

Publish/Subscribe User's Guide

Contents

1. [What's new in publish/subscribe for Version 7?](#)
 - 1.1. [Benefits of new function](#)
2. [Introduction to publish/subscribe](#)
 - 2.1. [Overview of publish/subscribe components](#)
 - 2.2. [Example of a single queue manager configuration](#)
 - 2.3. [Publishers and publications](#)
 - 2.3.1. [State and event information](#)
 - 2.3.2. [Retained publications](#)
 - 2.4. [Subscribers and subscriptions](#)
 - 2.4.1. [Managed queues](#)
 - 2.4.2. [Subscription durability](#)
 - 2.4.3. [Selection strings](#)
 - 2.5. [Topics](#)
 - 2.5.1. [Topic strings](#)
 - 2.5.1.1. [Wildcard schemes](#)
 - 2.5.1.1.1. [Topic-based wildcard scheme](#)
 - 2.5.1.1.2. [Character-based wildcard scheme](#)
 - 2.5.2. [Constructing topic names](#)
 - 2.5.3. [Topic Trees](#)
 - 2.5.3.1. [Reducing the number of unwanted topics](#)
 - 2.5.4. [Administrative topic objects](#)
 - 2.5.4.1. [SYSTEM.BASE.TOPIC](#)
3. [Distributed publish/subscribe](#)
 - 3.1. [How does it work?](#)
 - 3.1.1. [Proxy subscription aggregation and publication aggregation](#)
 - 3.1.2. [More on routing mechanisms](#)
 - 3.1.3. [Wild card rules](#)
 - 3.2. [Publish/subscribe topologies](#)
 - 3.2.1. [Clusters](#)
 - 3.2.1.1. [Cluster topics](#)
 - 3.2.1.1.1. [Multiple cluster topic definitions](#)
 - 3.2.1.1.2. [Cluster topic performance](#)
 - 3.2.1.2. [Key roles for queue managers in publish/subscribe clusters](#)
 - 3.2.1.3. [Overlapping cluster support](#)
 - 3.2.1.4. [Subscription scope and publication scope in clusters](#)
 - 3.2.1.5. [REFRