topology using point-to-point channels with queue manager name alias.

About this task

These scenarios set up a publish/subscribe hierarchy in different ways to establish the connection between queue managers. These scenarios all use a parent queue manager called QM1, and two child queue managers called QM2, and QM3.

Scenario 1 is split into smaller sections to make the process easier to follow.

Scenario 1 part 1: Create the queue managers

Procedure

1. Create and start three queue managers called QM1, QM2, and QM3 using the following commands:

```
crtmqm -u SYSTEM.DEAD.LETTER.QUEUE QM1
strmqm QM1

crtmqm -u SYSTEM.DEAD.LETTER.QUEUE QM2
strmqm QM2

crtmqm -u SYSTEM.DEAD.LETTER.QUEUE QM3
strmqm QM3
```

2. Enable the queue manager publish/subscribe mode by using the following command on all three queue managers:

```
ALTER QMGR PSMODE(ENABLED)
```

Scenario 1 part 2: Point-to-point channel connections

About this task

Establish point-to-point channel connections between queue managers using a queue manager alias with the same name as the parent queue manager.

Procedure

1. Define a transmission queue and queue manager alias on QM2 to QM1. Define a sender channel to QM1 and a receiver channel for the sender channel created on QM1 for QM2:

```
DEFINE QLOCAL(QM1.XMITQ) USAGE(XMITQ)

DEFINE QREMOTE (QM1) RNAME('') RQMNAME(QM1) XMITQ(QM1.XMITQ)

DEFINE CHANNEL('QM2.TO.QM1') CHLTYPE(SDR) CONNAME('localhost(9999)') XMITQ(QM1.XMITQ)
TRPTYPE(TCP)

DEFINE CHANNEL('QM1.TO.QM2') CHLTYPE(RCVR) TRPTYPE(TCP)
```

2. Define a transmission queue and queue manager alias on QM3 to QM1. Define sender channel to QM1 and a receiver channel for the sender channel created on QM1 for QM3:

```
DEFINE QLOCAL(QM1.XMITQ) USAGE(XMITQ)

DEFINE QREMOTE (QM1) RNAME('') RQMNAME(QM1) XMITQ(QM1.XMITQ)

DEFINE CHANNEL('QM3.TO.QM1') CHLTYPE(SDR) CONNAME('localhost(9999)') XMITQ(QM1.XMITQ)
TRPTYPE(TCP)

DEFINE CHANNEL('QM1.TO.QM3') CHLTYPE(RCVR) TRPTYPE(TCP)
```

3. Define a transmission queue and queue manager alias on QM1 to QM2 and QM3. Define sender channel to QM2 and QM3, and a receiver channel for the sender channels created on QM2 and QM3 for QM1:

```
DEFINE QLOCAL(QM2.XMITQ) USAGE(XMITQ)

DEFINE QREMOTE (QM2) RNAME('') RQMNAME(QM2) XMITQ(QM2.XMITQ)

DEFINE CHANNEL('QM1.TO.QM2') CHLTYPE(SDR) CONNAME('localhost(7777)') XMITQ(QM2.XMITQ)
TRPTYPE(TCP)

DEFINE CHANNEL('QM2.TO.QM1') CHLTYPE(RCVR) TRPTYPE(TCP)

DEFINE QLOCAL(QM3.XMITQ) USAGE(XMITQ)

DEFINE QREMOTE (QM3) RNAME('') RQMNAME(QM3) XMITQ(QM3.XMITQ)

DEFINE CHANNEL('QM1.TO.QM3') CHLTYPE(SDR) CONNAME('localhost(8888)') XMITQ(QM3.XMITQ)
TRPTYPE(TCP)

DEFINE CHANNEL('QM3.TO.QM1') CHLTYPE(RCVR) TRPTYPE(TCP)
```

4. Start the appropriate listeners on the queue managers:

```
runmqlsr -m QM1 -t TCP -p 9999 &
runmqlsr -m QM2 -t TCP -p 7777 &
runmqlsr -m QM3 -t TCP -p 8888 &
```

5. Start the following channels:
   a. On QM1:

```
START CHANNEL('QM1.TO.QM2')

START CHANNEL('QM1.TO.QM3')
```

   b. On QM2:

```
START CHANNEL('QM2.TO.QM1')
```

   c. On QM3:

```
START CHANNEL('QM3.TO.QM1')
```

6. Check that all the channels have started:

```
DISPLAY CHSTATUS('QM1.TO.QM2')

DISPLAY CHSTATUS('QM1.TO.QM3')

DISPLAY CHSTATUS('QM2.TO.QM1')

DISPLAY CHSTATUS('QM3.TO.QM1')
```

Scenario 1 part 3: Connect queue managers and define a topic

About this task

Connect the child queue managers QM2 and QM3 to the parent queue manager QM1.

Procedure

1. On QM2 and QM3, set the parent queue manager to QM1:

```
ALTER QMGR PARENT (QM1)
```

2. Run the following command on all queue managers to check that the child queue managers are connected to the parent queue manager:

```
DISPLAY PUBSUB TYPE(ALL)
```

3. Define a topic object:

```
define topic(FOOTBALL) TOPICSTR('Sport/Soccer')
```

Scenario 1 part 4: Publish and subscribe the topic

About this task

Use the `amqspub.exe` and `amqssub.exe` applications to publish and subscribe the topic.

Procedure

1. Run this command in the first command window:

```
amqspub Sport/Soccer QM2
```

2. Run this command in the second command window:

```
amqssub Sport/Soccer QM1
```

3. Run this command in the third command window:

```
amqssub Sport/Soccer QM3
```

Results

The `amqssub.exe` applications in the second and third command windows receive the messages published in the first command window.

Related tasks

"Publish/subscribe hierarchy example: Scenario 2" on page 84
Set up a publish/subscribe hierarchy topology using point-to-point channels with the transmission queue name the same as the remote queue manager.

"Publish/subscribe hierarchy example: Scenario 3" on page 87
Add a queue manager to a hierarchy topology using a cluster channel.

Publish/subscribe hierarchy example: Scenario 2
Set up a publish/subscribe hierarchy topology using point-to-point channels with the transmission queue name the same as the remote queue manager.

About this task

These scenarios set up a publish/subscribe hierarchy in different ways to establish the connection between queue managers. These scenarios all use a parent queue manager called `QM1`, and two child queue managers called `QM2`, and `QM3`.

Scenario 2 is split into smaller sections to make the process easier to follow. This scenario reuses Scenario 1 part 1, Scenario 1 part 3, and Scenario 1 part 4 from "Publish/subscribe hierarchy example: Scenario 1" on page 82.

Scenario 2 part 1: Create queue manager and set PSMODE

Procedure

1. Create and start three queue managers called `QM1`, `QM2`, and `QM3` using the following commands:

```
crtmqm -u SYSTEM.DEAD.LETTER.QUEUE QM1
strmqm QM1

crtmqm -u SYSTEM.DEAD.LETTER.QUEUE QM2
strmqm QM2
```

```
crtmqm -u SYSTEM.DEAD.LETTER.QUEUE QM3
strmqm QM3
```

2. Enable the queue manager publish/subscribe mode by using the following command on all three queue managers:

```
ALTER QMGR PSMODE(ENABLED)
```

Scenario 2 part 2: Point-to-point channel connections

About this task

Establish point-to-point channel connections between a queue manager using a transmission queue with the same name as the parent queue manager.

Procedure

1. Define a transmission queue on QM2 to QM1. Define a sender channel to QM1 and a receiver channel for the sender channel for QM2 created on QM1:

```
DEFINE QLOCAL(QM1) USAGE(XMITQ)

DEFINE CHANNEL('QM2.TO.QM1') CHLTYPE(SDR) CONNAME('localhost(9999)') XMITQ(QM1) TRPTYPE(TCP)

DEFINE CHANNEL('QM1.TO.QM2') CHLTYPE(RCVR) TRPTYPE(TCP)
```

2. Define a transmission queue on QM3 to QM1. Define sender channel to QM1 and a receiver channel for the sender channel created on QM1 for QM3:

```
DEFINE QLOCAL(QM1) USAGE(XMITQ)

DEFINE CHANNEL('QM3.TO.QM1') CHLTYPE(SDR) CONNAME('localhost(9999)') XMITQ(QM1) TRPTYPE(TCP)

DEFINE CHANNEL('QM1.TO.QM3') CHLTYPE(RCVR) TRPTYPE(TCP)
```

3. Define transmission queues on QM1 to QM2 and QM3. Define sender channels to QM2 and QM3, and a receiver channel for the sender channels created on QM2 and QM3 for QM1:

```
DEFINE QLOCAL(QM2) USAGE(XMITQ)

DEFINE CHANNEL('QM1.TO.QM2') CHLTYPE(SDR) CONNAME('localhost(7777)') XMITQ(QM2) TRPTYPE(TCP)

DEFINE CHANNEL('QM2.TO.QM1') CHLTYPE(RCVR) TRPTYPE(TCP)

DEFINE QLOCAL(QM3) USAGE(XMITQ)

DEFINE CHANNEL('QM1.TO.QM3') CHLTYPE(SDR) CONNAME('localhost(8888)') XMITQ(QM3) TRPTYPE(TCP)

DEFINE CHANNEL('QM3.TO.QM1') CHLTYPE(RCVR) TRPTYPE(TCP)
```

4. Start the appropriate listeners on the queue managers:

```
runmqlsr -m QM1 -t TCP -p 9999 &
runmqlsr -m QM2 -t TCP -p 7777 &
runmqlsr -m QM3 -t TCP -p 8888 &
```

5. Start the following channels:

   a. On QM1:

```
START CHANNEL('QM1.TO.QM2')

START CHANNEL('QM1.TO.QM3')
```

   b. On QM2:

```
START CHANNEL('QM2.TO.QM1')
```

c. On QM3:

```
START CHANNEL('QM3.TO.QM1')
```

6. Check that all the channels have started:

```
DISPLAY CHSTATUS('QM1.TO.QM2')
DISPLAY CHSTATUS('QM1.TO.QM3')
DISPLAY CHSTATUS('QM2.TO.QM1')
DISPLAY CHSTATUS('QM3.TO.QM1')
```

Scenario 2 part 3: Connect queue managers and define a topic

About this task

Connect the child queue managers QM2 and QM3 to the parent queue manager QM1.

Procedure

1. On QM2 and QM3, set the parent queue manager to QM1:

```
ALTER QMGR PARENT (QM1)
```

2. Run the following command on all queue managers to check that the child queue managers are connected to the parent queue manager:

```
DISPLAY PUBSUB TYPE(ALL)
```

3. Define a topic object:

```
define topic(FOOTBALL) TOPICSTR('Sport/Soccer')
```

Scenario 2 part 4: Publish and subscribe the topic

About this task

Use the `amqspub.exe` and `amqssub.exe` applications to publish and subscribe the topic.

Procedure

1. Run this command in the first command window:

```
amqspub Sport/Soccer QM2
```

2. Run this command in the second command window:

```
amqssub Sport/Soccer QM1
```

3. Run this command in the third command window:

```
amqssub Sport/Soccer QM3
```

Results

The `amqssub.exe` applications in the second and third command windows receive the messages published in the first command window.

Related tasks

“Publish/subscribe hierarchy example: Scenario 1” on page 82

Set up a publish/subscribe hierarchy topology using point-to-point channels with queue manager name alias.

"Publish/subscribe hierarchy example: Scenario 3" on page 87
Add a queue manager to a hierarchy topology using a cluster channel.

Publish/subscribe hierarchy example: Scenario 3
Add a queue manager to a hierarchy topology using a cluster channel.

About this task

These scenarios set up a publish/subscribe hierarchy in different ways to establish the connection between queue managers. These scenarios all use a parent queue manager called QM1, and two child queue managers called QM2, and QM3.

Scenario 3 is split into smaller sections to make the process easier to follow. This scenario reuses Scenario 1 part 1, Scenario 1 part 3, and Scenario 1 part 4 from "Publish/subscribe hierarchy example: Scenario 1" on page 82.

This scenario creates a cluster called DEMO where QM1 and QM2 are full repositories, and QM3 is a partial repository. Queue manager QM1 is the parent of queue managers QM2 and QM3.

Scenario 2 part 1: Create queue manager and set PSMODE

Procedure

1. Create and start three queue managers called QM1, QM2, and QM3 using the following commands:

```
crtmqm -u SYSTEM.DEAD.LETTER.QUEUE QM1
strmqm QM1

crtmqm -u SYSTEM.DEAD.LETTER.QUEUE QM2
strmqm QM2

crtmqm -u SYSTEM.DEAD.LETTER.QUEUE QM3
strmqm QM3
```

2. Enable the queue manager publish/subscribe mode by using the following command on all three queue managers:

```
ALTER QMGR PSMODE(ENABLED)
```

Scenario 2 part 2: Point-to-point channel connections

About this task

Establish point-to-point channel connections between queue managers a cluster.

Procedure

1. On QM1 and QM2, set the **REPOS** parameter to the name of the cluster DEMO:

```
ALTER QMGR REPOS(DEMO)
```

2. Start the appropriate listeners on the queue managers:

```
runmqlsr -m QM1 -t TCP -p 9999 &
runmqlsr -m QM2 -t TCP -p 7777 &
runmqlsr -m QM3 -t TCP -p 8888 &
```

3. Define the cluster receiver channel on each queue manager:

a. On QM1:

```
DEFINE CHANNEL(TO.QM1) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('localhost(9999)')
CLUSTER(DEMO)
```

b. On QM2:

```
DEFINE CHANNEL(TO.QM2) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('localhost(7777)')
CLUSTER(DEMO)
```

c. On QM3:

```
DEFINE CHANNEL(TO.QM3) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('localhost(8888)')
CLUSTER(DEMO)
```

4. Define a cluster sender channel to a full repository on each queue manager in the cluster:

a. On QM1:

```
DEFINE CHANNEL(TO.QM2) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('localhost(7777)')
CLUSTER(DEMO)
```

b. On QM2:

```
DEFINE CHANNEL(TO.QM1) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('localhost(9999)')
CLUSTER(DEMO)
```

c. QM3 can have a cluster sender channel to either full repository on QM1 or QM2. This example defines the channel to QM1:

```
DEFINE CHANNEL(TO.QM1) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('localhost(9999)')
CLUSTER(DEMO)
```

Scenario 2 part 3: Connect queue managers and define a topic

About this task

Connect the child queue managers QM2 and QM3 to the parent queue manager QM1.

Procedure

1. On QM2 and QM3, set the parent queue manager to QM1:

```
ALTER QMGR PARENT (QM1)
```

2. Run the following command on all queue managers to check that the child queue managers are connected to the parent queue manager:

```
DISPLAY PUBSUB TYPE(ALL)
```

3. Define a topic object:

```
define topic(FOOTBALL) TOPICSTR('Sport/Soccer')
```

Scenario 2 part 4: Publish and subscribe the topic

About this task

Use the `amqspub.exe` and `amqssub.exe` applications to publish and subscribe the topic.

Procedure

1. Run this command in the first command window:

```
amqspub Sport/Soccer QM2
```

2. Run this command in the second command window:

```
amqssub Sport/Soccer QM1
```

3. Run this command in the third command window:

```
amqssub Sport/Soccer QM3
```

Results

The `amqssub.exe` applications in the second and third command windows receive the messages published in the first command window.

Related tasks

“Publish/subscribe hierarchy example: Scenario 1” on page 82
Set up a publish/subscribe hierarchy topology using point-to-point channels with queue manager name alias.

“Publish/subscribe hierarchy example: Scenario 2” on page 84
Set up a publish/subscribe hierarchy topology using point-to-point channels with the transmission queue name the same as the remote queue manager.

Controlling the flow of publications and subscriptions

Queue managers that are connected together into a distributed publish/subscribe topology share a common federated topic space. You can control the flow of publications and subscriptions within the topology by choosing whether each publication and subscription is either local or global.

Local publications and subscriptions are not propagated beyond the queue manager to which the publisher or subscriber is connected.

You can control the extent of topic spaces created by connecting queue managers together in clusters or hierarchies. In a publish/subscribe cluster, the topic object must be 'clustered', or all elements stay local and the publication or subscription has no effect.

A subscription, when it matches topic strings in different publications, can resolve to different topic objects. These are called overlapping topics. The topic object that is associated with a publication for a particular match provides the topic attributes, and determines for example, if the subscriber is to receive the publication.

Publication scope

The scope of a publication controls whether queue managers forward a publication to remote queue managers. Use the **PUBSCOPE** topic attribute to administer the scope of publications.

If a publication is not forwarded to remote queue managers, only local subscribers receive the publication.

The **PUBSCOPE** topic attribute is used to determine the scope of publications made to a specific topic. You can set the attribute to one of the following values:

QMGR
: The publication is delivered only to local subscribers. These publications are called *local publications*. Local publications are not forwarded to remote queue managers and therefore are not received by subscribers connected to remote queue managers.

ALL
: The publication is delivered to local subscribers and subscribers connected to remote queue managers. These publications are called *global publications*.

ASPARENT
: Use the **PUBSCOPE** setting of the parent.

Publishers can also specify whether a publication is local or global using the MQPMO_SCOPE_QMGR put message option. If this option is used, it overrides any behavior that has been set using the **PUBSCOPE** topic attribute.

Subscription scope
The scope of a subscription controls whether a subscription on one queue manager receives publications that are published on another queue manager in a publish/subscribe cluster or hierarchy, or only publications from local publishers.

Limiting the subscription scope to a queue manager stops proxy subscriptions from being forwarded to other queue managers in the publish/subscribe topology. This reduces inter-queue manager publish/subscribe messaging traffic.

The **SUBSCOPE** topic attribute is used to determine the scope of subscriptions that are made to a specific topic. You can set the attribute to one of the following values:

QMGR
A subscription receives only local publications, and proxy subscriptions are not propagated to remote queue managers.

ALL
A proxy subscription is propagated to remote queue managers, and the subscriber receives local and remote publications.

ASPARENT
Use the **SUBSCOPE** setting of the parent.

Individual subscribers can override the **SUBSCOPE** setting of ALL by specifying the MQSO_SCOPE_QMGR subscription option when creating a subscription. A subscription can override a topic's **SUBSCOPE** setting of ALL.

Note: Individual subscribers can only restrict the **SUBSCOPE** of the topic. When an individual subscription has **SUBSCOPE** set to ALL, the subscription honors the matching topics' **SUBSCOPE** setting.

Combining publication and subscription scopes
In WebSphere MQ versions 7 onwards, publication and subscription scope work independently to determine the flow of publications between queue managers.

Publications can flow to all queue managers that are connected in a publish/subscribe topology, or only to the local queue manager. Similarly for proxy subscriptions. Which publications match a subscription is governed by the combination of these two flows.

Publications and subscriptions can both be scoped to QMGR or ALL. If a publisher and a subscriber are both connected to the same queue manager, scope settings do not affect which publications the subscriber receives from that publisher.

If the publisher and subscriber are connected to different queue managers, both settings must be ALL to receive remote publications.

Suppose publishers are connected to different queue managers. If you want a subscriber to receive publications from any publisher, set the subscription scope to ALL. You can then decide, for each publisher, whether to limit the scope of its publications to subscribers local to the publisher.

Suppose subscribers are connected to different queue managers. If you want the publications from a publisher to be sent to all the subscribers, set the publication scope to ALL. If you want a subscriber to receive publications only from a publisher connected to the same queue manager, set the subscription scope to QMGR.

In version 6, and earlier, publication and subscription scope not only governed which publications flowed. In addition the scope of the publication had to match the scope of the subscription.

Example: football results service

Suppose you are a member team in a football league. Each team has a queue manager connected to all the other teams in a publish/subscribe cluster.

The teams publish the results of all the games played on their home ground using the topic, `Football/result/`*`Home team name/Away team name`*. The strings in italics are variable topic names, and the publication is the result of the match.

Each club also republishes the results just for the club using the topic string `Football/myteam/`*`Home team name/Away team name`*.

Both topics are published to the whole cluster.

The following subscriptions have been set up by the league so that fans of any team can subscribe to the results in three interesting ways.

Notice that you can set up cluster topics with `SUBSCOPE(QMGR)`. The topic definitions are propagated to each member of the cluster, but the scope of the subscription is just the local queue manager. Thus subscribers at each queue manager receive different publications from the same subscription.

Receive all results

```
DEFINE TOPIC(A) TOPICSTR('Football/result/') CLUSTER SUBSCOPE(ALL)
```

Receive all home results

```
DEFINE TOPIC(B) TOPICSTR('Football/result/') CLUSTER SUBSCOPE(QMGR)
```

Because the subscription has `QMGR` scope, only results published at the home ground are matched.

Receive all my teams results

```
DEFINE TOPIC(C) TOPICSTR('Football/myteam/') CLUSTER SUBSCOPE(QMGR)
```

Because the subscription has `QMGR` scope, only the local team results, which are republished locally, are matched.

Topic spaces

A topic space is the set of topics you can subscribe on. A subscriber connected to a queue manager in a distributed publish/subscribe topology has a topic space that potentially includes topics defined on connected queue managers.

Topics are initially created administratively, when you define a topic object or durable subscription, or dynamically when an application creates a publication or subscription dynamically.

Topics are propagated to other queue managers both through proxy subscriptions, and by creating administrative cluster topic objects. Proxy subscriptions result in publications being forwarded from the queue manager to which a publisher is connected, to the queue managers of subscribers. Proxy subscriptions are the mechanism by which topics defined on different queue managers are combined into a common topic space.

Proxy subscriptions are propagated between all queue managers that are connected together by parent-child relationships in a queue manager hierarchy. The result is, you can subscribe on one queue manager to a topic defined on any other queue manager in the hierarchy. As long as there is a connected path between the queue managers, it does not matter how the queue managers are connected.

Proxy subscriptions are also propagated for *cluster* topics between all the members of a cluster. A cluster topic is a topic that is attached to a topic object that has the **CLUSTER** attribute, or inherits the attribute from its parent. Topics that are not cluster topics are known as local topics and are not replicated to the cluster. No proxy subscriptions are propagated to the cluster from subscriptions to local topics.

To summarize, proxy subscriptions are created for subscribers in two circumstances.

1. A queue manager is a member of a hierarchy, and a proxy subscription is forwarded to the parent and children of the queue manager.
2. A queue manager is a member of a cluster, and the subscription topic string resolves to a topic that is associated with a cluster topic object. Proxy subscriptions are forwarded to all members of the cluster. See “Overlapping topics” on page 100 for more information about complications.

If a queue manager is a member of a cluster and a hierarchy, proxy subscriptions are propagated by both mechanisms without delivering duplicate publications to the subscriber.

The effect of creating a cluster topic object is twofold. Proxy subscriptions to a topic are sent to other members of the cluster when a subscription resolves to a cluster topic. It also sends a copy of the topic object to the other members of the cluster. The effect of forwarding cluster topic objects is to simplify the administration of topics. Typically, cluster topic objects are defined on a single queue manager in the cluster, called the cluster topic host.

The topics spaces of three publish/subscribe topologies are described in the following list:

- "Case 1. Publish/subscribe clusters" on page 92.
- "Case 2. Publish/subscribe hierarchies in version 7" on page 93.
- "Case 3. Publish/subscribe hierarchies and streams in version 6" on page 93.

In separate topics, the following tasks describe how to combine topic spaces.

- "Create a single topic space in a publish/subscribe cluster" on page 94.
- "Add a version 7 queue manager to existing version 6 topic spaces" on page 95.
- "Combine the topic spaces of multiple clusters" on page 96.
- "Combine and isolate topic spaces in multiple clusters" on page 97
- "Publish and subscribe to topic spaces in multiple clusters" on page 99

Case 1. Publish/subscribe clusters

In the example, assume that the queue manager is *not* connected to a publish/subscribe hierarchy.

If a queue manager is a member of a publish/subscribe cluster, its topic space is made up from local topics and cluster topics. Local topics are associated with topic objects without the **CLUSTER** attribute. If a queue manager has local topic object definitions, its topic space is different from another queue manager in the cluster that also has its own locally defined topic objects.

In a publish/subscribe cluster, you cannot subscribe to a topic defined on another queue manager, unless the topic you subscribe to resolves to a cluster topic object.

Conflicting definitions of a cluster topic defined elsewhere in the cluster are resolved in favor of the most recent definition. At any point in time, if a cluster topic has been multiply defined, the cluster topic definition on different queue managers might be different.

A local definition of a topic object, whether the definition is for a cluster topic or a local topic, takes precedence over the same topic object defined elsewhere in the cluster. The locally defined topic is used, even if the object defined elsewhere is more recent.

Set either of the **PUBSCOPE** and **SUBSCOPE** options to `QMGR`, to prevent a publication or a subscription on a cluster topic flowing to different queue managers in the cluster.

Suppose you define a cluster topic object `Alabama` with the topic string `USA/Alabama` on your cluster topic host. The result is as follows:

1. The topic space at the cluster topic host now includes the cluster topic object `Alabama` and the topic `USA/Alabama`.
2. The cluster topic object `Alabama` is replicated to all queue managers in the cluster where it is combined with the topic space at each queue manager. What happens at each queue manager in the cluster depends on whether the topic object `Alabama` exists at a queue manager.
   - If `Alabama` is a new topic object, the queue manager adds the cluster topic object `Alabama`, and the topic `USA/Alabama`, to its topic space.
   - If `Alabama` is a local definition, the cluster topic object `Alabama` is added. Unless the local definition is deleted, the remotely defined cluster topic object is ignored. The queue manager retains both definitions.

- If Alabama is an older cluster topic object defined elsewhere, it is replaced by the newer cluster topic object.

3. An application or administrator, anywhere in the cluster, can create a subscription to USA/Alabama by referring to the Alabama topic object.
4. An application, anywhere in the cluster, using the topic string USA/Alabama directly can create a subscription that inherits the attributes of the topic object Alabama. The Alabama topic object is inherited by a subscription formed from any topic string beginning with USA/Alabama,

   If there is another definition of the Alabama topic object on one of the other queue managers, it takes precedence over the definition on the cluster topic host. The local object might have a cluster attribute, or it might not. The cluster attribute might refer to the same cluster or another cluster. Try to avoid these multiple definition cases. They lead to differences in behavior.
5. If the topic object Alabama has the **PUBSCOPE** attribute ALL, subscriptions that resolve to Alabama are sent to all the other queue managers in the cluster.

   Set the Alabama **PUBSCOPE** attribute to QMGR to prevent publications flowing from publishers to subscribers connected to different queue managers in the cluster.

   The Alabama topic object is replicated to every queue manager in the cluster, so the **PUBSCOPE** and **PUBSCOPE** attributes apply to all the queue managers in the cluster.

It is important that a cluster topic object is associated with the same topic string everywhere in the cluster. You cannot modify the topic string with which a topic object is associated. To associate the same topic object with a different topic string, you must delete the topic object and re-create it with the new topic string. If the topic is clustered, the effect is to delete the copies of the topic object stored on the other members of the cluster and then to create copies of the new topic object everywhere in the cluster. The copies of the topic object all refer to the same topic string.

However, you can create a duplicate definition of a topic object on another queue manager in the cluster, with a different topic string. Always try to avoid duplicates by managing cluster topic hosts on one queue manager. See “Multiple cluster topic definitions” on page 70 for more information on this important point. Multiple definitions of the same topic object with different topic strings can produce different results depending how and where the topic is referenced.

Case 2. Publish/subscribe hierarchies in version 7

In the example, assume that the queue manager is *not* a member of a publish/subscribe cluster.

In version 7, if a queue manager is a member of a publish/subscribe hierarchy, its topic space consists of all the topics defined locally and on connected queue managers. The topic space of all the queue managers in a hierarchy is the same. There is no division of topics into local topics and cluster topics.

Set either of the **PUBSCOPE** and **SUBSCOPE** options to QMGR, to prevent a publication on a topic flowing from a publisher to a subscriber connected to different queue managers in the hierarchy.

Suppose you define a topic object Alabama with the topic string USA/Alabama on queue manager QMA. The result is as follows:

1. The topic space at QMA now includes the topic object Alabama and the topic string USA/Alabama.
2. An application or administrator can create a subscription at QMA using the topic object name Alabama.
3. An application can create a subscription to any topic, including USA/Alabama, at any queue manager in the hierarchy. If QMA has not been defined locally, the topic USA/Alabama resolves to the topic object SYSTEM.BASE.TOPIC.

Case 3. Publish/subscribe hierarchies and streams in version 6

Before version 7, the topic space was divided into separate streams, which included the default stream that was present on all queue managers. Publications cannot flow between different streams. If named streams are used, the topic spaces at different queue managers might be different. Topics are divided into topics in the default stream, and topics in different named streams.

Note: Each named stream forms a separate topic space. To form a connected topology each named stream must exist on the connected queue managers. Suppose stream X is defined on QMA and QMC, but not on QMB. If QMA is the parent of QMB, and QMB is the parent of QMC, no topics in stream X can flow between QMA and QMC.

Setting both of the **PUBSCOPE** and **SUBSCOPE** options either to QMGR or to ALL requires a publisher and subscriber to a topic to exchange only publications for local consumption, or to exchange only publications for global consumption.

From version 7, streams are not available using the publish/subscribe API. If you use queued publish/subscribe on a version 7 queue manager, streams are mapped to different topic objects that can simulate the effect of streams. A stream is simulated by creating a topic object that is the root topic for all the topics in the stream. The queue manager maps publications and subscriptions between the stream and the corresponding root topic of each tree.

Combining topics spaces
Combine the topic space of a queue manager with other queue managers in a publish/subscribe cluster or hierarchy. Combine publish/subscribe clusters, and publish/subscribe clusters with hierarchies.

You can create different publish/subscribe topics spaces by using the building blocks of **CLUSTER**, **PUBSCOPE** and **SUBSCOPE** attributes, publish/subscribe clusters, and publish/subscribe hierarchies.

Starting from the example of scaling up from a single queue manager to a publish/subscribe cluster, the following scenarios illustrate different publish/subscribe topologies.

Create a single topic space in a publish/subscribe cluster
Scale up a publish/subscribe system to run on multiple queue managers. Use a publish/subscribe cluster to provide each publisher and subscriber with a single identical topic space.

Before you begin
You have implemented a publish/subscribe system on a single version 7 queue manager.

Always create topic spaces with their own root topics, rather than relying on inheriting the attributes of SYSTEM.BASE.TOPIC. If you scale your publish/subscribe system up to a cluster, you can define your root topics as cluster topics, on the cluster topic host, and then all your topics are shared throughout the cluster.

About this task
You now want to scale the system up to support more publishers and subscribers and have every topic visible throughout the cluster.

Procedure
1. Create a cluster to use with the publish/subscribe system.

   If you have an existing traditional cluster, for performance reasons it is better to set up a new cluster for the new publish subscribe system. You can use the same servers for the cluster repositories of both clusters
2. Choose one queue manager, possibly one of the repositories, to be the cluster topic host.
3. Ensure every topic that is to be visible throughout the publish/subscribe cluster resolves to an administrative topic object.
   Set the **CLUSTER** attribute naming the publish/subscribe cluster.

What to do next
Connect publisher and subscriber applications to any queue managers in the cluster.

Create administrative topic objects that have the **CLUSTER** attribute. The topics are also propagated throughout the cluster. Publisher and subscriber programs use the administrative topics so that their behavior is not altered by being connected to different queue managers in the cluster

If you need SYSTEM.BASE.TOPIC to act like a cluster topic on every queue manager, you need to modify it on every queue manager.

Add a version 7 queue manager to existing version 6 topic spaces
Extend an existing version 6 publish/subscribe system to interoperate with a version 7 queue manager, sharing the same topic spaces.

Before you begin

You have an existing version 6 publish/subscribe system.

You have installed WebSphere MQ version 7 on a new server and configured a queue manager.

About this task

You want to extend your existing version 6 publish/subscribe system to work with version 7 queue managers.

You have decided to stabilize development of the version 6 publish/subscribe system that uses the queued publish/subscribe interface. You intend to add extensions to the system using the version 7 MQI. You have no plans now to rewrite the queued publish/subscribe applications.

You intend to upgrade the version 6 queue managers to version 7 in the future. For now, you are continuing to run the existing queued publish/subscribe applications on the version 7 queue managers.

Procedure

1. Create one set of sender-receiver channels to connect the version 7 queue manager with one of the version 6 queue managers in both directions.
2. Create two transmission queues with the names of the target queue managers. Use queue manager aliases if you cannot use the name of the target queue manager as the transmission queue name for some reason.
3. Configure the transmission queues to trigger the sender channels.
4. If the version 6 publish/subscribe system uses streams, add the streams to the version 7 queue manager as described in Adding a stream .
5. Check the version 7 queue manager **PSMODE** is set to ENABLE.
6. Alter its **PARENT** attribute to refer to one of the version 6 queue managers.
7. Check the status of the parent-child relationship between the queue managers is active in both directions.

What to do next

Once you have completed the task, both the version 6 and version 7 queue manager share the same topic spaces. For example, you can do all the following tasks.

- Exchange publications and subscriptions between version 6 and version 7 queue managers.
- Run your existing version 6 publish/subscribe programs on the version 7 queue manager.
- View and modify the topic space on either the version 6 or version 7 queue manager.
- Write version 7 publish/subscribe applications and run them on the version 7 queue manager.
- Create new publications and subscriptions with the version 7 applications and exchange them with version 6 applications.

Combine the topic spaces of multiple clusters
Create topic spaces that span multiple clusters. Publish to a topic in one cluster and subscribe to it in another.

Before you begin

You have existing publish/subscribe clusters, and you want to propagate some cluster topics into all the clusters.

About this task

To propagate publications from one cluster to another, you need to join the clusters together in a hierarchy; see Figure 35 on page 96. The hierarchical connections propagate subscriptions and publications between the connected queue managers, and the clusters propagate cluster topics within each cluster, but not between clusters.

The combination of these two mechanisms propagates cluster topics between all the clusters. You need to repeat the cluster topic definitions in each cluster.

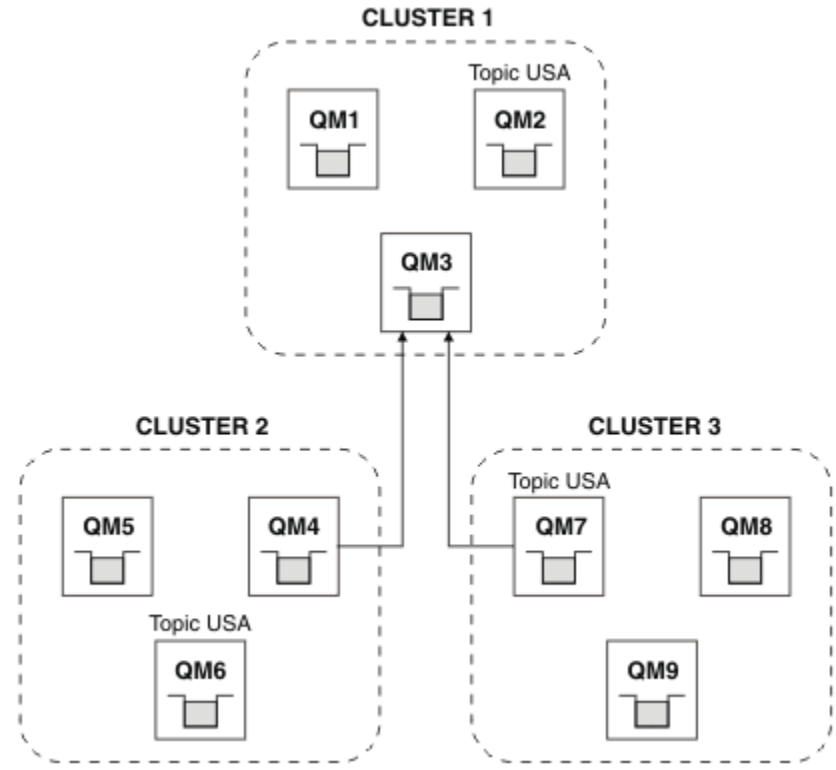


Figure 35. Connecting clusters using hierarchies

The following steps connect the clusters into a hierarchy.

Procedure

1. Create two sets of sender-receiver channels to connect QM3 and QM4, and QM3 and QM7, in both directions. You must use traditional sender-receiver channels and transmission queues, rather than a cluster, to connect a hierarchy.
2. Create three transmission queues with the names of the target queue managers. Use queue manager aliases if you cannot use the name of the target queue manager as the transmission queue name for some reason.
3. Configure the transmission queues to trigger the sender channels.

4. Check the **PSMODE** of QM3, QM4 and QM7 is set to ENABLE.
5. Alter the **PARENT** attribute of QM4 and QM7 to QM3.
6. Check the status of the parent-child relationship between the queue managers is active in both directions.
7. Create the administrative topic USA with the attribute **CLUSTER**('CLUSTER 1'), **CLUSTER**('CLUSTER 2'), and **CLUSTER**('CLUSTER 3') on each of the three cluster topic hosts in clusters 1, 2 and 3. The cluster topic host does not need to be a hierarchically connected queue manager.

What to do next

You can now publish or subscribe to the cluster topic USA in Figure 35 on page 96. The publications subscriptions flow to publishers and subscribers in all three clusters.

Suppose that you did not create USA as a cluster topic in the other clusters. If USA is only defined on QM7, then publications and subscriptions to USA are exchanged between QM7, QM8, QM9, and QM3. Publishers and subscribers running on QM7, QM8, QM9 inherit the attributes of the administrative topic USA. Publishers and subscribers on QM3 inherit the attributes of SYSTEM.BASE.TOPIC on QM3.

Combine and isolate topic spaces in multiple clusters

Isolate some topic spaces to a specific cluster, and combine other topic spaces to make them accessible in all the connected clusters.

Before you begin

Examine the topic “Combine the topic spaces of multiple clusters” on page 96. It might be sufficient for your needs, without adding an additional queue manager as a bridge.

About this task

A potential improvement on the topology shown in Figure 35 on page 96 in “Combine the topic spaces of multiple clusters” on page 96 is to isolate cluster topics that are not shared across all the clusters. Isolate clusters by creating a bridging queue manager that is not in any of the clusters; see Figure 36 on page 98. Use the bridging queue manager to filter which publications and subscriptions can flow from one cluster to another.

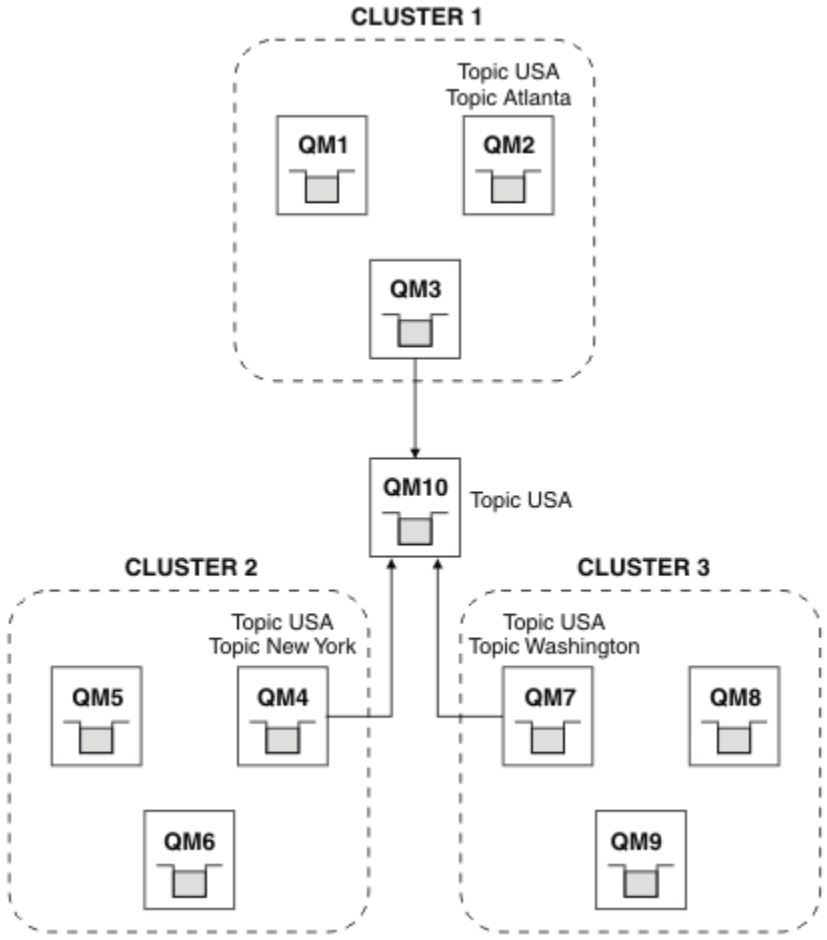


Figure 36. Bridged clusters

Use the bridge to isolate cluster topics that you do not want exposed across the bridge on the other clusters. In Figure 36 on page 98, USA is a cluster topic shared in all the clusters, and `Atlanta`, `New York` and `Washington` are cluster topics that are shared only in one cluster each.

Model your configuration using the following procedure:

Procedure

1. Modify all the `SYSTEM.BASE.TOPIC` topic objects to have **SUBSCOPE**(`QMGR`) and **PUBSCOPE**(`QMGR`) on all the queue managers.

   No topics (even cluster topics) are propagated onto other queue managers unless you explicitly set **SUBSCOPE**(`ALL`) and **PUBSCOPE**(`ALL`) on the root topic of your cluster topics.
2. Define the topics on the three cluster topic hosts that you want to be shared in each cluster with the attributes **CLUSTER**(*clustername*), **SUBSCOPE**(`ALL`) and **PUBSCOPE**(`ALL`).

   If you want some cluster topics shared between all the clusters, define the same topic in each of the clusters. Use the cluster name of each cluster as the cluster attribute.
3. For the cluster topics you want shared between all the clusters, define the topics again on the bridge queue manager (`QM10`), with the attributes **SUBSCOPE**(`ALL`), and **PUBSCOPE**(`ALL`).

Example

In the example in Figure 36 on page 98, only topics that inherit from USA propagate between all three clusters.

What to do next

Subscriptions for topics defined on the bridge queue manager with **SUBSCOPE**(ALL) and **PUBSCOPE**(ALL) are propagated between the clusters.

Subscriptions for topics defined within each cluster with attributes **CLUSTER**(*clustername*), **SUBSCOPE**(ALL) and **PUBSCOPE**(ALL) are propagated within each cluster.

Any other subscriptions are local to a queue manager.

Publish and subscribe to topic spaces in multiple clusters
Publish and subscribe to topics in multiple clusters using overlapped clusters. You can use this technique as long as the topic spaces in the clusters do not overlap.

Before you begin

Create multiple traditional clusters with some queue managers in the intersections between the clusters.

About this task

You might have chosen to overlap clusters for various different reasons.

1. You have a limited number of high availability servers, or queue managers. You decide to deploy all the cluster repositories, and cluster topic hosts to them.
2. You have existing traditional queue manager clusters that are connected using gateway queue managers. You want to deploy publish/subscribe applications to the same cluster topology.
3. You have a several self contained publish/subscribe applications. For performance reasons, it is better to keep publish/subscribe clusters small and separate from traditional clusters. You have decided to deploy the applications to different clusters. However, you also want to monitor all the publish/subscribe applications on one queue manager, as you have licensed only one copy of the monitoring application. This queue manager must have access to the publications to cluster topics in all the clusters.

By ensuring that your topics are defined in non-overlapping topic spaces, you can deploy the topics into overlapping publish/subscribe clusters, see Figure 37 on page 99. If the topic spaces overlap, then deploying to overlapping clusters leads to problems.

Because the publish/subscribe clusters overlap you can publish and subscribe to any of the topic spaces using the queue managers in the overlap.

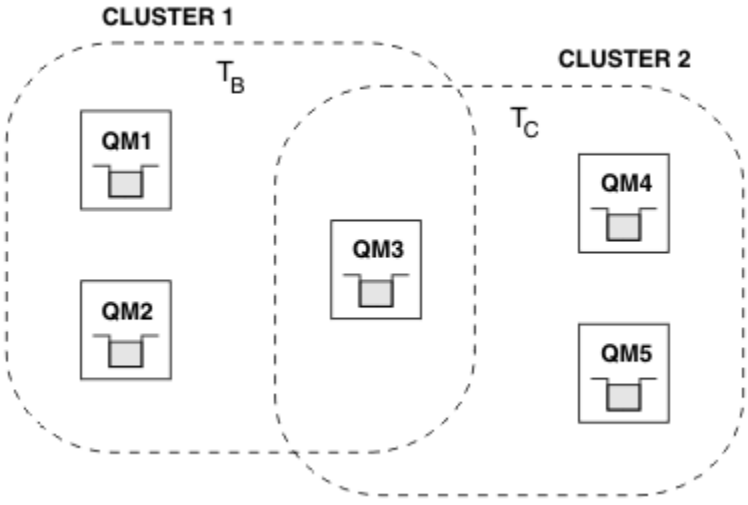


Figure 37. Overlapping clusters, non-overlapping topic spaces

Procedure

Create a means of ensuring that topic spaces do not overlap.

For example, define a unique root topic for each of the topic spaces. Make the root topics cluster topics.

a) `DEFINE TOPIC(B) TOPICSTR('B') CLUSTER('CLUSTER 1') ...`

b) `DEFINE TOPIC(C) TOPICSTR('C') CLUSTER('CLUSTER 2') ...`

Example

In Figure 37 on page 99 publishers and subscriber connected to QM3 can publish or subscribe to T_B or T_C

What to do next

Connect publishers and subscribers that use topics in both clusters to queue managers in the overlap.

Connect publishers and subscribers that must only use topics in a specific cluster to queue managers not in the overlap.

Overlapping topics
Overlapping topics occur when a publication can be associated with different topic objects, depending on the distributed publish/subscribe topology, the publication, and the subscription topic strings.

Overlaps between topics have to be considered if a topic could be resolved to more than one topic object.

Local topics in a cluster
A topic can be defined on any queue manager in a cluster. If the topic is defined locally, then it takes precedence over a cluster topic that is defined elsewhere and resolves to the same topic string.

Cluster topics in a cluster
A topic can be defined on any queue manager in a cluster. If the topic is clustered, then it is replicated to other members of the cluster. If the topic is defined as a cluster topic on another queue manager in the cluster, it is an error. An error message is written to the error log of the queue manager that has an existing cluster definition.

As a rule, define cluster topics only on one queue manager in the cluster, the "cluster topic host", to ensure there is only one definition of a cluster topic.

If you redefine a cluster topic, the change takes time to reach each queue manager. Eventually, the latest definition overrides the earlier cluster topic definitions that have been replicated to non-cluster topic hosts.

If you define a cluster topic on multiple queue managers in the cluster with different attributes, the latest definition does not override any earlier local definitions.

Wildcard subscriptions resolve to multiple topic strings

When a subscription contains wildcards, potentially different topics in a topic space can match the subscription and result in the subscription resolving to different topic objects.

For example, consider the following topic definitions in the cluster SPORTS.

```
DEFINE TOPIC(A) TOPICSTR('Football/result/#') SUBSCOPE(QMGR) CLUSTER(SPORTS)
DEFINE TOPIC(B) TOPICSTR('Football/#') SUBSCOPE(ALL) CLUSTER(SPORTS)
DEFINE TOPIC(C) TOPICSTR('Football/result/Newport/Cardiff') PUBSCOPE(ALL) SUBSCOPE(ALL)
CLUSTER(SPORTS)
DEFINE TOPIC(D) TOPICSTR('Football/matches/Newport/Cardiff') PUBSCOPE(ALL) SUBSCOPE(QMGR)
CLUSTER(SPORTS)
```

Suppose there are two queue managers QM1 and QM2 in the cluster. Topics C and D are published on QM1.

Consider what a subscriber on QM2 receives, if these subscriptions are ungrouped.

- A subscription to topic A receives nothing.
  - SUBSCOPE(QMGR), and the publication is on the other queue manager.

- A subscription to topic B receives both publications.
  - SUBSCOPE(ALL) and PUBSCOPE(ALL) in both cases.
- A subscription to topic C receives one publication.
  - SUBSCOPE(ALL) and PUBSCOPE(ALL), and a match to the publication on topic C.
- A subscription to topic D receives nothing.
  - SUBSCOPE(QMGR), and the publication is on the other queue manager.

Consider what a subscriber on QM2 receives, if these subscriptions are grouped.

- The subscriber receives one publication on topic C.
  - The matching subscription on topic A with SUBSCOPE(QMGR) is overridden by the matching subscription on topic C with SUBSCOPE(ALL). The more specific subscription wins, and the publication is received.
  - The matching subscription on topic B is rejected in favor of the matching subscription on topic C, because the subscriptions are grouped, and C is more specific. The duplicate publication is discarded.
- The subscriber receives no publication on topic D
  - The matching subscription on topic B with SUBSCOPE(ALL) is overridden by the matching subscription on topic D with SUBSCOPE(QMGR). The more specific subscription wins, and the publication is discarded.

How loop detection works

In a distributed publish/subscribe network, it is important that publications and proxy subscriptions cannot loop, as this would result in a flooded network with connected subscribers receiving multiple copies of the same original publication.

The proxy subscription aggregation system described in "Proxy subscription aggregation and publication aggregation" on page 51 does not prevent the formation of a loop, although it will prevent the perpetual looping of proxy subscriptions. Because the propagation of publications is determined by the existence of proxy subscriptions, they can enter a perpetual loop. Websphere MQ V7.0 uses the following technique to prevent publications from perpetually looping:

As publications move around a publish/subscribe topology each queue manager adds a unique fingerprint to the message header. Whenever a publish/subscribe queue manager receives a publication from another publish/subscribe queue manager, the fingerprints held in the message header are checked. If its own fingerprint is already present, the publication has fully circulated around a loop, so the queue manager discards the message, and adds an entry to the error log.

Note: Within a loop, publications are propagated in both directions around the loop, and each queue manager within the loop receives both publications before the originating queue manager discards the looped publications. This results in subscribing applications receiving duplicate copies of publications until the loop is broken.

Loop detection fingerprint format

The loop detection fingerprints are inserted into an RFH2 header or flow as part of the V7.0 protocol. An RFH2 programmer needs to understand the header and pass on the fingerprint information intact. WebSphere MessageBroker uses RFH1 headers which will not contain the fingerprint information.

```
<ibm>
  <Rfp>uuid1</Rfp>
  <Rfp>uuid2</Rfp>
  <Rfp>uuid3</Rfp>
  . . .
</ibm>
```

<ibm> is the name of the folder that holds the list of routing fingerprints containing the unique user identifier (uuid) of each queue manager that has been visited.

Every time that a message is published by a queue manager, it adds its uuid into the <ibm> folder using the <Rfp> (routing fingerprint) tag. Whenever a publication is received, WebSphere MQ uses the message properties API to iterate through the <Rfp> tags to see if that particular uuid value is present. Because of the way that the WebSphere Platform Messaging component of WebSphere MQ attaches to Websphere Message Broker through a channel and RFH2 subscription when using the queued publish/subscribe interface, WebSphere MQ also creates a fingerprint when it receives a publication by that route.

The goal is to not deliver any RFH2 to an application if it is not expecting any, simply because we have added in our fingerprint information.

Whenever an RFH2 is converted into message properties, it will also be necessary to convert the <ibm> folder; this removes the fingerprint information from the RFH2 that is passed on or delivered to applications that have used the Websphere MQ V7.0 API.

Whenever a message that has fingerprint information is delivered to an RFH1 subscriber or is passed onto Websphere Message Broker V6.0, the fingerprint information is converted to an RFH1.

When Websphere Message Broker V6.0 passes this message to an RFH2 subscriber, such as SIB, it has to convert the fingerprint information back to an RFH2 format.

JMS applications do not see the fingerprint information, because the JMS interface does not extract that information from the RFH2, and therefore does not hand it on to its applications.

The Rfp message properties are created with `propDesc.CopyOptions = MQCOPY_FORWARD` and `MQCOPY_PUBLISH`. This has implications for applications receiving and then republishing the same message. It means that such an application can continue the chain of routing fingerprints by using `PutMsgOpts.Action = MQACTP_FORWARD`, but must be coded appropriately to remove its own fingerprint from the chain. By default the application uses `PutMsgOpts.Action = MQACTP_NEW` and starts a new chain.

Retained publications in a distributed publish/subscribe topology

When using retained publications in a distributed publish/subscribe topology, it is best practice to publish only retained publications on the same topic from a single queue manager in the topology.

Otherwise, it is possible that different retained publications might be active at different queue managers for the same topic, leading to unexpected behavior. As multiple proxy subscriptions are distributed, multiple retained publications might be received.

Publish/subscribe security between queue managers

Publish/subscribe internal messages, such as proxy subscriptions and publications, are put to publish/subscribe system queues using normal channel security rules. The information and diagrams in this topic highlight the various processes and user IDs involved in the delivery of these messages.

Local access control

Access to topics for publication and subscriptions is governed by local security definitions and rules that are described in Publish/subscribe security. On z/OS, no local topic object is required to establish access control. No local topic is required for access control on other platforms either. Administrators can choose to apply access control to clustered topic objects, irrespective of whether they exist in the cluster yet.

System administrators are responsible for access control on their local system. They must trust the administrators of other members of the hierarchy or cluster collectives to be responsible for their access control policy. Because access control is defined for each separate machine it is likely to be burdensome if fine level control is needed. It might not be necessary to impose any access control, or access control might be defined on high-level objects in the topic tree. Fine level access control can be defined for each subdivision of the topic namespace.

Making a proxy subscription

Trust for an organization to connect its queue manager to your queue manager is confirmed by normal channel authentication means. If that trusted organization is also allowed to do distributed publish/subscribe, an authority check is done. The check is made when the channel puts a

message to a distributed publish/subscribe queue. For example, if a message is put to the SYSTEM.INTER.QMGR.CONTROL queue. The user ID for the queue authority check depends on the PUTAUT values of the receiving channel. For example, the user ID of the channel, MCAUSER, the message context, depending on the value and platform. For more information about channel security, see Channel security.

Proxy subscriptions are made with the user ID of the distributed publish/subscribe agent on the remote queue manager. For example, QM2 in Figure 38 on page 103. The user is then easily granted access to local topic object profiles, because that user ID is defined in the system and there are therefore no domain conflicts.

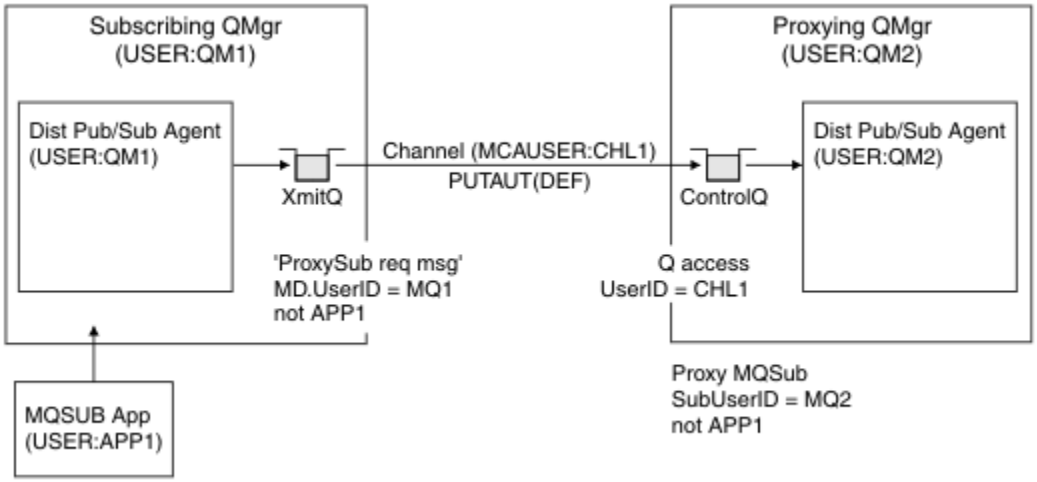


Figure 38. Proxy subscription security, making a subscription

Sending back remote publications

When a publication is created on the publishing queue manager, a copy of the publication is created for any proxy subscription. The context of the copied publication contains the context of the user ID which made the subscription; QM2 in Figure 39 on page 104. The proxy subscription is created with a destination queue that is a remote queue, so the publication message is resolved onto a transmission queue.

Trust for an organization to connect its queue manager, QM2, to another queue manager, QM1, is confirmed by normal channel authentication means. If that trusted organization is then allowed to do distributed publish/subscribe, an authority check is done when the channel puts the publication message to the distributed publish/subscribe publication queue SYSTEM.INTER.QMGR.PUBS. The user ID for the queue authority check depends on the PUTAUT value of the receiving channel (for example, the user ID of the channel, MCAUSER, message context, and others, depending on value and platform). For more information about channel security, see Channel security.

When the publication message reaches the subscribing queue manager, another MQPUT to the topic is done under the authority of that queue manager and the context with the message is replaced by the context of each of the local subscribers as they are each given the message.

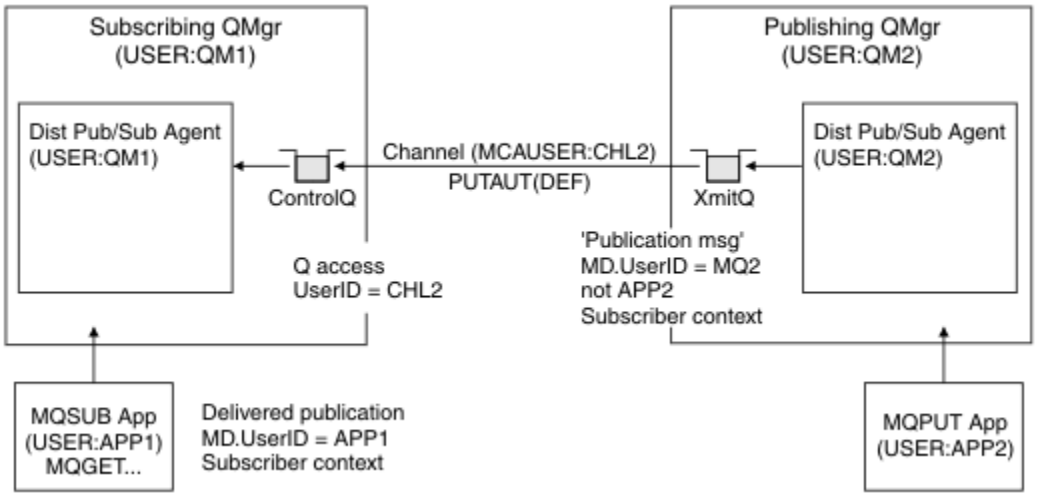


Figure 39. Proxy subscription security, forwarding publications

On a system where little has been considered regarding security, the distributed publish/subscribe processes are likely to be running under a user ID in the `mqm` group, the `MCAUSER` parameter on a channel is blank (the default), and messages are delivered to the various system queues as required. The unsecured system makes it easy to set up a proof of concept to demonstrate distributed publish/subscribe.

On a system where security is more seriously considered, these internal messages are subject to the same security controls as any message going over the channel.

If the channel is set up with a non-blank `MCAUSER` and a `PUTAUT` value specifying that `MCAUSER` must be checked, then the `MCAUSER` in question must be granted access to `SYSTEM.INTER.QMGR.*` queues. If there are multiple different remote queue managers, with channels running under different `MCAUSER` IDs, all those user IDs need to be granted access to the `SYSTEM.INTER.QMGR.*` queues. Channels running under different `MCAUSER` IDs might occur, for example, when multiple hierarchical connections are configured on a single queue manager.

If the channel is set up with a `PUTAUT` value specifying that the context of the message is used, then access to the `SYSTEM.INTER.QMGR.*` queues are checked based on the user ID inside the internal message. Because all these messages are put with the user ID of the distributed publish/subscribe agent from the queue manager that is sending the internal message, or publication message (see Figure 39 on page 104), it is not too large a set of user IDs to grant access to the various system queues (one per remote queue manager), if you want to set up your distributed publish/subscribe security in this way. It still has all the same issues that channel context security always has; that of the different user ID domains and the fact that the user ID in the message might not be defined on the receiving system. However, it is a perfectly acceptable way to run if required.

All inter-queue manager messaging for the purposes of distributed publish/subscribe runs using normal channel security.

For information about restricting publications and proxy subscriptions at the topic level, see Publish/subscribe security .

Using default user IDs with a queue manager hierarchy

If you have a hierarchy of queue managers running on different platforms and are using default user IDs, note that these default user IDs differ between platforms and might not be known on the target platform. As a result, a queue manager running on one platform rejects messages received from queue managers on other platforms with the reason code `MQRC_NOT_AUTHORIZED`.

To avoid messages being rejected, at a minimum, the following authorities need to be added to the default user IDs used on other platforms:

- *PUT *GET authority on the SYSTEM.BROKER. queues

- *PUB *SUB authority on the SYSTEM.BROKER. topics
- *ADMCRT *ADMDLT *ADMCHG authority on the SYSTEM.BROKER.CONTROL.QUEUE queue.

The default user IDs are as follows:

Platform	Default user ID
Windows	MUSR_MQADMIN **Note:** MUSR_MQADMIN is the default user ID for the first installation only. For subsequent installations, the Prepare IBM WebSphere MQ Wizard creates a user account named forMUSR_MQADMINx, where x is the next available number representing a user ID that does not exist.
UNIX and Linux systems	mqm
IBM i	QMQM
z/OS	The channel initiator address space user ID

Create and grant access to the 'qmqm' user ID if hierarchically attached to a queue manager on IBM i for Queue Managers on Windows, UNIX, Linux, and z/OS platforms.

Create and grant access to the 'mqm' user ID if hierarchically attached to a queue manager on Windows, UNIX, or Linux for Queue Managers on IBM i and z/OS platforms.

Create and grant user access to the z/OS channel initiator address space user ID if hierarchically attached to a queue manager on z/OS for Queue Managers on Windows, UNIX, Linux, and IBM i platforms.

User IDs can be case sensitive. The originating queue manager (if IBM i, Windows, UNIX, or Linux systems) forces the user ID to be all uppercase. The receiving queue manager (if Windows, UNIX or Linux systems) forces the user ID to be all lowercase. Therefore, all user IDs created on UNIX and Linux systems must be created in their lowercase form. If a message exit has been installed, forcing the user ID into uppercase or lowercase does not take place. Care must be taken to understand how the message exit processes the user ID.

To avoid potential problems with conversion of user IDs:

- On UNIX, Linux and Windows systems, ensure the user IDs are specified in lowercase.
- On IBM i and z/OS, ensure the user IDs are specified in uppercase.

Distributed publish/subscribe system queues

Four system queues are used by queue managers for publish/subscribe messaging. You need to be aware of their existence only for problem determination or capacity planning purposes.

Table 9. Publish/subscribe system queues on distributed platforms

System queue	Purpose
SYSTEM.INTER.QMGR.CONTROL	WebSphere MQ distributed publish/subscribe control queue
SYSTEM.INTER.QMGR.FANREQ	WebSphere MQ distributed publish/subscribe internal proxy subscription fan-out process input queue
SYSTEM.INTER.QMGR.PUBS	WebSphere MQ distributed publish/subscribe publications
SYSTEM.HIERARCHY.STATE	WebSphere MQ distributed publish/subscribe hierarchy relationship state

The attributes of the publish/subscribe system queues are shown in Table 10 on page 106.

Table 10. Attributes of publish/subscribe system queues	
Attribute	**Default value**
DEFPSIST	Yes
DEFSOPT	EXC
MAXMSGL	On AIX, HP-UX, Linux, IBM i, Solaris, and Windows platforms: The value of the MAXMSGL parameter of the ALTER QMGR command
MAXDEPTH	999999999
SHARE	N/A
STGCLASS	This attribute is used only on z/OS platforms

Publish/subscribe system queue errors
Errors can occur when distributed publish/subscribe queue manager queues are unavailable.

If the fan-out request queue SYSTEM.INTER.QMGR.FANREQ is unavailable, the MQSUB API receives reason codes and error messages written to the error log, on occasions where proxy subscriptions need to be delivered to directly connected queue managers.

If the hierarchy relationship state queue SYSTEM.HIERARCHY.STATE is unavailable, an error message is written to the error log and the publish/subscribe engine is put into COMPAT mode.

If any other of the SYSTEM.INTER.QMGR queues are unavailable, an error message is written to the error log, and although function is not disabled, it is likely that publish/subscribe messages will build up on queues on remote queue managers.

If the transmission queue to a parent, child or publish/subscribe cluster queue manager is unavailable:

1. The MQPUT API receives reason codes and the publications are not delivered.
2. Received inter-queue manager publications are backed out to the input queue, and subsequently re-attempted, being placed on the dead letter queue if the backout threshold is reached.
3. Proxy subscriptions are backed out to the fanout request queue, and subsequently attempted again, being placed on the dead letter queue if the backout threshold is reached; in which case the proxy subscription will not be delivered to any connected queue manager.
4. Hierarchy relationship protocol messages fail, and the connection status is marked as ERROR on the PUBSUB command.

Handling undelivered messages with the WebSphere MQ dead-letter queue handler

What is a dead-letter queue, how are messages put on it, and how do you manage it?

A *dead-letter queue* (DLQ), sometimes referred to as an *undelivered-message queue*, is a holding queue for messages that cannot be delivered to their destination queues. Every queue manager in a network should have an associated DLQ.

Messages can be put on the DLQ by queue managers, message channel agents (MCAs), and applications. All messages on the DLQ must be prefixed with a *dead-letter header* structure, MQDLH.

Messages put on the DLQ by a queue manager or a message channel agent always have an MQDLH; applications putting messages on the DLQ must supply an MQDLH. The *Reason* field of the MQDLH structure contains a reason code that identifies why the message is on the DLQ.

All WebSphere MQ environments need a routine to process messages on the DLQ regularly. WebSphere MQ supplies a default routine, called the *dead-letter queue handler* (the DLQ handler), which you invoke using the `runmqdlq` command.

Instructions for processing messages on the DLQ are supplied to the DLQ handler by means of a user-written *rules table*. That is, the DLQ handler matches messages on the DLQ against entries in the rules table; when a DLQ message matches an entry in the rules table, the DLQ handler performs the action associated with that entry.

Invoking the DLQ handler

Invoke the DLQ handler using the `runmqdlq` command. You can name the DLQ you want to process and the queue manager you want to use in two ways.

The two ways are as follows:

- As parameters to `runmqdlq` from the command prompt. For example:

```
runmqdlq ABC1.DEAD.LETTER.QUEUE ABC1.QUEUE.MANAGER <qrule.rul
```

- In the rules table. For example:

```
INPUTQ(ABC1.DEAD.LETTER.QUEUE) INPUTQM(ABC1.QUEUE.MANAGER)
```

The examples apply to the DLQ called ABC1.DEAD.LETTER.QUEUE, owned by the queue manager ABC1.QUEUE.MANAGER.

If you do not specify the DLQ or the queue manager as shown, the default queue manager for the installation is used along with the DLQ belonging to that queue manager.

The `runmqdlq` command takes its input from `stdin`; you associate the rules table with `runmqdlq` by redirecting `stdin` from the rules table.

To run the DLQ handler you must be authorized to access both the DLQ itself and any message queues to which messages on the DLQ are forwarded. For the DLQ handler to put messages on queues with the authority of the user ID in the message context, you must also be authorized to assume the identity of other users.

For more information about the `runmqdlq` command, see runmqdlq.

The sample DLQ handler, amqsdlq

In addition to the DLQ handler invoked using the `runmqdlq` command, WebSphere MQ provides the source of a sample DLQ handler, amqsdlq, with a function that is similar to that provided by `runmqdlq`.

You can customize amqsdlq to provide a DLQ handler that meets your requirements. For example, you might decide that you want a DLQ handler that can process messages without dead-letter headers. (Both the default DLQ handler and the sample, amqsdlq, process only those messages on the DLQ that begin with a dead-letter header, MQDLH. Messages that do not begin with an MQDLH are identified as being in error, and remain on the DLQ indefinitely.)

MQ_INSTALLATION_PATH represents the high-level directory in which WebSphere MQ is installed.

In WebSphere MQ for Windows, the source of amqsdlq is supplied in the directory:

```
MQ_INSTALLATION_PATH\tools\c\samples\dlq
```

and the compiled version is supplied in the directory:

```
MQ_INSTALLATION_PATH\tools\c\samples\bin
```

In WebSphere MQ for UNIX and Linux systems, the source of amqsdlq is supplied in the directory:

MQ_INSTALLATION_PATH/samp/dlq

and the compiled version is supplied in the directory:

MQ_INSTALLATION_PATH/samp/bin

The DLQ handler rules table

The DLQ handler rules table defines how the DLQ handler is to process messages that arrive on the DLQ.

There are two types of entry in a rules table:

- The first entry in the table, which is optional, contains *control data*.
- All other entries in the table are *rules* for the DLQ handler to follow. Each rule consists of a *pattern* (a set of message characteristics) that a message is matched against, and an *action* to be taken when a message on the DLQ matches the specified pattern. There must be at least one rule in a rules table.

Each entry in the rules table comprises one or more keywords.

Control data

This section describes the keywords that you can include in a control-data entry in a DLQ handler rules table.

Note:

- The vertical line (|) separates alternatives, only one of which can be specified.
- All keywords are optional.

INPUTQ (*QueueName*|'__')
The name of the DLQ you want to process:

1. Any INPUTQ value you supply as a parameter to the `runmqdlq` command overrides any INPUTQ value in the rules table.
2. If you do not specify an INPUTQ value as a parameter to the `runmqdlq` command, but you **do** specify a value in the rules table, the INPUTQ value in the rules table is used.
3. If no DLQ is specified or you specify INPUTQ(' ') in the rules table, the name of the DLQ belonging to the queue manager with the name that is supplied as a parameter to the `runmqdlq` command is used.
4. If you do not specify an INPUTQ value as a parameter to the `runmqdlq` command or as a value in the rules table, the DLQ belonging to the queue manager named on the INPUTQM keyword in the rules table is used.

INPUTQM (*QueueManagerName*|'__')
The name of the queue manager that owns the DLQ named on the INPUTQ keyword:

1. Any INPUTQM value you supply as a parameter to the `runmqdlq` command overrides any INPUTQM value in the rules table.
2. If you do not specify an INPUTQM value as a parameter to the `runmqdlq` command, the INPUTQM value in the rules table is used.
3. If no queue manager is specified or you specify INPUTQM(' ') in the rules table, the default queue manager for the installation is used.

RETRYINT (*Interval*|60)
The interval, in seconds, at which the DLQ handler should reprocess messages on the DLQ that could not be processed at the first attempt, and for which repeated attempts have been requested. By default, the retry interval is 60 seconds.

WAIT (YES|NO|*nnn*)
Whether the DLQ handler should wait for further messages to arrive on the DLQ when it detects that there are no further messages that it can process.

YES
The DLQ handler waits indefinitely.

NO
The DLQ handler ends when it detects that the DLQ is either empty or contains no messages that it can process.

nnn
The DLQ handler waits for *nnn* seconds for new work to arrive before ending, after it detects that the queue is either empty or contains no messages that it can process.

Specify WAIT (YES) for busy DLQs, and WAIT (NO) or WAIT (*nnn*) for DLQs that have a low level of activity. If the DLQ handler is allowed to terminate, invoke it again using triggering. For more information about triggering, see Starting WebSphere MQ applications using triggers .

An alternative to including control data in the rules table is to supply the names of the DLQ and its queue manager as input parameters to the `runmqdlq` command. If you specify a value both in the rules table and as input to the `runmqdlq` command, the value specified on the `runmqdlq` command takes precedence.

If you include a control-data entry in the rules table, it must be the **first** entry in the table.

Rules (patterns and actions)

A description of the pattern-matching keywords (those against which messages on the DLQ are matched), and the action keywords (those that determine how the DLQ handler is to process a matching message). An example rule is also provided.

The pattern-matching keywords

The pattern-matching keywords, which you use to specify values against which messages on the DLQ are matched, are as follows. (All pattern-matching keywords are optional):

APPLIDAT (*ApplIdentityData*|*)
The *ApplIdentityData* value specified in the message descriptor, MQMD, of the message on the DLQ.

APPLNAME (*PutApplName*|*)
The name of the application that issued the MQPUT or MQPUT1 call, as specified in the *PutApplName* field of the message descriptor, MQMD, of the message on the DLQ.

APPLTYPE (*PutApplType*|*)
The *PutApplType* value, specified in the message descriptor, MQMD, of the message on the DLQ.

DESTQ (*QueueName*|*)
The name of the message queue for which the message is destined.

DESTQM (*QueueManagerName*|*)
The name of the queue manager of the message queue for which the message is destined.

FEEDBACK (*Feedback*|*)
When the *MsgType* value is MQFB_REPORT, *Feedback* describes the nature of the report.

You can use symbolic names. For example, you can use the symbolic name MQFB_COA to identify those messages on the DLQ that need confirmation of their arrival on their destination queues.

FORMAT (*Format*|*)
The name that the sender of the message uses to describe the format of the message data.

MSGTYPE (*MsgType*|*)
The message type of the message on the DLQ.

You can use symbolic names. For example, you can use the symbolic name MQMT_REQUEST to identify those messages on the DLQ that need replies.

PERSIST (*Persistence*|*)
The persistence value of the message. (The persistence of a message determines whether it survives restarts of the queue manager.)

You can use symbolic names. For example, you can use the symbolic name MQPER_PERSISTENT to identify messages on the DLQ that are persistent.

REASON (*ReasonCode*|*)
The reason code that describes why the message was put to the DLQ.

You can use symbolic names. For example, you can use the symbolic name MQRC_Q_FULL to identify those messages placed on the DLQ because their destination queues were full.

REPLYQ (*QueueName*|*)
The name of the reply-to queue specified in the message descriptor, MQMD, of the message on the DLQ.

REPLYQM (*QueueManagerName*|*)
The name of the queue manager of the reply-to queue, as specified in the message descriptor, MQMD, of the message on the DLQ.

USERID (*UserIdentifier*|*)
The user ID of the user who originated the message on the DLQ, as specified in the message descriptor, MQMD, of the message on the DLQ.

The action keywords

The action keywords, used to describe how a matching message is to be processed, are as follows:

ACTION (DISCARD|IGNORE|RETRY|FWD)
The action to be taken for any message on the DLQ that matches the pattern defined in this rule.

DISCARD
Delete the message from the DLQ.

IGNORE
Leave the message on the DLQ.

RETRY
If the first attempt to put the message on its destination queue fails, try again. The RETRY keyword sets the number of tries made to implement an action. The RETRYINT keyword of the control data controls the interval between attempts.

FWD
Forward the message to the queue named on the FWDQ keyword.

You must specify the ACTION keyword.

FWDQ (*QueueName*|&DESTQ|&REPLYQ)
The name of the message queue to which to forward the message when ACTION (FWD) is requested.

QueueName
The name of a message queue. FWDQ(' ') is not valid.

&DESTQ
Take the queue name from the *DestQName* field in the MQDLH structure.

&REPLYQ
Take the queue name from the *ReplyToQ* field in the message descriptor, MQMD.

To avoid error messages when a rule specifying FWDQ (&REPLYQ) matches a message with a blank *ReplyToQ* field, specify REPLYQ (?*) in the message pattern.

FWDQM (*QueueManagerName*|&DESTQM|&REPLYQM|'__')
The queue manager of the queue to which to forward a message.

QueueManagerName
The name of the queue manager of the queue to which to forward a message when ACTION (FWD) is requested.

&DESTQM
Take the queue manager name from the *DestQMgrName* field in the MQDLH structure.

&REPLYQM
Take the queue manager name from the *ReplyToQMgr* field in the message descriptor, MQMD.

' '
FWDQM(' '), which is the default value, identifies the local queue manager.

HEADER (YES|NO)
Whether the MQDLH should remain on a message for which ACTION (FWD) is requested. By default, the MQDLH remains on the message. The HEADER keyword is not valid for actions other than FWD.

PUTAUT (DEF|CTX)
The authority with which messages should be put by the DLQ handler:

DEF
Put messages with the authority of the DLQ handler itself.

CTX
Put the messages with the authority of the user ID in the message context. If you specify PUTAUT (CTX), you must be authorized to assume the identity of other users.

RETRY (*RetryCount*|1)
The number of times, in the range 1 - 999,999,999, to try an action (at the interval specified on the RETRYINT keyword of the control data). The count of attempts made by the DLQ handler to implement any particular rule is specific to the current instance of the DLQ handler; the count does not persist across restarts. If the DLQ handler is restarted, the count of attempts made to apply a rule is reset to zero.

Example rule

Here is an example rule from a DLQ handler rules table:

```
PERSIST(MQPER_PERSISTENT) REASON (MQRC_PUT_INHIBITED) +
 ACTION (RETRY) RETRY (3)
```

This rule instructs the DLQ handler to make three attempts to deliver to its destination queue any persistent message that was put on the DLQ because MQPUT and MQPUT1 were inhibited.

All keywords that you can use on a rule are described in the rest of this section. Note the following:

- The default value for a keyword, if any, is underlined. For most keywords, the default value is * (asterisk), which matches any value.
- The vertical line (|) separates alternatives, only one of which can be specified.
- All keywords except ACTION are optional.

Rules table conventions

The syntax, structure and contents of the DLQ handler rules table must adhere to these conventions.

The rules table must adhere to the following conventions:

- A rules table must contain at least one rule.
- Keywords can occur in any order.
- A keyword can be included only once in any rule.
- Keywords are not case-sensitive.
- A keyword and its parameter value must be separated from other keywords by at least one blank or comma.
- There can be any number of blanks at the beginning or end of a rule, and between keywords, punctuation, and values.
- Each rule must begin on a new line.
- On Windows systems, the last rule in the table must end with a carriage return/line feed character. You can achieve this by ensuring that you press the Enter key at the end of the rule, so that the last line of the table is a blank line.
- For reasons of portability, the significant length of a line must not be greater than 72 characters.
- Use the plus sign (+) as the last nonblank character on a line to indicate that the rule continues from the first nonblank character in the next line. Use the minus sign (-) as the last nonblank character on a

line to indicate that the rule continues from the start of the next line. Continuation characters can occur within keywords and parameters.

For example:

```
APPLNAME('ABC+
  D')
```

results in 'ABCD', and

```
APPLNAME('ABC-
  D')
```

results in 'ABC D'.

- Comment lines, which begin with an asterisk (*), can occur anywhere in the rules table.
- Blank lines are ignored.
- Each entry in the DLQ handler rules table comprises one or more keywords and their associated parameters. The parameters must follow these syntax rules:
  - Each parameter value must include at least one significant character. The delimiting single quotation marks in values that are enclosed in quotation marks are not considered to be significant. For example, these parameters are valid:

`FORMAT('ABC')`	3 significant characters
`FORMAT(ABC)`	3 significant characters
`FORMAT('A')`	1 significant character
`FORMAT(A)`	1 significant character
`FORMAT(' ')`	1 significant character

    These parameters are invalid because they contain no significant characters:

    ```
    FORMAT('')
    FORMAT( )
    FORMAT()
    FORMAT
    ```

  - Wildcard characters are supported. You can use the question mark (?) instead of any single character, except a trailing blank; you can use the asterisk (*) instead of zero or more adjacent characters. The asterisk (*) and the question mark (?) are ***always*** interpreted as wildcard characters in parameter values.
  - Wildcard characters cannot be included in the parameters of these keywords: ACTION, HEADER, RETRY, FWDQ, FWDQM, and PUTAUT.
  - Trailing blanks in parameter values, and in the corresponding fields in the message on the DLQ, are not significant when performing wildcard matches. However, leading and embedded blanks within strings that are enclosed in single quotation marks are significant to wildcard matches.
  - Numeric parameters cannot include the question mark (?) wildcard character. You can use the asterisk (*) instead of an entire numeric parameter, but not as part of a numeric parameter. For example, these are valid numeric parameters:

`MSGTYPE(2)`	Only reply messages are eligible
`MSGTYPE(*)`	Any message type is eligible
`MSGTYPE('*')`	Any message type is eligible

However, MSGTYPE('2*') is not valid, because it includes an asterisk (*) as part of a numeric parameter.

- Numeric parameters must be in the range 0-999 999 999. If the parameter value is in this range, it is accepted, even if it is not currently valid in the field to which the keyword relates. You can use symbolic names for numeric parameters.
- If a string value is shorter than the field in the MQDLH or MQMD to which the keyword relates, the value is padded with blanks to the length of the field. If the value, excluding asterisks, is longer than the field, an error is diagnosed. For example, these are all valid string values for an 8 character field:

  ```
  'ABCDEFGH'          8 characters
  'A*C*E*G*I'         5 characters excluding asterisks
  '*A*C*E*G*I*K*M*O   8 characters excluding asterisks
  *'
  ```

- Enclose strings that contain blanks, lowercase characters, or special characters other than period (.), forward slash (?), underscore (_), and percent sign (%) in single quotation marks. Lowercase characters not enclosed in single quotation marks are folded to uppercase. If the string includes a quotation, use two single quotation marks to denote both the beginning and the end of the quotation. When the length of the string is calculated, each occurrence of double quotation marks is counted as a single character.

How the rules table is processed

The DLQ handler searches the rules table for a rule where the pattern matches a message on the DLQ.

The search begins with the first rule in the table, and continues sequentially through the table. When the DLQ handler finds a rule with a matching pattern, it takes the action from that rule. The DLQ handler increments the retry count for a rule by 1 whenever it applies that rule. If the first try fails, the DLQ handler tries again until the number of tries matches the number specified on the RETRY keyword. If all attempts fail, the DLQ handler searches for the next matching rule in the table.

This process is repeated for subsequent matching rules until an action is successful. When each matching rule has been attempted the number of times specified on its RETRY keyword, and all attempts have failed, ACTION (IGNORE) is assumed. ACTION (IGNORE) is also assumed if no matching rule is found.

Note:

1. Matching rule patterns are sought only for messages on the DLQ that begin with an MQDLH. Messages that do not begin with an MQDLH are reported periodically as being in error, and remain on the DLQ indefinitely.
2. All pattern keywords can be allowed to default, such that a rule can consist of an action only. Note, however, that action-only rules are applied to all messages on the queue that have MQDLHs and that have not already been processed in accordance with other rules in the table.
3. The rules table is validated when the DLQ handler starts, and errors are flagged at that time. You can make changes to the rules table at any time, but those changes do not come into effect until the DLQ handler restarts.
4. The DLQ handler does not alter the content of messages, the MQDLH, or the message descriptor. The DLQ handler always puts messages to other queues with the message option MQPMO_PASS_ALL_CONTEXT.
5. Consecutive syntax errors in the rules table might not be recognized because the rules table is designed to eliminate the generation of repetitive errors during validation.
6. The DLQ handler opens the DLQ with the MQOO_INPUT_AS_Q_DEF option.
7. Multiple instances of the DLQ handler can run concurrently against the same queue, using the same rules table. However, it is more usual for there to be a one-to-one relationship between a DLQ and a DLQ handler.

Ensuring that all DLQ messages are processed

The DLQ handler keeps a record of all messages on the DLQ that have been seen but not removed.

If you use the DLQ handler as a filter to extract a small subset of the messages from the DLQ, the DLQ handler still has to keep a record of those messages on the DLQ that it did not process. Also, the DLQ handler cannot guarantee that new messages arriving on the DLQ are seen, even if the DLQ is defined as first-in-first-out (FIFO). If the queue is not empty, the DLQ is periodically re-scanned to check all messages.

For these reasons, try to ensure that the DLQ contains as few messages as possible; if messages that cannot be discarded or forwarded to other queues (for whatever reason) are allowed to accumulate on the queue, the workload of the DLQ handler increases and the DLQ itself can fill up.

You can take specific measures to enable the DLQ handler to empty the DLQ. For example, try not to use ACTION (IGNORE), which leaves messages on the DLQ. (Remember that ACTION (IGNORE) is assumed for messages that are not explicitly addressed by other rules in the table.) Instead, for those messages that you would otherwise ignore, use an action that moves the messages to another queue. For example:

```
ACTION (FWD) FWDQ (IGNORED.DEAD.QUEUE) HEADER (YES)
```

Similarly, make the final rule in the table a catchall to process messages that have not been addressed by earlier rules in the table. For example, the final rule in the table could be something like this:

```
ACTION (FWD) FWDQ (REALLY.DEAD.QUEUE) HEADER (YES)
```

This forwards messages that fall through to the final rule in the table to the queue `REALLY.DEAD.QUEUE`, where they can be processed manually. If you do not have such a rule, messages are likely to remain on the DLQ indefinitely.

An example DLQ handler rules table

An example rules table for the runmqdlq command, containing a single control-data entry and several rules.

```
*********************************************************************
*        An example rules table for the runmqdlq command            *
*********************************************************************
* Control data entry
* ------------------
* If no queue manager name is supplied as an explicit parameter to
* runmqdlq, use the default queue manager for the machine.
* If no queue name is supplied as an explicit parameter to runmqdlq,
* use the DLQ defined for the local queue manager.
*
inputqm(' ')  inputq(' ')

* Rules
* -----
* We include rules with ACTION (RETRY) first to try to
* deliver the message to the intended destination.
* If a message is placed on the DLQ because its destination
* queue is full, attempt to forward the message to its
* destination queue. Make 5 attempts at approximately
* 60-second intervals (the default value for RETRYINT).

REASON(MQRC_Q_FULL) ACTION(RETRY) RETRY(5)

* If a message is placed on the DLQ because of a put inhibited
* condition, attempt to forward the message to its
* destination queue. Make 5 attempts at approximately
* 60-second intervals (the default value for RETRYINT).

REASON(MQRC_PUT_INHIBITED) ACTION(RETRY) RETRY(5)

* The AAAA corporation are always sending messages with incorrect
* addresses. When we find a request from the AAAA corporation,
* we return it to the DLQ (DEADQ) of the reply-to queue manager
* (&REPLYQM).
```

```
* The AAAA DLQ handler attempts to redirect the message.

MSGTYPE(MQMT_REQUEST) REPLYQM(AAAA.*) +
  ACTION(FWD) FWDQ(DEADQ) FWDQM(&REPLYQM)

* The BBBB corporation never do things by half measures. If
* the queue manager BBBB.1 is unavailable, try to
* send the message to BBBB.2

DESTQM(bbbb.1) +
  action(fwd) fwdq(&DESTQ) fwdqm(bbbb.2) header(no)

* The CCCC corporation considers itself very security
* conscious, and believes that none of its messages
* will ever end up on one of our DLQs.
* Whenever we see a message from a CCCC queue manager on our
* DLQ, we send it to a special destination in the CCCC organization
* where the problem is investigated.

REPLYQM(CCCC.*) +
  ACTION(FWD) FWDQ(ALARM) FWDQM(CCCC.SYSTEM)
```

```
* Messages that are not persistent run the risk of being
* lost when a queue manager terminates. If an application
* is sending nonpersistent messages, it should be able
* to cope with the message being lost, so we can afford to
* discard the message.   PERSIST(MQPER_NOT_PERSISTENT) ACTION(DISCARD)
* For performance and efficiency reasons, we like to keep
* the number of messages on the DLQ small.
* If we receive a message that has not been processed by
* an earlier rule in the table, we assume that it
* requires manual intervention to resolve the problem.
* Some problems are best solved at the node where the
* problem was detected, and others are best solved where
* the message originated. We don't have the message origin,
* but we can use the REPLYQM to identify a node that has
* some interest in this message.
* Attempt to put the message onto a manual intervention
* queue at the appropriate node. If this fails,
* put the message on the manual intervention queue at
* this node.

REPLYQM('?*') +
  ACTION(FWD) FWDQ(DEADQ.MANUAL.INTERVENTION) FWDQM(&REPLYQM)

ACTION(FWD) FWDQ(DEADQ.MANUAL.INTERVENTION)
```

Multiple installations

On UNIX, Linux, and Windows, it is possible to have more than one copy of IBM WebSphere MQ on a system.

You can choose where each copy of IBM WebSphere MQ is installed, but each copy must be in a separate installation location. A maximum of 128 installations of IBM WebSphere MQ can exist on a single machine at a time. One installation can be an installation of IBM WebSphere MQ Version 7.0.1 at fix pack 6, or later. You now have a choice:

- Keep the simplicity of maintaining and managing a single installation of IBM WebSphere MQ on a machine.
- Take advantage of the flexibility offered by enabling multiple IBM WebSphere MQ installations.

Before you install multiple copies of IBM WebSphere MQ, you must make several decisions:

- Will you have a copy of IBM WebSphere MQ Version 7.0.1 on the system?

  When IBM WebSphere MQ Version 7.0.1 at fix pack 6, or later, is installed on the system, there are a number of restrictions to consider:

  – On UNIX and Linux systems, IBM WebSphere MQ Version 7.0.1 must be installed in the default location.

  - IBM WebSphere MQ Version 7.0.1 must be the first installation on a system. You cannot install IBM WebSphere MQ Version 7.0.1 after installing version 7.1, or later. If you uninstall version 7.0.1, it cannot be reinstalled while a later version of WebSphere MQ is installed.
  - IBM WebSphere MQ Version 7.0.1 is automatically the primary installation. You cannot select another installation as the primary installation while IBM WebSphere MQ Version 7.0.1 is installed.
- Where will you install each copy of IBM WebSphere MQ?

  You can choose the installation location for your installations at version 7.1, or later. For more information, see Choosing an installation location.
- Do you need a primary installation?

  A primary installation is an installation to which system-wide locations refer. For more information, see “Choosing a primary installation” on page 117.
- How will your applications connect?

  You need to consider how your applications locate the appropriate IBM WebSphere MQ libraries. For more information, see Connecting applications in a multiple installation environment, and Connecting .NET applications in a multiple installation environment.
- Do your existing exits need changing?

  If IBM WebSphere MQ is not installed in the default location, your exits need to be updated. For more information, see Writing and compiling exits and installable services .
- Which queue manager will be associated with which installation?

  Each queue manager is associated with a particular installation. The installation that a queue manager is associated with limits that queue manager so that it can be administered only by commands from that installation. For more information, see Associating a queue manager with an installation.
- How will you set up your environment to work with each installation?

  With multiple installations on a system, you need to consider how you will work with particular installations, and how you will issue commands from that installation. You can either specify the full path to the command, or you can use the **setmqenv** or **crtmqenv** command to set environment variables. Setting the environment variables allows you to omit the path to the commands for that installation. For more information, see setmqenv, and crtmqenv.

When you have answered these questions, you can install IBM WebSphere MQ using the steps provided in Installing IBM WebSphere MQ.

If you have existing installations of IBM WebSphere MQ and you want to use the multiple installation capability to migrate from one version of IBM WebSphere MQ to another version, see Multi-installation queue manager coexistence on UNIX, Linux, and Windows .

The IBM message service client for .NET support pack and multiple installations

For multiple version support, the *Java and .NET Messaging and Web Services* feature must be installed with the IBM WebSphere MQ product. This feature contains all the functionality included in the *IBM Message Service Client for .NET* support pack (IA9H). If the support pack is installed on the system, multiple versions are not supported. You must uninstall the support pack before installing IBM WebSphere MQ. For more information about installing the .NET feature, see Installing WebSphere MQ classes for .NET.

Related concepts

UNIX, Linux, and Windows: Side-by-side migration from version 7.0.1 to version 7.5

UNIX, Linux, and Windows: Multi-stage migration from version 7.0.1 to version 7.5

Related tasks

Configuring multiple installations

Finding installations of WebSphere MQ on a system

Choosing a primary installation

On systems that support multiple installations of IBM WebSphere MQ (UNIX, Linux, and Windows), the primary installation is the one to which IBM WebSphere MQ system-wide locations refer. Having a primary installation is optional, but convenient.

Before IBM WebSphere MQ Version 7.1, only one instance of the product could be installed at any time. On Windows systems, several global environment variables were set to point to that installation. On UNIX and Linux systems, symbolic links were added to `/usr/lib`, `/usr/bin`, and `/usr/include`, also pointing at that single installation.

From Version 7.1, you can install multiple versions of IBM WebSphere MQ on UNIX, Linux, and Windows. It is possible to have more than one installation of IBM WebSphere MQ on one of these systems at any time and, optionally, to configure one of these installations as the primary installation. Environment variables and symbolic links pointing to a single installation are less meaningful when multiple versions exist. However, some functions require these system-wide locations to work. For example, custom user scripts for administering IBM WebSphere MQ, and third party products. These functions work only on the primary installation.

On UNIX and Linux systems, if you set an installation as the primary installation, symbolic links to the external libraries and control commands of that installation are added into `/usr/lib`, and `/usr/bin`. If you do not have a primary installation, the symbolic links are not created. For a list of the symbolic links that are made to the primary installation, see External library and control command links to primary installation on UNIX and Linux.

On Windows systems, the global environmental variables point to the directories into which the primary installation was installed. These environment variables are used to locate IBM WebSphere MQ libraries, control commands, and header files. Additionally, on Windows systems, some features of the operating system require the central registration of interface libraries that are then loaded into a single process. With multiple versions of IBM WebSphere MQ, there would be conflicting sets of IBM WebSphere MQ libraries. The features would try to load these conflicting sets of libraries into a single process. Therefore, such features can be used only with the primary installation. For details about some of the features that are limited to use with the primary installation, see Features that can be used only with the primary installation on Windows.

If you have an installation of IBM WebSphere MQ Version 7.0.1 on the system, this installation is automatically the primary installation. The primary installation cannot be changed while Version 7.0.1 is installed. If all the installations on the system are at Version 7.1, or later, you can choose whether to have a primary installation. Consider the options in Table 11 on page 117.

Table 11. Primary installation options.

This table shows the valid installation configurations for primary installations. With a single Version 7.1, or later, it can be either primary, or non-primary. With multiple installations, one at Version 7.0.1 and one or more at Version 7.1, or later,Version 7.0.1 must be the primary, and the other installations must be non-primary. With multiple installations at Version 7.1, or later, one installation can be primary, or all installations can be non-primary.

Options	**Valid installation configurations**		**More information**
	Primary	Non-primary	

Table 11. Primary installation options.

This table shows the valid installation configurations for primary installations. With a single Version 7.1, or later, it can be either primary, or non-primary. With multiple installations, one at Version 7.0.1 and one or more at Version 7.1, or later,Version 7.0.1 must be the primary, and the other installations must be non-primary. With multiple installations at Version 7.1, or later, one installation can be primary, or all installations can be non-primary.

(continued)

Options	**Valid installation configurations**		**More information**
Single installation of Version 7.1, or later.	Version 7.1, or later.	None	If you want to continue working with a single installation in the same way as previous releases, configure your installation as the primary installation. For information about this option, see Single installation of IBM WebSphere MQ Version 7.1, or later, configured as the primary installation
	None	Version 7.1, or later.	If you want to continue working with a single installation, but do not want symbolic links or global environment variables created for you, configure your installation as non-primary. For information about the implications of this option, see Single installation of IBM WebSphere MQ Version 7.1, or later, configured as non-primary
Multiple installations: Version 7.0.1 and Version 7.1, or later.	Version 7.0.1	Version 7.1, or later.	If you want to have multiple installations of IBM WebSphere MQ, with one at version 7.0.1, the version 7.0.1 installation is automatically the primary installation. While IBM WebSphere MQ version 7.0.1 is installed, you cannot change which installation is the primary installation. For information about this option and its implications, see Multiple installations of IBM WebSphere MQ, one at Version 7.0.1
Multiple installations: Version 7.1, or later.	Version 7.1, or later.	Version 7.1, or later.	If you want to have multiple installations of WebSphere MQ at version 7.1 or greater, you can choose whether to make one of the installations primary. For information about this option, see Multiple installations of IBM WebSphere MQ Version 7.1, or later
	None	Version 7.1, or later.	

Related concepts

Single installation of WebSphere MQ Version 7.1, or later, configured as the primary installation

Single installation of WebSphere MQ Version 7.1, or later, configured as non-primary

Multiple installations of WebSphere MQ Version 7.1, or later

Multiple installations of WebSphere MQ, one at version 7.0.1

Related tasks

Changing the primary installation

Choosing an installation location

Planning your installation

Choosing an installation name

Single installation of IBM WebSphere MQ Version 7.1, or later, configured as the primary installation

Marking an IBM WebSphere MQ installation as primary adds symbolic links, or global environment variables to the system so that the IBM WebSphere MQ commands and libraries used by applications are automatically available with minimum system setup required.

You decide where to install IBM WebSphere MQ.

Where possible, configure applications and scripts to use the system search path to find the IBM WebSphere MQ control commands or IBM WebSphere MQ libraries. This configuration of applications and scripts provides maximum flexibility for undertaking future tasks such as migrating to the next release of IBM WebSphere MQ, or installing a second installation. For more information about options for connecting your applications, see Connecting applications in a multiple installation environment.

On Windows, the first installation is automatically configured as the primary installation. On UNIX and Linux platforms, the first installation onto a system must be manually configured to be the primary installation. Set the primary installation using the **setmqinst** command. For more information, see Uninstalling, upgrading, and maintaining the primary installation.

Related tasks

Changing the primary installation

Choosing an installation location

Planning your installation

Choosing an installation name

Single installation of IBM WebSphere MQ Version 7.1, or later, configured as non-primary

If you install IBM WebSphere MQ Version 7.1, or later, as non-primary you might have to configure a library path for applications to load IBM WebSphere MQ libraries. On Windows, some product capabilities are available only when IBM WebSphere MQ is configured as primary.

UNIX and Linux systems

The implications of running a non-primary installation on UNIX and Linux are:

- Applications that locate their IBM WebSphere MQ libraries using an embedded library path, for example, RPATH, cannot find those libraries if the following conditions are true:
  - IBM WebSphere MQ is installed in a different directory from the directory specified in the RPATH
  - There are no symbolic links in `/usr`
- Where applications locate their libraries using an external library path, for example, LD_LIBRARY_PATH, you must configure the external library path to include the *MQ_INSTALLATION_PATH*`/lib` or *MQ_INSTALLATION_PATH*`/lib64` directory. The **setmqenv** and **crtmqenv** commands can configure a number of environment variables in the current shell, including the external library path.
- Most IBM WebSphere MQ processes run as setuid/setgid. As a result, when loading user exits they ignore the external library path. User exits that reference IBM WebSphere MQ libraries can find those libraries only if they are found in the library path embedded within them. They would be resolved if there were a symbolic link in `/usr`. User exits that are intended to be run on IBM WebSphere MQ Version 7.1, or later can now be built so that they do not refer to IBM WebSphere MQ libraries at all. Instead they rely on IBM WebSphere MQ to pass in function pointers to the IBM WebSphere MQ functions that the exit can then use. For more information, see Writing and compiling exits and installable services .

For more information about options for connecting your applications, see Connecting applications in a multiple installation environment.

On UNIX and Linux platforms, the first installation onto a system is not automatically configured as the primary installation. However, a single symbolic link is included in `/usr/bin` to locate the **dspmqver** command. If you do not want any symbolic links, you must remove this link using the following command:

```
setmqinst -x -p MQ_INSTALLATION_PATH
```

Windows systems

The implications of running a non-primary installation on Windows are:

- Applications normally find their libraries using the external library path, PATH. There is no concept of an embedded library path or explicit library location. If the installation is non-primary, the global PATH environment variable does not contain the IBM WebSphere MQ installation directory. For applications to find IBM WebSphere MQ libraries, update the PATH environment variable to reference the IBM WebSphere MQ installation directory. The **setmqenv** and **crtmqenv** commands can configure a number of environment variables in the current shell, including the external library path.
- Some product capabilities are available only when an installation is configured as the primary installation; see Features that can be used only with the primary installation on Windows.

By default, on Windows, the first installation is automatically configured as primary. You must manually deselect it as the primary installation.

Related tasks

Changing the primary installation

Choosing an installation location

Planning your installation

Choosing an installation name

Related reference

setmqenv

crtmqenv

Multiple installations of IBM WebSphere MQ Version 7.1, or later

You can choose to have one of the IBM WebSphere MQ Version 7.1 or later installations configured as the primary installation. Your choice depends on how applications locate libraries.

The IBM WebSphere MQ libraries, such as mqm, which ship with IBM WebSphere MQ Version 7.1 automatically use libraries of the level required by the queue manager to which they are connecting. This means that provided an application locates its IBM WebSphere MQ libraries from a IBM WebSphere MQ Version 7.1 installation, it can connect to any queue manager on that system. Having one IBM WebSphere MQ Version 7.1 installation configured as primary ensures that if the application finds its IBM WebSphere MQ interface library, the application can connect to any queue manager.

For more information about connecting applications in a multiple installation environment, see Connecting applications in a multiple installation environment.

The primary installation is not automatically changed when you uninstall the primary installation. If you want another installation to be the primary installation, you must manually set the primary installation using the **setmqinst** command. For more information, see Uninstalling, upgrading, and maintaining the primary installation.

Related concepts

Multiple installations

Related tasks

Changing the primary installation

Choosing an installation location

Planning your installation

Choosing an installation name

Multiple installations of IBM WebSphere MQ, one at Version 7.0.1

IBM WebSphere MQ Version 7.1, or later, can coexist with IBM WebSphere MQ Version 7.0.1 with some limitations.

- On UNIX and Linux systems, Version 7.0.1 can be installed only in a fixed, default location, so you cannot install Version 7.1, or later, in that default location.
- IBM WebSphere MQ Version 7.0.1 is automatically configured as the primary installation. On UNIX and Linux systems, symbolic links are automatically created to the appropriate IBM WebSphere MQ directories. On Windows, everything that the product provided is registered globally. IBM WebSphere MQ Version 7.0.1 must be installed in this way to work. So where IBM WebSphere MQ Version 7.0.1 is installed, a IBM WebSphere MQ Version 7.1, or later, installation cannot be made primary.

The libraries from IBM WebSphere MQ Version 7.1, or later, can work with any queue manager running under IBM WebSphere MQ Version 7.0.1, or later. If an application needs to connect to queue managers running under Version 7.0.1 as well as later versions, it can continue to operate normally if the following conditions are true:

- It locates IBM WebSphere MQ Version 7.1, or later, libraries at run time.
- It uses only functions available in Version 7.0.1.

For more information about connecting applications in a multiple installation environment, see Connecting applications in a multiple installation environment.

The primary installation is not automatically changed when you uninstall IBM WebSphere MQ Version 7.0.1. If you want another installation to be the primary installation, you must manually set the primary installation using the **setmqinst** command. For more information, see Uninstalling, upgrading, and maintaining the primary installation.

Related concepts

Multiple installations

Related tasks

Choosing an installation location

Planning your installation

Choosing an installation name

Planning your storage and performance requirements

You must set realistic and achievable storage, and performance goals for your IBM WebSphere MQ system. Use the links to find out about factors that affect storage and performance on your platform.

The requirements vary depending on the systems that you are using IBM WebSphere MQ on, and what components you want to use.

For the latest information about supported hardware and software environments, see the System Requirements for IBM WebSphere MQ Web site:

```
www.ibm.com/software/integration/wmq/requirements/
```

IBM WebSphere MQ stores queue manager data in the file system. Use the following links to find out about planning and configuring directory structures for use with IBM WebSphere MQ:

- “Planning file system support” on page 123
- “Requirements for shared file systems” on page 124
- “Sharing IBM WebSphere MQ files” on page 132
- “Directory structure on UNIX and Linux systems” on page 135
- “Directory structure on Windows systems” on page 144

Use the following links for information about system resources, shared memory, and process priority on UNIX and Linux:

- “IBM WebSphere MQ and UNIX System V IPC resources” on page 147
- “Shared memory on AIX” on page 148
- “ WebSphere MQ and UNIX Process Priority” on page 148

Related concepts

“Planning” on page 5
When planning your IBM WebSphere MQ environment, you must consider the IBM WebSphere MQ architecture that you want to configure, resource requirements, the need for logging, and backup facilities. Use the links in this topic to plan the environment where IBM WebSphere MQ runs.

“Designing an IBM WebSphere MQ architecture” on page 14
Find out about the different architectures that IBM WebSphere MQ supports for point-to-point and publish/subscribe messaging styles.

Hardware and software requirements on UNIX and Linux

Hardware and software requirements on Windows

Disk space requirements

The storage requirements for WebSphere MQ depend on which components you install, and how much working space you need.

Disk storage is required for the optional components you choose to install, including any prerequisite components they require. The total storage requirement also depends on the number of queues that you use, the number and size of the messages on the queues, and whether the messages are persistent. You also require archiving capacity on disk, tape, or other media, as well as space for your own application programs.

The following table shows the approximate disk space required when you install various combinations of the product on different platforms. (Values are rounded up to the nearest 5 MB, where a MB is 1,048,576 bytes.)

Platform	Client installation [1]	Server installation [2]	WebSphere MQ MFT installation [3]	Full installation [4]
AIX	145 MB	190 MB	705 MB	915 MB
HP-UX	225 MB	310 MB	1075 MB	1340 MB
IBM i	215 MB	450 MB	80 MB	655 MB
Linux for System x (32 bit)	85 MB	N/A	N/A	120 MB
Linux for System x (64 bit)	125 MB	170 MB	575 MB	935 MB
Linux on POWER Systems - Big Endian	130 MB	170 MB	565 MB	715 MB
Solaris x86-64, AMD64, EM64T, and compatible processors	105 MB [5]	150 MB [5]	695 MB	860 MB
Solaris SPARC	105 MB [5]	150 MB [5]	680 MB	820 MB
Windows (32 bit install) [6]	390 MB	N/A	N/A	475 MB
Windows (64 bit install) [6]	445 MB	555 MB	710 MB	1005 MB

Usage notes

1. A client installation includes the following components:
   - Runtime
   - Client
2. A server installation includes the following components:
   - Runtime
   - Server
3. An IBM WebSphere MQ Managed File Transfer installation includes the following components:
   - IBM WebSphere MQ Managed File Transfer Service, Logger, Agent, Tools, and Base components
   - Runtime
   - Server
   - Java
   - JRE
4. A full installation includes all available components.
5. Solaris On Solaris platforms you must install silently to get this combination of components.
6. Windows Not all the components listed here are installable features on Windows systems; their functionality is sometimes included in other features. See WebSphere MQ features for Windows systems.

Related tasks

Choosing what to install

Planning file system support

Queue manager data is stored in the file system. A queue manager makes use of file system locking to prevent multiple instances of a multi-instance queue manager being active at the same time.

Shared file systems

Shared file systems enable multiple systems to access the same physical storage device concurrently. Corruption would occur if multiple systems accessed the same physical storage device directly without some means of enforcing locking and concurrency control. Operating systems provide local file systems with locking and concurrency control for local processes; network file systems provide locking and concurrency control for distributed systems.

Historically, networked file systems have not performed fast enough, or provided sufficient locking and concurrency control, to meet the requirements for logging messages. Today, networked file systems can provide good performance, and implementations of reliable network file system protocols such as *RFC 3530, Network File System (NFS) version 4 protocol*, meet the requirements for logging messages reliably.

Shared file systems and WebSphere MQ

Queue manager data for a multi-instance queue manager is stored in a shared network file system. On Microsoft Windows, UNIX and Linux systems, the queue manager's data files and log files must be placed in shared network file system.

Prior to release v7.0.1, WebSphere MQ does not support queue manager data stored on networked storage accessed as a shared file system. If queue manager data is placed on shared networked storage, then you need to ensure the queue manager data is not accessed by another instance of the queue manager running at the same time.

From v7.0.1 onwards, WebSphere MQ uses locking to prevent multiple instances of the same multi-instance queue manager being active at the same time. The same locking also ensures that two separate

queue managers cannot inadvertently use the same set of queue manager data files. Only one instance of a queue manager can have its lock at a time. Consequently, WebSphere MQ does support queue manager data stored on networked storage accessed as a shared file system.

Because not all the locking protocols of network file systems are robust, and because a file system might be configured for performance rather than data integrity, you must run the **amqmfsck** command to test whether a network file system will control access to queue manager data and logs correctly. This command applies only UNIX and IBM i systems. On Microsoft Windows, there is only one supported network file system and the **amqmfsck** command is not required.

Related tasks

"Verifying shared file system behavior" on page 125
Run **amqmfsck** to check whether a shared file system on UNIX systems meets the requirements for storing the queue manager data of a multi-instance queue manager. Run the IBM WebSphere MQ MQI client sample program **amqsfhac** in parallel with **amqmfsck** to demonstrate that a queue manager maintains message integrity during a failure.

Requirements for shared file systems

Shared files systems must provide data write integrity, guaranteed exclusive access to files and release locks on failure to work reliably with IBM WebSphere MQ.

Requirements that a shared file system must meet

There are three fundamental requirements that a shared file system must meet to log messages reliably:

1. Data write integrity

   Data write integrity is sometimes called *Write through to disk on flush*. The queue manager must be able to synchronize with data being successfully committed to the physical device. In a transactional system, you need to be sure that some writes have been safely committed before continuing with other processing.

   More specifically, IBM WebSphere MQ on UNIX platforms uses the *O_SYNC* open option and the `fsync()` system call to explicitly force writes to recoverable media, and is dependent upon these options operating correctly.

   Attention: Linux You should mount the file system with the `async` option, which still supports the option of synchronous writes and gives better performance than the `sync` option.

   Note, however, that if the file system has been exported from Linux, you must still export the file system using the `sync` option.
2. Guaranteed exclusive access to files

   In order to synchronize multiple queue managers, there needs to be a mechanism for a queue manager to obtain an exclusive lock on a file.
3. Release locks on failure

   If a queue manager fails, or if there is a communication failure with the file system, files locked by the queue manager need to be unlocked and made available to other processes without waiting for the queue manager to be reconnected to the file system.

A shared file system must meet these requirements for IBM WebSphere MQ to operate reliably. If it does not, the queue manager data and logs get corrupted when using the shared file system in a multi-instance queue manager configuration.

For multi-instance queue managers on Microsoft Windows, the networked storage must be accessed by the Common Internet File System (CIFS) protocol used by Microsoft Windows networks. The Common Internet File System (CIFS) client does not meet IBM WebSphere MQ's requirements for locking semantics on platforms other than Microsoft Windows, so multi-instance queue managers running on platforms other than Microsoft Windows must not use Common Internet File System (CIFS) as their shared file system.

For multi-instance queue managers on other supported platforms, the storage must be accessed by a network file system protocol which is Posix-compliant and supports lease-based locking. Modern file systems, such as Network File System (NFS) Version 4, use leased locks to detect failures and then release locks following a failure. Older file systems such as Network File System Version 3, which do not have a reliable mechanism to release locks after a failure, must not be used with multi-instance queue managers.

Checks on whether the shared file system meets the requirements

You must check whether the shared file system you plan to use meets these requirements. You must also check whether the file system is correctly configured for reliability. Shared file systems sometimes provide configuration options to improve performance at the expense of reliability.

Under normal circumstances IBM WebSphere MQ operates correctly with attribute caching and it is not necessary to disable caching, for example by setting NOAC on an NFS mount. Attribute caching can cause issues when multiple file system clients are contending for write access to the same file on the file system server, as the cached attributes used by each client might not be the same as those attributes on the server. An example of files accessed in this way are queue manager error logs for a multi-instance queue manager. The queue manager error logs might be written to by both an active and a standby queue manager instance and cached file attributes might cause the error logs to grow larger than expected, before rollover of the files occurs.

To help to check the file system, run the task "Verifying shared file system behavior" on page 125. This task checks if your shared file system meets requirements 2 and 3. You need to verify requirement 1 in your shared file system documentation, or by experimenting with logging data to the disk.

Disk faults can cause errors when writing to disk, which IBM WebSphere MQ reports as First Failure Data Capture errors. You can run the file system checker for your operating system to check the shared file system for any disk faults. For example, on UNIX and Linux platforms the file system checker is called fsck. On Windows platforms the file system checker is called CHKDSK, or SCANDISK.

NFS server security

Note: You should put only queue manager data on a Network File System (NFS) server. On the NFS, use the following three options with the mount command to make the system secure:

noexec
By using this option, you stop binary files from being run on the NFS, which prevents a remote user from running unwanted code on the system.

nosuid
By using this option, you prevent the use of the set-user-identifier and set-group-identifier bits, which prevents a remote user from gaining higher privileges.

nodev
By using this option, you stop character and block special devices from being used or defined, which prevents a remote user from getting out of a chroot jail.

Verifying shared file system behavior

Run **amqmfsck** to check whether a shared file system on UNIX systems meets the requirements for storing the queue manager data of a multi-instance queue manager. Run the IBM WebSphere MQ MQI client sample program **amqsfhac** in parallel with **amqmfsck** to demonstrate that a queue manager maintains message integrity during a failure.

Before you begin

You need a server with networked storage, and two other servers connected to it that have WebSphere MQ installed. You must have administrator (root) authority to configure the file system, and be a WebSphere MQ Administrator to run **amqmfsck**.

About this task

"Requirements for shared file systems" on page 124 describes the file system requirements for using a shared file system with multi-instance queue managers. The IBM WebSphere MQ technote Testing and support statement for WebSphere MQ multi-instance queue managers lists the shared file systems that IBM has already tested with. The procedure in this task describes how to test a file system to help you assess whether an unlisted file system maintains data integrity.

Failover of a multi-instance queue manager can be triggered by hardware or software failures, including networking problems which prevent the queue manager writing to its data or log files. Mainly, you are interested in causing failures on the file server. But you must also cause the IBM WebSphere MQ servers to fail, to test any locks are successfully released. To be confident in a shared file system, test all of the following failures, and any other failures that are specific to your environment:

1. Shutting down the operating system on the file server including syncing the disks.
2. Halting the operating system on the file server without syncing the disks.
3. Pressing the reset button on each of the servers.
4. Pulling the network cable out of each of the servers.
5. Pulling the power cable out of each of the servers.
6. Switching off each of the servers.

Create the directory on the networked storage that you are going to use to share queue manager data and logs. The directory owner must be a WebSphere MQ Administrator, or in other words, a member of the `mqm` group on UNIX. The user who runs the tests must have WebSphere MQ Administrator authority.

Use the example of exporting and mounting a file system in Create a multi-instance queue manager on Linux to help you through configuring the file system. Different file systems require different configuration steps. Read the file system documentation.

Procedure

In each of the checks, cause all the failures in the previous list while the file system checker is running. If you intend to run **amqsfhac** at the same time as **amqmfsck**, do the task, "Running amqsfhac to test message integrity" on page 131 in parallel with this task.

1. Mount the exported directory on the two IBM WebSphere MQ servers.

   On the file system server create a shared directory `shared`, and a subdirectory to save the data for multi-instance queue managers, `qmdata`. For an example of setting up a shared directory for multi-instance queue managers on Linux, see Example in Create a multi-instance queue manager on Linux
2. Check basic file system behavior.

   On one IBM WebSphere MQ server, run the file system checker with no parameters.

   ```
   amqmfsck /shared/qmdata
   ```

   Figure 40. On IBM WebSphere MQ server 1
3. Check concurrently writing to the same directory from both IBM WebSphere MQ servers.

   On both IBM WebSphere MQ servers, run the file system checker at the same time with the `-c` option.

   ```
   amqmfsck -c /shared/qmdata
   ```

   Figure 41. On IBM WebSphere MQ server 1

```
amqmfsck -c /shared/qmdata
```

Figure 42. On IBM WebSphere MQ server 2

4. Check waiting for and releasing locks on both IBM WebSphere MQ servers.

   On both IBM WebSphere MQ servers run the file system checker at the same time with the `-w` option.

```
amqmfsck -w /shared/qmdata
```

Figure 43. On IBM WebSphere MQ server 1

```
amqmfsck -w /shared/qmdata
```

Figure 44. On IBM WebSphere MQ server 2

5. Check for data integrity.

   a) Format the test file.

   Create a large file in the directory being tested. The file is formatted so that the subsequent phases can complete successfully. The file must be large enough that there is sufficient time to interrupt the second phase to simulate the failover. Try the default value of 262144 pages (1 GB). The program automatically reduces this default on slow file systems so that formatting completes in about 60 seconds

```
amqmfsck -f /shared/qmdata
```

   The server responds with the following messages:

```
Formatting test file for data integrity test.
Test file formatted with 262144 pages of data.
```

   Figure 45. On IBM WebSphere MQ server 1

   b) Write data into the test file using the file system checker while causing a failure.

   Run the test program on two servers at the same time. Start the test program on the server which is going to experience the failure, then start the test program on the server that is going to survive the failure. Cause the failure you are investigating.

   The first test program stops with an error message. The second test program obtains the lock on the test file and writes data into the test file starting where the first test program left off. Let the second test program run to completion.

Table 12. Running the data integrity check on two servers at the same time

IBM WebSphere MQ server 1	**IBM WebSphere MQ server 2**
`amqmfsck -a /shared/qmdata`	

Table 12. Running the data integrity check on two servers at the same time (continued)

IBM WebSphere MQ server 1	IBM WebSphere MQ server 2
	`amqmfsck -a /shared/qmdata`

`Please start this program on a second machine`
`with the same parameters.`
`File lock acquired.`
`Start a second copy of this program with the same parameters on another server.`
 	
`Writing data into test file.`
 	
`To increase the effectiveness of the test,`
`interrupt the writing by ending the process,`
`temporarily breaking the network connection`
`to the networked storage,`
`rebooting the server of turning off the power.`
`To increase the effectiveness of the test,`
`interrupt the writing by ending the process,`
`temporarily breaking the network connection`
`to the networked storage,`
`rebooting the server of turning off the power.`
	`Waiting for lock...`
	`Waiting for lock...`
	`Waiting for lock...`
	`Waiting for lock...`
	`Waiting for lock...`
	`Waiting for lock...`

Turn the power off here	

	`File lock acquired.`
	`Reading test file`
	`Checking the integrity of the data read.`
	`Appending data into the test file after data already found.`
	`The test file is full of data.`
	`It is ready to be inspected for data integrity.`

The timing of the test depends on the behavior of the file system. For example, it typically takes 30 - 90 seconds for a file system to release the file locks obtained by the first program following a power outage. If you have too little time to introduce the failure before the first test program has filled the file, use the `-x` option of **amqmfsck** to delete the test file. Try the test from the start with a larger test file.

c) Verify the integrity of the data in the test file.

```
amqmfsck -i /shared/qmdata
```

The server responds with the following messages:

```
File lock acquired
Reading test file checking the integrity of the data read.
The data read was consistent.
The tests on the directory completed successfully.
```

Figure 46. On IBM WebSphere MQ server 2

6. Delete the test files.

```
amqmfsck -x /shared/qmdata
Test files deleted.
```

Figure 47. On IBM WebSphere MQ server 2

The server responds with the message:

```
Test files deleted.
```

Results

The program returns an exit code of zero if the tests complete successfully, and non-zero otherwise.

Examples

The first set of three examples shows the command producing minimal output.

Successful test of basic file locking on one server

```
> amqmfsck /shared/qmdata
The tests on the directory completed successfully.
```

Failed test of basic file locking on one server

```
> amqmfsck /shared/qmdata
AMQ6245: Error Calling 'write()[2]' on file '/shared/qmdata/amqmfsck.lck' error '2'.
```

Successful test of locking on two servers

Table 13. Successful locking on two servers

IBM WebSphere MQ server 1	IBM WebSphere MQ server 2

`> amqmfsck -w /shared/qmdata`
`Please start this program on a second`
`machine with the same parameters.`
`Lock acquired.`
`Press Return`
`or terminate the program to release the lock.`	

	`> amqmfsck -w /shared/qmdata` `Waiting for lock...`
`[ Return pressed ]` `Lock released.`	

	`Lock acquired.`
	`The tests on the directory completed`
	`successfully`

The second set of three examples shows the same commands using verbose mode.

Successful test of basic file locking on one server

```
> amqmfsck -v /shared/qmdata
System call: stat("/shared/qmdata")'
System call: fd = open("/shared/qmdata/amqmfsck.lck", O_RDWR, 0666)
System call: fchmod(fd, 0666)
System call: fstat(fd)
System call: fcntl(fd, F_SETLK, F_WRLCK)
System call: write(fd)
System call: close(fd)
System call: fd = open("/shared/qmdata/amqmfsck.lck", O_RDWR, 0666)
```

```
System call: fcntl(fd, F_SETLK, F_WRLCK)
System call: close(fd)
System call: fd1 = open("/shared/qmdata/amqmfsck.lck", O_RDWR, 0666)
System call: fcntl(fd1, F_SETLK, F_RDLCK)
System call: fd2 = open("/shared/qmdata/amqmfsck.lck", O_RDWR, 0666)
System call: fcntl(fd2, F_SETLK, F_RDLCK)
System call: close(fd2)
System call: write(fd1)
System call: close(fd1)
The tests on the directory completed successfully.
```

Failed test of basic file locking on one server

```
> amqmfsck -v /shared/qmdata
System call: stat("/shared/qmdata")
System call: fd = open("/shared/qmdata/amqmfsck.lck", O_RDWR, 0666)
System call: fchmod(fd, 0666)
System call: fstat(fd)
System call: fcntl(fd, F_SETLK, F_WRLCK)
System call: write(fd)
System call: close(fd)
System call: fd = open("/shared/qmdata/amqmfsck.lck", O_RDWR, 0666)
System call: fcntl(fd, F_SETLK, F_WRLCK)
System call: close(fd)
System call: fd = open("/shared/qmdata/amqmfsck.lck", O_RDWR, 0666)
System call: fcntl(fd, F_SETLK, F_RDLCK)
System call: fdSameFile = open("/shared/qmdata/amqmfsck.lck", O_RDWR, 0666)
System call: fcntl(fdSameFile, F_SETLK, F_RDLCK)
System call: close(fdSameFile)
System call: write(fd)
AMQxxxx: Error calling 'write()[2]' on file '/shared/qmdata/amqmfsck.lck', errno 2
(Permission denied).
```

Successful test of locking on two servers

Table 14. Successful locking on two servers - verbose mode

IBM WebSphere MQ server 1	IBM WebSphere MQ server 2

`> amqmfsck -wv /shared/qmdata`
`Calling 'stat("/shared/qmdata")'`
`Calling 'fd = open("/shared/qmdata/`
`amqmfsck.lkw",`
`O_EXCL | O_CREAT | O_RDWR, 0666)'`
`Calling 'fchmod(fd, 0666)'`
`Calling 'fstat(fd)'`
`Please start this program on a second`
`machine with the same parameters.`
`Calling 'fcntl(fd, F_SETLK, F_WRLCK)'`
`Lock acquired.`
`Press Return`
`or terminate the program to release the lock.`	

	`> amqmfsck -wv /shared/qmdata`
	`Calling 'stat("/shared/qmdata")'`
	`Calling 'fd = open("/shared/qmdata/`
	`amqmfsck.lkw",`
	`O_EXCL | O_CREAT | O_RDWR,0666)'`
	`Calling 'fd = open("/shared/qmdata/amqmfsck.lkw,`
	`O_RDWR, 0666)'`
	`Calling 'fcntl(fd, F_SETLK, F_WRLCK)`
	`'Waiting for lock...`

`[ Return pressed ]`
`Calling 'close(fd)'`
`Lock released.`	

	`Calling 'fcntl(fd, F_SETLK, F_WRLCK)'`
	`Lock acquired.`
	`The tests on the directory completed`
	`successfully`

Related reference
amqmfsck (file system check)

*Running **amqsfhac** to test message integrity*
amqsfhac checks that a queue manager using networked storage maintains data integrity following a failure.

Before you begin

You require four servers for this test. Two servers for the multi-instance queue manager, one for the file system, and one for running **amqsfhac** as a IBM WebSphere MQ MQI client application.

Follow step "1" on page 126 in Procedure to set up the file system for a multi-instance queue manager.

About this task

Procedure

1. Create a multi-instance queue manager on another server, QM1, using the file system you created in step "1" on page 126 in Procedure.

   See Create a multi-instance queue manager.
2. Start the queue manager on both servers making it highly available.

   On server 1:

   ```
   strmqm -x QM1
   ```

   On server 2:

   ```
   strmqm -x QM1
   ```
3. Set up the client connection to run **amqsfhac**.
   a) Use the procedure in Verifying a client installation to set up a client connection, or the example scripts in Reconnectable client samples .
   b) Modify the client channel to have two IP addresses, corresponding to the two servers running QM1.

      In the example script, modify:

      ```
      DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN)  TRPTYPE(TCP) +
      CONNAME('LOCALHOST(2345)') QMNAME(QM1) REPLACE
      ```

      To:

      ```
      DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN)  TRPTYPE(TCP) +
      CONNAME('server1(2345),server2(2345)') QMNAME(QM1) REPLACE
      ```

      Where server1 and server2 are the host names of the two servers, and 2345 is the port that the channel listener is listening on. Usually this defaults to 1414. You can use 1414 with the default listener configuration.
4. Create two local queues on QM1 for the test.
   Run the following MQSC script:

   ```
   DEFINE QLOCAL(TARGETQ) REPLACE
   DEFINE QLOCAL(SIDEQ) REPLACE
   ```
5. Test the configuration with **amqsfhac**

   ```
   amqsfhac QM1 TARGETQ SIDEQ 2 2 2
   ```
6. Test message integrity while you are testing file system integrity.

Run **amqsfhac** during step “5” on page 127 of Procedure.

```
amqsfhac QM1 TARGETQ SIDEQ 10 20 0
```

If you stop the active queue manager instance, **amqsfhac** reconnects to the other queue manager instance once it has become active. Restart the stopped queue manager instance again, so that you can reverse the failure in your next test. You will probably need to increase the number of iterations based on experimentation with your environment so that the test program runs for sufficient time for the failover to occur.

Results

An example of running **amqsfhac** in step “6” on page 131 is shown in Figure 48 on page 132. The test is a success.

If the test detected a problem, the output would report the failure. In some test runs `MQRC_CALL_INTERRUPTED` might report "`Resolving to backed out`". It makes no difference to the result. The outcome depends on whether the write to disk was committed by the networked file storage before or after the failure took place.

```
Sample AMQSFHAC start
qmname = QM1
qname = TARGETQ
sidename = SIDEQ
transize = 10
iterations = 20
verbose = 0
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Resolving MQRC_CALL_INTERRUPTED
MQGET browse side tranid=14 pSideinfo->tranid=14
Resolving to committed
Iteration 7
Iteration 8
Iteration 9
Iteration 10
Iteration 11
Iteration 12
Iteration 13
Iteration 14
Iteration 15
Iteration 16
Iteration 17
Iteration 18
Iteration 19
Sample AMQSFHAC end
```

*Figure 48. Output from a successful run of **amqsfhac***

Related reference

High availability sample programs

Sharing IBM WebSphere MQ files

Some IBM WebSphere MQ files are accessed exclusively by an active queue manager, other files are shared.

WebSphere MQ files are split into program files and data files. Program files are typically installed locally on each server running WebSphere MQ. Queue managers share access to data files and directories in the default data directory. They require exclusive access to their own queue manager directory trees contained in each of the `qmgrs` and `log` directories shown in Figure 49 on page 133.

Figure 49 on page 133 is a high-level view of the WebSphere MQ directory structure. It shows the directories which can be shared between queue managers and made remote. The details vary by platform. The dotted lines indicate configurable paths.

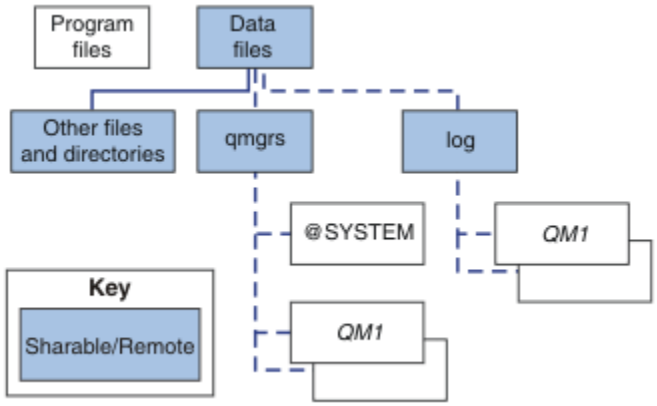


Figure 49. Overall view of WebSphere MQ directory structure

Program files

The program files directory is typically left in the default location, is local, and shared by all the queue managers on the server.

Data files

The data files directory is typically local in the default location, `/var/mqm` on UNIX and Linux systems and configurable on installation on Windows. It is shared between queue managers. You can make the default location remote, but do not share it between different installations of WebSphere MQ. The `DefaultPrefix` attribute in the WebSphere MQ configuration points to this path.

`qmgrs`

From v7.0.1, there are two alternative ways to specify the location of queue manager data.

Using `Prefix`

The `Prefix` attribute specifies the location of the `qmgrs` directory. WebSphere MQ constructs the queue manager directory name from the queue manager name and creates it as a subdirectory of the `qmgrs` directory.

The `Prefix` attribute is located in the QueueManager stanza, and is inherited from the value in the `DefaultPrefix` attribute. By default, for administrative simplicity, queue managers typically share the same `qmgrs` directory.

The `QueueManager` stanza is in the `mqs.ini` file.

If you change the location of the `qmgrs` directory for any queue manager, you must change the value of its `Prefix` attribute.

The `Prefix` attribute for the `QM1` directory in Figure 49 on page 133 for a UNIX and Linux platform is,

```
Prefix=/var/mqm
```

Using `DataPath`

The `DataPath` attribute specifies the location of the queue manager data directory.

The `DataPath` attribute specifies the complete path, including the name of the queue manager data directory. The `DataPath` attribute is unlike the `Prefix` attribute, which specifies an incomplete path to the queue manager data directory.

The `DataPath` attribute, if it is specified, is located in the QueueManager stanza. If it has been specified, it takes precedence over any value in the `Prefix` attribute.

The QueueManager stanza is in the mqs.ini file.

If you change the location of the queue manager data directory for any queue manager you must change the value of the DataPath attribute.

The DataPath attribute for the QM1 directory in Figure 49 on page 133 for a UNIX or Linux platform is,

```
DataPath=/var/mqm/qmgrs/QM1
```

log

The log directory is specified separately for each queue manager in the Log stanza in the queue manager configuration. The queue manager configuration is in qm.ini.

***DataPath*/*QmgrName*/@IPCC subdirectories**

The *DataPath*/*QmgrName*/@IPCC subdirectories are in the shared directory path. They are used to construct the directory path for IPC file system objects. They need to distinguish the namespace of a queue manager when a queue manager is shared between systems. Before V7.0.1, a queue manager was only used on one system. One set of subdirectories was sufficient to define the directory path to IPC file system objects, see Figure 50 on page 134.

```
DataPath/QmgrName/@IPCC/esem
```

Figure 50. Example IPC subdirectory, pre-V7.0.1

In V7.0.1, and later, the IPC file system objects have to be distinguished by system. A subdirectory, for each system the queue manager runs on, is added to the directory path, see Figure 51 on page 134.

```
DataPath/QmgrName/@IPCC/esem/myHostName/
```

Figure 51. Example IPC subdirectory, V7.0.1 and subsequent releases

myHostName is up to the first 20 characters of the host name returned by the operating system. On some systems, the host name might be up to 64 characters in length before truncation. The generated value of *myHostName* might cause a problem for two reasons:

1. The first 20 characters are not unique.
2. The host name is generated by a DHCP algorithm that does not always allocate the same host name to a system.

In these cases, set *myHostName* using the environment variable, MQC_IPC_HOST; see Figure 52 on page 134.

```
export MQS_IPC_HOST=myHostName
```

Figure 52. Example: setting MQC_IPC_HOST

Other files and directories

Other files and directories, such as the directory containing trace files, and the common error log, are normally shared and kept on the local file system.

Up until v7.0.1, WebSphere MQ relied upon external management to guarantee queue managers exclusive access to the queue manager data and log files. From v7.0.1 onwards, with support of shared file systems, WebSphere MQ manages exclusive access to these files using file system locks. A file system lock allows only one instance of a particular queue manager to be active at a time.

When you start the first instance of a particular queue manager it takes ownership of its queue manager directory. If you start a second instance, it can only take ownership if the first instance has stopped. If the first queue manager is still running, the second instance fails to start, and reports the queue manager is running elsewhere. If the first queue manager has stopped, then the second queue manager takes over ownership of the queue manager files and becomes the running queue manager.

You can automate the procedure of the second queue manager taking over from the first. Start the first queue manager with the `strmqm -x` option that permits another queue manager to take over from it. The second queue manager then waits until the queue manager files are unlocked before attempting to take over ownership of the queue manager files, and start.

Directory structure on UNIX and Linux systems

The WebSphere MQ directory structure on UNIX and Linux systems can be mapped to different file systems for easier management, better performance, and better reliability.

Use the flexible directory structure of WebSphere MQ to take advantage of shared file systems for running multi-instance queue managers.

Use the command **crtmqm** *QM1* to create the directory structure shown in Figure 53 on page 135 where R is the release of the product. It is a typical directory structure for a queue manager created on a WebSphere MQ system from v7.0.1 onwards. Some directories, files and .ini attribute settings are omitted for clarity, and another queue manager name might be altered by mangling. The names of the file systems vary on different systems.

In a typical installation, every queue manager that you create points to common `log` and `qmgrs` directories on the local file system. In a multi-instance configuration, the `log` and `qmgrs` directories are on a network file system shared with another installation of WebSphere MQ.

Figure 53 on page 135 shows the default configuration for WebSphere MQ v7.R on AIX where R is the release of the product. For examples of alternative multi-instance configurations, see "Example directory configurations on UNIX and Linux systems" on page 139.

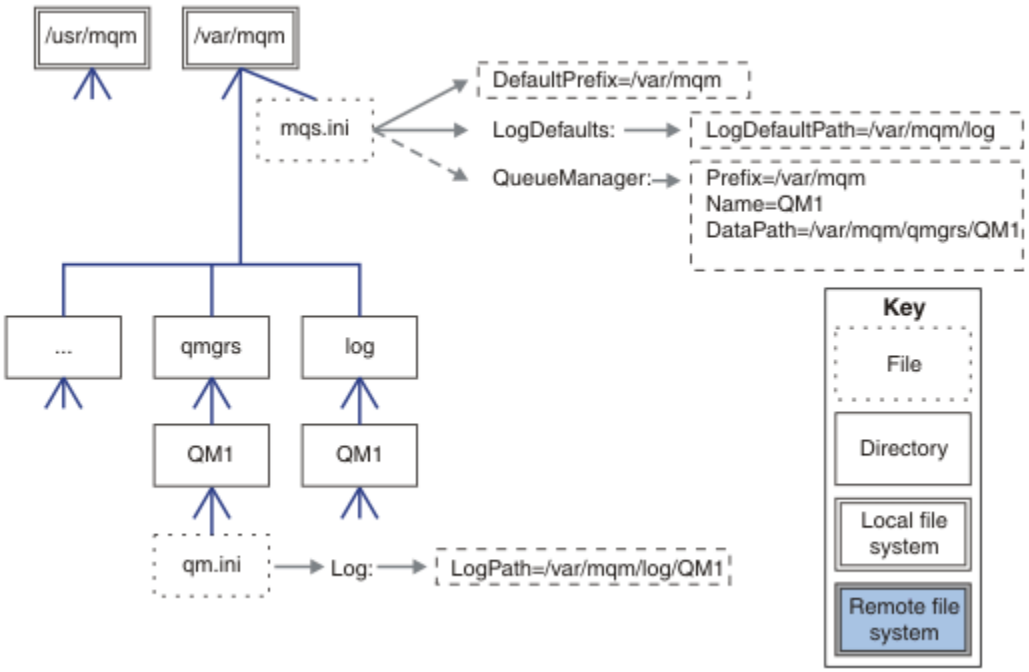


Figure 53. Example default WebSphere MQ v7.R directory structure for UNIX and Linux systems

The product is installed into `/usr/mqm` on AIX and `/opt/mqm` on the other systems, by default. The working directories are installed into the `/var/mqm` directory.

Note: If you created the `/var/mqm` file system prior to installing IBM WebSphere MQ, ensure that the mqm user has full directory permissions, for example, file mode 755.

The `log` and `qmgrs` directories are shown in their default locations as defined by the default values of the `LogDefaultPath` and `DefaultPrefix` attributes in the `mqs.ini` file. When a queue manager is created, by default the queue manager data directory is created in *`DefaultPrefix`*`/qmgrs` , and the log file directory in *`LogDefaultPath`* `/log`. `LogDefaultPath` and `DefaultPrefix` only effects where

queue managers and log files are created by default. The actual location of a queue manager directory is saved in the `mqs.ini` file, and the location of the log file directory is saved in the `qm.ini` file.

The log file directory for a queue manager is defined in the `qm.ini` file in the `LogPath` attribute. Use the `-ld` option on the **crtmqm** command to set the `LogPath` attribute for a queue manager; for example, **crtmqm** `-ld` *LogPath* `QM1`. If you omit the `ld` parameter the value of `LogDefaultPath` is used instead.

The queue manager data directory is defined in the `DataPath` attribute in the `QueueManager` stanza in the `mqs.ini` file. Use the `-md` option on the **crtmqm** command to set the `DataPath` for a queue manager; for example, **crtmqm** `-md` *DataPath* `QM1`. If you omit the `md` parameter the value of the `DefaultPrefix` or `Prefix` attribute is used instead. `Prefix` takes precedence over `DefaultPrefix`.

Typically, create `QM1` specifying both the log and data directories in a single command.

```
crtmqm
 -md DataPath -ld
 LogPath QM1
```

You can modify the location of a queue manager log and data directories of an existing queue manager by editing the `DataPath` and `LogPath` attributes in the `qm.ini` file when the queue manager is stopped.

The path to the `errors` directory, like the paths to all the other directories in `/var/mqm`, is not modifiable. However the directories can be mounted on different file systems, or symbolically linked to different directories.

Directory content on UNIX and Linux systems

Content of the directories associated with a queue manager.

For information about the location of the product files, see Choosing an installation location

For information about alternative directory configurations, see “Planning file system support” on page 123.

In Figure 54 on page 137, the layout is representative of WebSphere MQ after a queue manager has been in use for some time. The actual structure that you have depends on which operations have occurred on the queue manager.

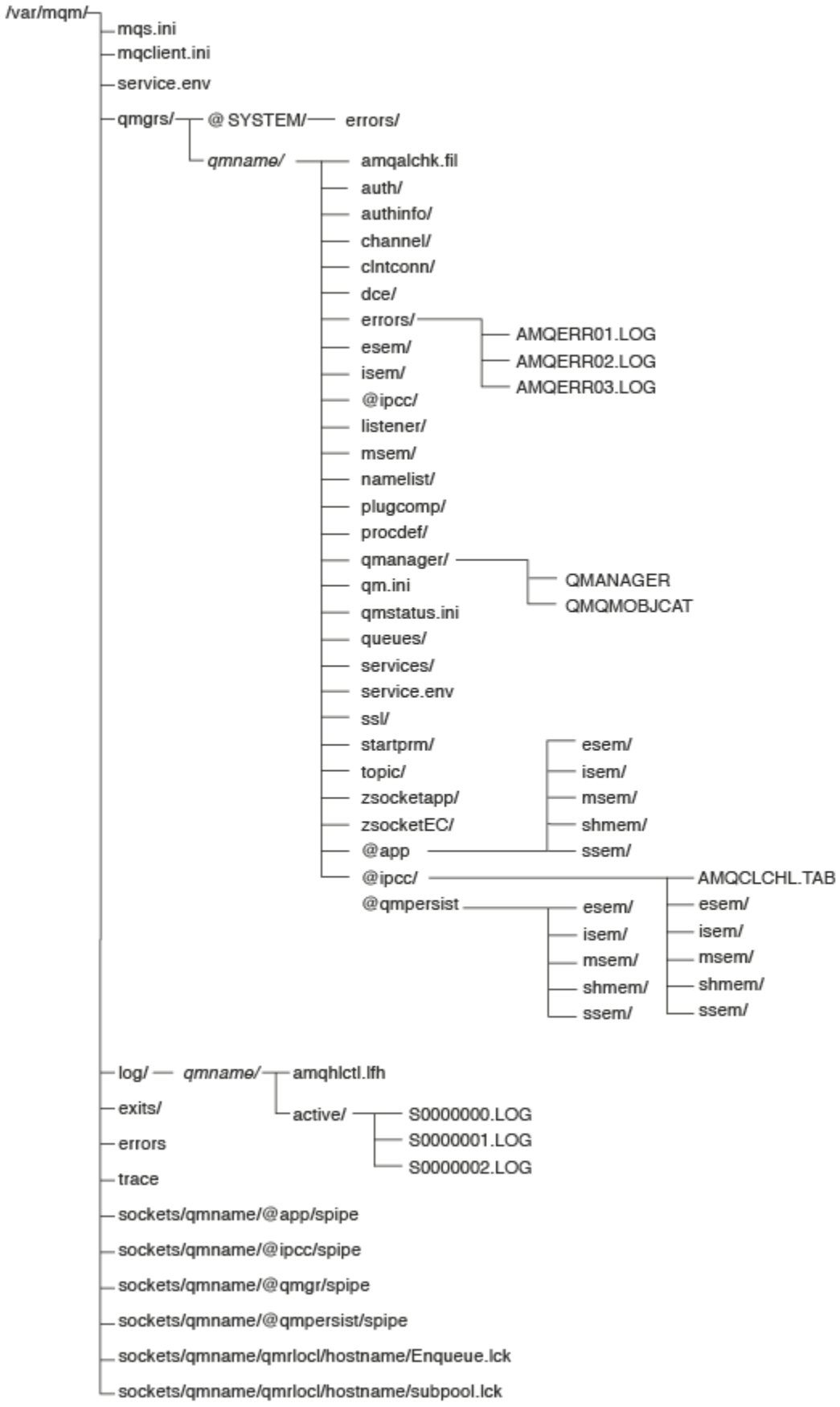


Figure 54. Default directory structure (UNIX systems) after a queue manager has been started

/var/mqm/

The /var/mqm directory contains configuration files and output directories that apply to a WebSphere MQ installation as a whole, and not to an individual queue manager.

Table 15. Documented content of the /var/mqm directory on UNIX systems	
mqs.ini	WebSphere MQ installation wide configuration file, read when a queue manager starts. File path modifiable using the **AMQ_MQS_INI_LOCATION** environment variable. Ensure this is set and exported in the shell in which the **strmqm** command is run.
mqclient.ini	Default client configuration file read by WebSphere MQ MQI client programs. File path modifiable using the **MQCLNTCF** environment variable.
service.env	Contains machine scope environment variables for a service process. File path fixed.
errors/	Machine scope error logs, and FFST files. Directory path fixed. See also, FFST: IBM WebSphere MQ for UNIX and Linux systems.
sockets/	Contains information for each queue manager for system use only.
trace/	Trace files. Directory path fixed.
exits/	Default directory containing user channel exit programs. Location modifiable in ApiExit stanzas in the mqs.ini file.
exits64/	

/var/mqm/qmgrs/*qmname*/

/var/mqm/qmgrs/*qmname*/ contains directories and files for a queue manager. The directory is locked for exclusive access by the active queue manager instance. The directory path is directly modifiable in the mqs.ini file, or by using the **md** option of the **crtmqm** command.

Table 16. Documented contents of the /var/mqm/qmgrs/qmname directory on UNIX systems	
qm.ini	Queue manager configuration file, read when a queue manager starts.
errors/	Queue manager scope error logs. *qmname* = @system contains channel-related messages for an unknown or unavailable queue manager.
@ipcc/ AMQCLCHL.TAB	Default client channel control table, created by the WebSphere MQ server, and read by WebSphere MQ MQI client programs. File path modifiable using the **MQCHLLIB** and **MQCHLTAB** environment variables.
qmanager	Queue manager object file: QMANAGER Queue manager object catalog: QMQMOBJCAT

Table 16. Documented contents of the /var/mqm/qmgrs/qmname directory on UNIX systems (continued)	
authinfo/	Each object defined within the queue manager is associated with a file in these directories. The file name approximately matches the definition name; see, Understanding WebSphere MQ file names .
channel/	
clntconn/	
listener/	
namelist/	
procdef/	
queues/	
services/	
topics/	
...	Other directories used by WebSphere MQ, such as @ipcc, to be modified only by WebSphere MQ.

/var/mqm/log/*qmname*/

/var/mqm/log/*qmname*/ contains the queue manager log files. The directory is locked for exclusive access by the active queue manager instance. The directory path is modifiable in the qm.ini file, or by using the **ld** option of the **crtmqm** command.

Table 17. Documented contents of the /var/mqm/log/qmname directory on UNIX systems	
amqhlctl.lfh	Log control file.
active/	This directory contains the log files numbered S0000000.LOG, S0000001.LOG, S0000002.LOG, and so on.

Example directory configurations on UNIX and Linux systems

Examples of alternative file system configurations on UNIX and Linux systems.

You can customize the WebSphere MQ directory structure in various ways to achieve a number of different objectives.

- Place the qmgrs and log directories on remote shared file systems to configure a multi-instance queue manager.
- Use separate file systems for the data and log directories, and allocate the directories to different disks, to improve performance by reducing I/O contention.
- Use faster storage devices for directories that have a greater effect on performance. Physical device latency is frequently a more important factor in the performance of persistent messaging than whether a device is mounted locally or remotely. The following list shows which directories are most and least performance sensitive.
  1. log
  2. qmgrs
  3. Other directories, including /usr/mqm
- Create the qmgrs and log directories on file systems that are allocated to storage with good resilience, such as a redundant disk array, for example.
- It is better to store the common error logs in var/mqm/errors , locally, rather than on a network file system, so that error relating to the network file system can be logged.

Figure 55 on page 140 is a template from which alternative WebSphere MQ directory structures are derived. In the template, dotted lines represent paths that are configurable. In the examples, the

dotted lines are replaced by solid lines that correspond to the configuration information stored in the AMQ_MQS_INI_LOCATION environment variable, and in the `mqs.ini` and `qm.ini` files.

Note: The path information is shown as it appears in the `mqs.ini` or `qm.ini` files. If you supply path parameters in the **crtmqm** command, omit the name of the queue manager directory: the queue manager name is added to the path by WebSphere MQ after it has been mangled.

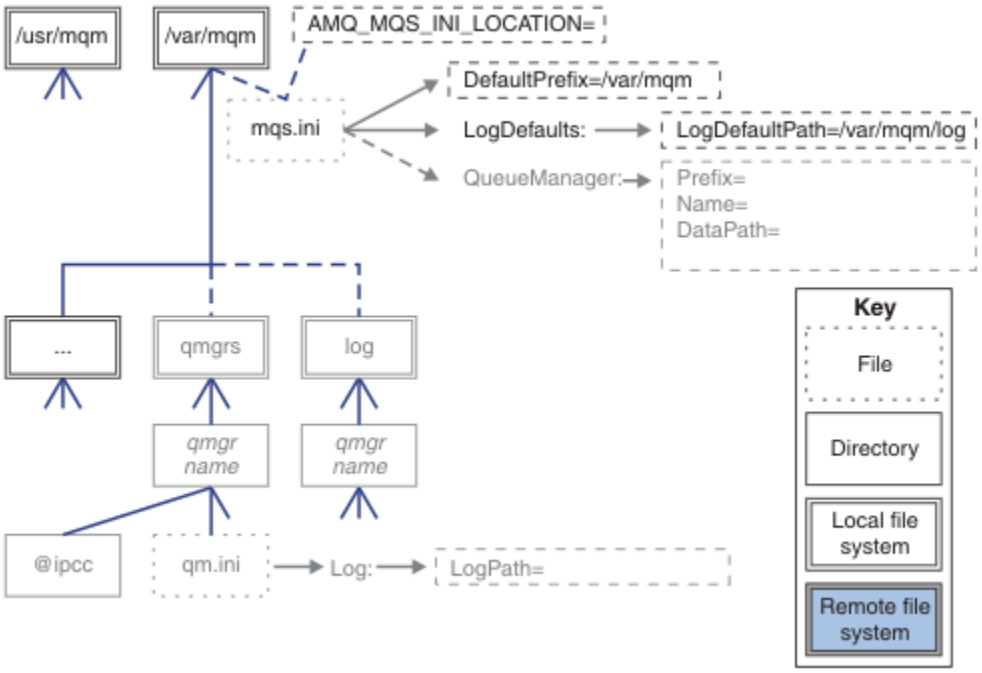


Figure 55. Directory structure pattern template

Examples of configured directory structures follow. The first example shows a typical default directory structure for WebSphere MQ v7.0.1 created by issuing the **crtmqm** *QM1* command. The second example shows how a typical directory structure appears for a queue manager created using a WebSphere MQ release earlier than v7.0.1. The directory structure does not change.

Queue managers newly created in version 7.0.1 have a different configuration file to earlier releases of v7. If you need to remove the v7.0.1 fix pack to revert to v7.0.0.2, you need to re-create the configuration files. You might only need to use the Prefix attribute to define the path to the new queue manager data directory, or you might need to move the queue manager data directory and log directories to a different location. The safest way to reconfigure the queue manager is to save the queue manager data and log directories, delete and re-create the queue manager, and then replace the data and log directories in their new location, with the ones that have been saved.

Typical directory structure for release v7.0.1 onwards

Figure 56 on page 141 is the default directory structure created in v7.0.1 by issuing the command **crtmqm** *QM1*.

The `mqs.ini` file has a stanza for the QM1 queue manager, created by referring to the value of `DefaultPrefix` . The `Log` stanza in the `qm.ini` file has a value for `LogPath`, set by reference to `LogDefaultPath` in `mqs.ini` .

Use the optional **crtmqm** parameters to override the default values of `DataPath` and `LogPath`.

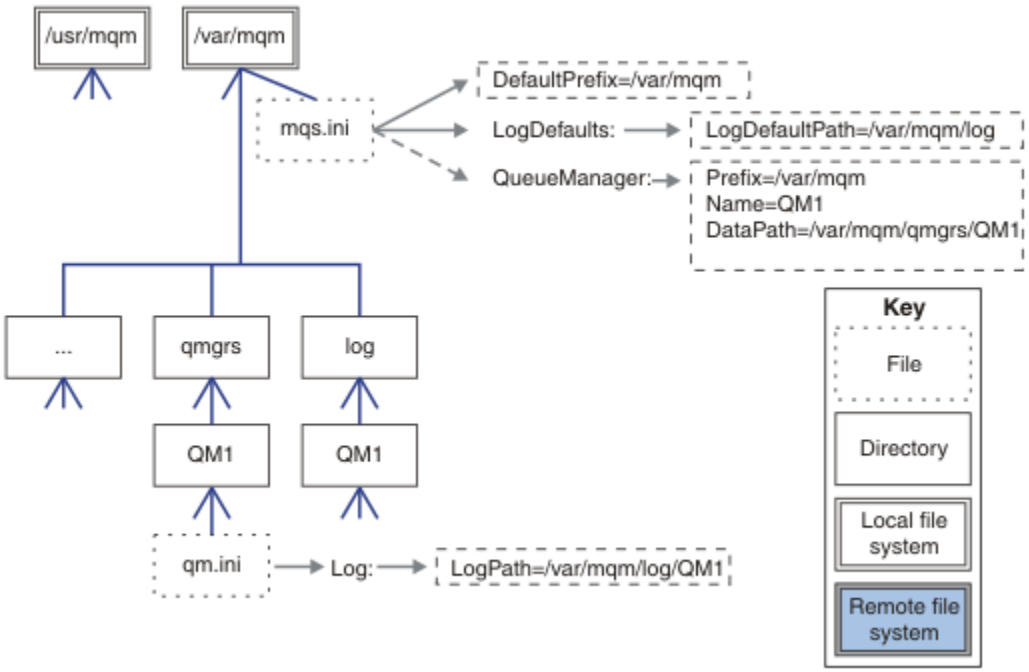


Figure 56. Example default WebSphere MQ v7.R directory structure for UNIX and Linux systems

Typical directory structure for releases earlier than v7.0.1

The `DataPath` attribute did not exist before WebSphere MQ v7.0.1; the attribute is not present in the `mqs.ini` file. The location of the `qmgrs` directory was configured using the `Prefix` attribute. The location of individual directories could be configured by using symbolic links to point to different file system locations.

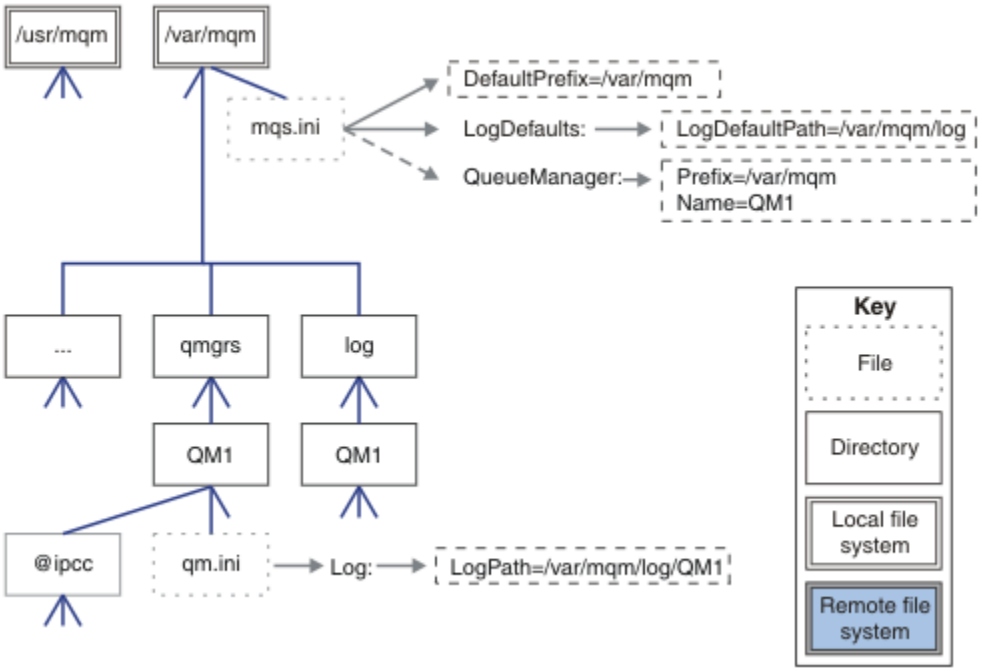


Figure 57. Typical directory structure for releases earlier than v7.0.1

Share default qmgrs and log directories (Release v7.0.1 onwards)

An alternative to “Share everything (Release v7.0.1 onwards)” on page 143 , is to share the qmgrs and log directories separately (Figure 58 on page 142). In this configuration, there is no need to set AMQ_MQS_INI_LOCATION as the default mqs.ini is stored in the local /var/mqm file system. The files and directories, such as mqclient.ini and mqserver.ini are also not shared.

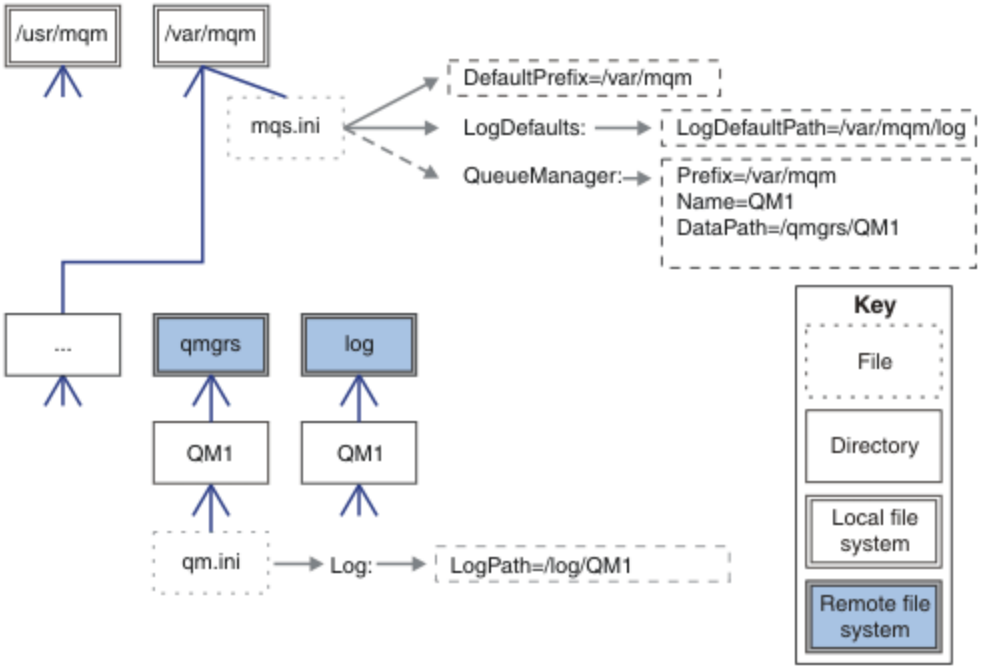


Figure 58. Share qmgrs and log directories

Share named qmgrs and log directories (Release v7.0.1 onwards)

The configuration in Figure 59 on page 143 places the log and qmgrs in a common named remote shared file system called /ha . The same physical configuration can be created in two different ways.

1. Set LogDefaultPath=/ha and then run the command, **crtmqm** -md */ha/qmgrs* QM1. The result is exactly as illustrated in Figure 59 on page 143.
2. Leave the default paths unchanged and then run the command, **crtmqm** -ld */ha/log* - md */ha/qmgrs* QM1.

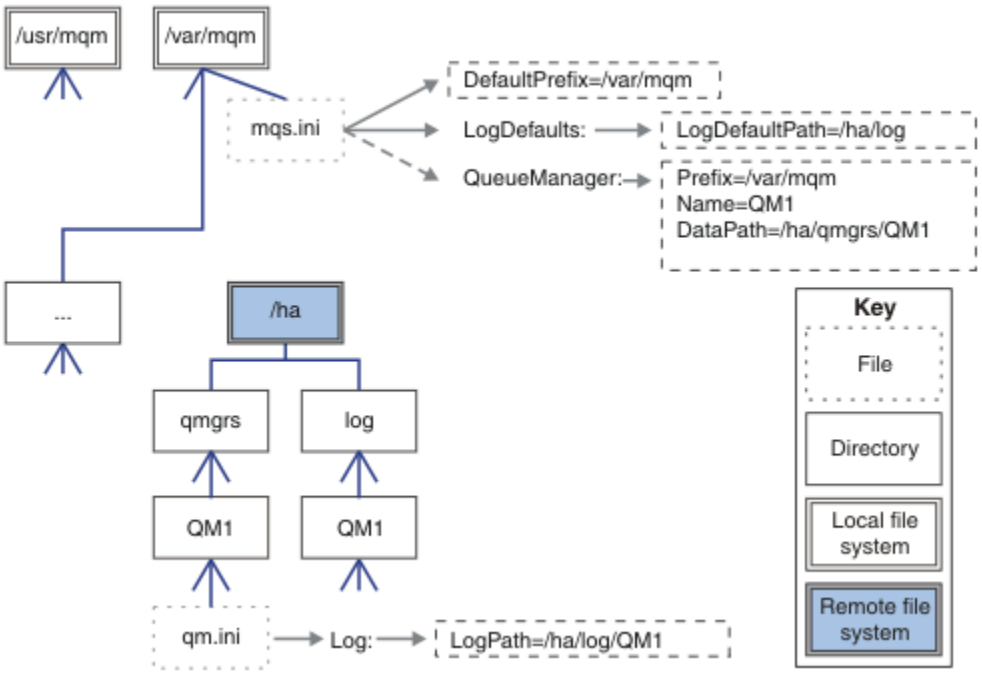


Figure 59. Share named `qmgrs` and `log` directories

Share everything (Release v7.0.1 onwards)

Figure 60 on page 144 is a simple configuration for system with fast networked file storage.

Mount `/var/mqm` as a remote shared file system. By default, when you start QM1, it looks for `/var/mqm` , finds it on the shared file system, and reads the `mqs.ini` file in `/var/mqm`. Rather than use the single `/var/mqm/mqs.ini` file for queue managers on all your servers, you can set the `AMQ_MQS_INI_LOCATION` environment variable on each server to point to different `mqs.ini` files.

Note: The contents of the generic error file in `/var/mqm/errors/` are shared between queue managers on different servers.

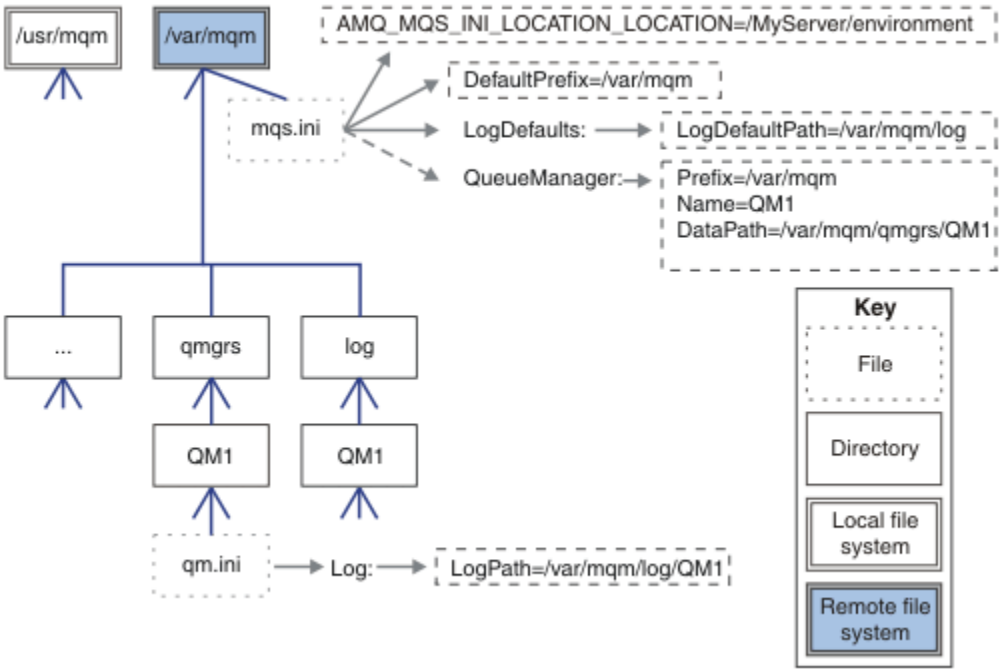


Figure 60. Share everything

Note that you cannot use this for multi-instance queue managers. The reason is that it is necessary for each host in a multi-instance queue manager to have its own local copy of `/var/mqm` to keep track of local data, such as semaphores and shared memory. These entities cannot be shared across hosts.

Directory structure on Windows systems

How to find queue manager configuration information and directories on Windows.

The default directory for IBM WebSphere MQ for Windows installation is:

32 bit
```
C:\Program Files\IBM\WebSphere MQ
```
64 bit
```
C:\Program Files (x86)\IBM\WebSphere MQ
```

The installation information is stored in the Windows registry. The registry key where IBM WebSphere MQ information is stored is:

32 bit
```
My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere MQ\
```
64 bit
```
My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\IBM\WebSphere MQ\
```

Each installation has a specific sub key:

```
Installation\<InstallationName>\
```

The path that points to the IBM WebSphere MQ data directory is stored in a string value named `WorkPath` and the default directory for logs is stored in `LogDefaultPath`. Queue manager data directories are created in *`WorkPath`*`\qmgrs\`*`Qmgrname`*. Queue manager logs are created in *`LogDefaultPath`*`\`*`QmgrName`*. See Figure 61 on page 145.

If you define the queue manager data and log directories when installing IBM WebSphere MQ, then *WorkPath* and *LogDefaultPath* are updated with the customized path information.

WorkPath and *LogDefaultPath* are only used to create a queue manager.

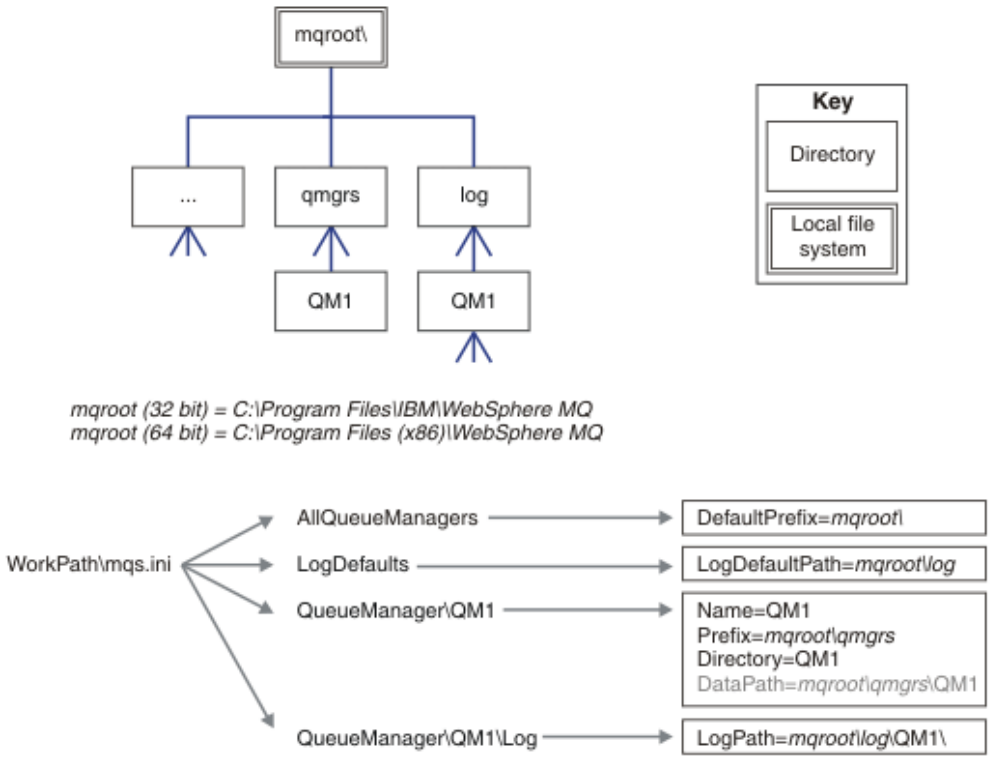


Figure 61. Default directory structure for Windows

Multi-instance queue managers

To configure a multi-instance queue manager, the log and data directories must be placed on networked storage, preferably on a different server to any of the servers that are running instances of the queue manager.

Two parameters are provided on the **crtmqm** command, **-md** and **-ld**, to make it easier specify the location of the queue manager data and log directories. The effect of specifying the **-md** parameter is fourfold:

1. The `mqs.ini` stanza `QueueManager\`*`QmgrName`* contains a new variable, *DataPath*, which points to the queue manager data directory. Unlike the *Prefix* variable, the path includes the name of the queue manager directory.
2. The queue manager configuration information stored in the mqs.ini file is reduced to *Name*, *Prefix*, *Directory* and *DataPath*.

Directory content

Lists the location and content of WebSphere MQ directories.

A WebSphere MQ configuration has three main sets of files and directories,

1. Executable, and other read-only files such as the readme file, the WebSphere MQ Explorer plug-in and help files, and licenses files, that are only updated when maintenance is applied. These files are described in Table 18 on page 146.
2. Potentially modifiable files and directories that are not specific to a particular queue manager. These files and directories are described in Table 19 on page 146.
3. Files and directories that are specific to each queue manager on a server. These files and directories are described in Table 18 on page 146

Resource directories and files

The resource directories and files contain all the executable code and resources to run a queue manager. The variable, *FilePath*, in the installation specific IBM WebSphere MQ configuration registry key, contains the path to the resource directories.

Table 18. Directories and files in the FilePath directory

File path	Contents
FilePath\bin	Commands and DLLs
FilePath\bin64	Commands and DLLs (64 bit)
FilePath\conv	Data conversion tables
FilePath\doc	Wizard help files
FilePath\MQExplorer	Explorer and Explorer help Eclipse plug-ins
FilePath\gskit8	Global security kit
FilePath\java	Java resources, including JRE
FilePath\licenses	License information
FilePath\Non_IBM_License	License information
FilePath\properties	Used internally
FilePath\Tivoli	
FilePath\tools	Development resources and samples
FilePath\Uninst	Used internally
FilePath\README.TXT	Readme file

Directories not specific to a queue manager

Some directories contain files, such as trace files and error logs, that are not specific to a specific queue manager. The *DefaultPrefix* variable contains the path to these directories. *DefaultPrefix* is part of the `AllQueueManagers` stanza.

Table 19. Directories and files in DefaultPrefix directory

File path	Contents
DefaultPrefix\Config	Used internally
DefaultPrefix\conv	ccsid.tbl data conversion control file, described in Data conversion
DefaultPrefix\errors	Non queue manager error logs, AMQERR*nn*.LOG
DefaultPrefix\exits	Channel exit programs
DefaultPrefix\exits64	Channel exit programs (64 bit)
DefaultPrefix\ipc	Not used
DefaultPrefix\Qmgrs	Described in Table 20 on page 147
DefaultPrefix\trace	Trace files
DefaultPrefix\amqmjpse.txt	Used internally

Queue manager directories

When you create a queue manager, a new set of directories, specific to the queue manager, is created.

If you create a queue manager with the **-md** *filepath* parameter, the path is stored in the *DataPath* variable in the queue manager stanza of the mqs.ini file. If you create a queue manager without setting the **-md** *filepath* parameter, the queue manager directories are created in the path stored in *DefaultPrefix*, and the path is copied into the *Prefix* variable in the queue manager stanza of the mqs.ini file.

Table 20. Directories and files in DataPath and Prefix/Qmgrs/QmgrName directories

File path	Contents
DataPath\@ipcc	Default location for AMQCLCHL.TAB, the client connection table.
DataPath\authinfo	Used internally.
DataPath\channel	
DataPath\clntconn	
DataPath\errors	Error logs, AMQERR*nn*.LOG
DataPath\listener	Used internally.
DataPath\namelist	
DataPath\plugcomp	
DataPath\procdef	
DataPath\qmanager	
DataPath\queues	
DataPath\services	
DataPath\ssl	
DataPath\startprm	
DataPath\topic	
DataPath\active	
DataPath\active.dat	
DataPath\amqalchk.fil	
DataPath\master	
DataPath\master.dat	
DataPath\qm.ini	Queue manager configuration
DataPath\qmstatus.ini	Queue manager status
Prefix\Qmgrs*QmgrName*	Used internally
Prefix\Qmgrs\@SYSTEM	Not used
Prefix\Qmgrs\@SYSTEM\errors	

IBM WebSphere MQ and UNIX System V IPC resources

A queue manager uses some IPC resources. Use **ipcs -a** to find out what resources are being used.

This information applies to IBM WebSphere MQ running on UNIX and Linux systems only.

IBM WebSphere MQ uses System V interprocess communication (IPC) resources (*semaphores* and *shared memory segments*) to store and pass data between system components. These resources are used by queue manager processes and applications that connect to the queue manager. IBM WebSphere MQ MQI clients do not use IPC resources, except for IBM WebSphere MQ trace control. Use the UNIX command **ipcs -a** to get full information on the number and size of the IPC resources currently in use on the machine.

Shared memory on AIX

If certain application types fail to connect because of an AIX memory limitation, in most cases this can be resolved by setting the environment variable EXTSHM=ON.

Some 32-bit processes on AIX might encounter an operating system limitation that affects their ability to connect to WebSphere MQ queue managers. Every standard connection to WebSphere MQ uses shared memory, but unlike other UNIX and Linux platforms, AIX allows 32-bit processes to attach only 11 shared memory sets.

Most 32-bit processes will not encounter this limit, but applications with high memory requirements might fail to connect to WebSphere MQ with reason code 2102: MQRC_RESOURCE_PROBLEM. The following application types might see this error:

- Programs running in 32-bit Java virtual machines
- Programs using the large or very large memory models
- Programs connecting to many queue managers or databases
- Programs that attach to shared memory sets on their own

AIX offers an extended shared memory feature for 32-bit processes that allows them to attach more shared memory. To run an application with this feature, export the environment variable EXTSHM=ON before starting your queue managers and your program. The EXTSHM=ON feature prevents this error in most cases, but it is incompatible with programs that use the SHM_SIZE option of the shmctl function.

WebSphere MQ MQI client applications and all 64-bit processes are unaffected by this limitation. They can connect to WebSphere MQ queue managers regardless of whether EXTSHM has been set.

WebSphere MQ and UNIX Process Priority

Good practices when setting process priority *nice* values.

This information applies to WebSphere MQ running on UNIX and Linux systems only.

If you run a process in the background, that process can be given a higher *nice* value (and hence lower priority) by the invoking shell. This might have general WebSphere MQ performance implications. In highly-stressed situations, if there are many ready-to-run threads at a higher priority and some at a lower priority, operating system scheduling characteristics can deprive the lower priority threads of processor time.

It is good practice that independently started processes associated with queue managers, such as `runmqlsr`, have the same *nice* values as the queue manager they are associated with. Ensure the shell does not assign a higher *nice* value to these background processes. For example, in ksh, use the setting "`set +o bgnice`" to stop ksh from raising the *nice* value of background processes. You can verify the *nice* values of running processes by examining the *NI* column of a "ps -efl" listing.

Also, start WebSphere MQ application processes with the same *nice* value as the queue manager. If they run with different *nice* values, an application thread might block a queue manager thread, or vice versa, causing performance to degrade.

Planning your IBM WebSphere MQ client environment on HP Integrity NonStop Server

When you are planning your IBM WebSphere MQ environment, you must consider the HP Integrity NonStop Server environment, and HP NonStop TMF. Use the information to plan the environment where IBM WebSphere MQ client for HP Integrity NonStop Server runs.

Before you plan your IBM WebSphere MQ client for HP Integrity NonStop Server architecture, familiarize yourself with the basic IBM WebSphere MQ client for HP Integrity NonStop Server concepts, see the topics in IBM WebSphere MQ client for HP Integrity NonStop Server technical overview.

Preparing the HP Integrity NonStop Server environment

Before installation, the environment must be prepared depending on whether the installation is to be verified immediately or not.

For the installation, you require the following items:

- A user ID that meets the requirements. For details about user ID requirements, see Setting up the user and group on HP Integrity NonStop Server.
- Verified locations in the OSS and Guardian file systems that can be for the installation files.
- An operational OSS shell and OSS file system. You can verify the file system by doing the following tasks:
  - Log on to the OSS environment (shell). Ensure that you have write access to the OSS installation root directory you intend to use.
  - Log on to the TACL environment using the user ID in the MQM group. Verify that the volume you intend to use meets the requirements and is accessible to you, and that the subvolume does not exist.

  You can login to both OSS or TACL using either an alias, if you have one, or your full principal.

If you intend to proceed immediately to verify that the installation is usable, you might also need the following optional items:

- An operational and accessible Local Sockets subsystem in the OSS environment.
- An operational TCP/IP subsystem.

If you intend to use TMF coordinated global units of work, you will need the following items:

- An operational TMF subsystem.
- An operational Pathway (TS/MP) subsystem.

Work with your systems administrator if you are in any doubt about the status of these critical subsystems.

IBM WebSphere MQ and HP NonStop TMF

IBM WebSphere MQ client for HP Integrity NonStop Server can participate in HP NonStop Transaction Management Facility (HP NonStop TMF) coordinated units of work. Coordinating transactions with HP NonStop TMF is only supported where the queue manager is at IBM WebSphere MQ Version 7.1 or later.

The IBM WebSphere MQ provided TMF/Gateway converts transactions from TMF coordination into eXtended Architecture (XA) transaction coordination to communicate with the remote queue manager. The IBM WebSphere MQ provided TMF/Gateway is the bridge between TMF and queue manager transactions, using the services provided by HP NonStop TMF, and has been designed to run in a Pathway environment.

HP NonStop TMF software provides transaction protection and database consistency in demanding environments. For more information about HP NonStop TMF, see HP NonStop TMF Introduction.

For information about how to configure the IBM WebSphere MQ provided TMF/Gateway, see Configuring HP Integrity NonStop Server.

Using HP NonStop TMF

The HP NonStop Transaction Management Facility (TMF) is the native transaction manager on HP Integrity NonStop Server and is integrated with the file system and the relational database managers, SQL/MP, and SQL/MX.

IBM WebSphere MQ client for HP Integrity NonStop Server can use TMF to coordinate global units of work.

To coordinate global units of work, TMF acts as the transaction manager, and an application must use the API provided by TMF to start, commit, and back out global units of work. An application starts a global unit of work by calling BEGINTRANSACTION, and then updates IBM WebSphere MQ resources within the global unit of work by issuing MQPUT, MQPUT1, and MQGET calls within syncpoint control. The application can then commit the global unit of work by calling ENDTRANSACTION, or back it out by calling ABORTTRANSACTION.

An application that is using TMF transactions can only actively work on one transaction at any one time, however using RESUMETRANSACTION allows an application to switch from one active transaction to another, or to being associated with no TMF transaction, without completing or aborting the previously active transaction. Any calls to MQPUT, MQPUT1, or MQGET are made under the currently active TMF transaction, if present, or a local unit of work, if not present. Therefore, care must be taken within the application to ensure that these calls are being made within the correct unit of work.

Within a global unit of work, as well as updating IBM WebSphere MQ resources, an application can update Enscribe files, SQL/MP databases, or SQL/MX databases.

Using global units of work

A global unit of work is implemented as a TMF transaction. An application starts a global unit of work by calling BEGINTRANSACTION, and either commits the unit of work by calling ENDTRANSACTION or backs out the unit of work by calling ABORTTRANSACTION. An application can use other TMF API calls as well.

An application can inherit a TMF transaction from another application. For example, an application (the first application) can perform work within the transaction before replying and passing the transaction back to a second application for further processing. Both the first and the second applications can therefore participate in the same global unit of work that involves updates to IBM WebSphere MQ queues and updates to files and databases. The ability to pass a TMF transaction between applications means that several IBM WebSphere MQ applications can perform messaging operations within the same global unit of work.

An application can manage and control multiple active TMF transactions at the same time. The transactions can be started by the application itself, or inherited from other applications, or both. This means that an application can participate in multiple global units of work at the same time.

The maximum number of concurrent active TMF transactions per process is 1000, which is an architectural limit. If an application is managing multiple TMF transactions, only one transaction can be current at any point in time. Alternatively, none of the transactions can be current. The application can use TMF API calls such as RESUMETRANSACTION, ACTIVATERECEIVETRANSID, and TMF_SET_TX_ID to move the state of being current from one transaction to another, or to designate that no transaction is current. The application uses this level of control to determine whether a messaging operation is performed within a local unit of work, a global unit of work, or outside of syncpoint control:

- If an application calls MQPUT, MQPUT1, or MQGET within syncpoint control when no TMF transaction is current, IBM WebSphere MQ processes the call within a local unit of work.
- If an application calls MQPUT, MQPUT1, or MQGET within syncpoint control when the application has a current TMF transaction, IBM WebSphere MQ processes the call within the global unit of work that is implemented by the current TMF transaction.

- If an application calls MQPUT, MQPUT1, or MQGET outside of syncpoint control, IBM WebSphere MQ processes the call outside of syncpoint control, irrespective of whether the application has a current TMF transaction at the time of the call.

IBM WebSphere MQ never changes the state of an application's TMF transaction during an MQI call, except when a software or hardware failure occurs during processing and IBM WebSphere MQ or the operating system determines that the transaction must be backed out to preserve data integrity. Every MQI call restores the transaction state of the application just before returning control to the application.

Avoiding long running transactions

Avoid designing applications in which TMF transactions remain active for more than a few tens of seconds. Long running transactions can cause the circular audit trail of TMF to fill up. Because TMF is a critical system-wide resource, TMF protects itself by backing out application transactions that are active for too long.

Suppose that the processing within an application is driven by getting messages from a queue, and that the application gets a message from the queue and processes the message within a unit of work. Typically, an application calls MQGET with the wait option and within syncpoint control to get a message from the queue.

If the application is using a global unit of work instead, the specified wait interval on the MQGET call must be short to avoid a long running transaction. This means that the application might need to issue the MQGET call more than once before it retrieves a message.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document 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 document. The furnishing of this document does not grant 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:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, 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 publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

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 Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

Programming interface information, if provided, is intended to help you create application software for use with this program.

This book contains information on intended programming interfaces that allow the customer to write programs to obtain the services of IBM WebSphere MQ.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Important: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks

IBM, the IBM logo, ibm.com®, are trademarks of IBM Corporation, registered in many jurisdictions worldwide. A current list of IBM trademarks is available on the Web at "Copyright and trademark information"www.ibm.com/legal/copytrade.shtml. Other product and service names might be trademarks of IBM or other companies.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

This product includes software developed by the Eclipse Project (http://www.eclipse.org/).

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Part Number:

(1P) P/N: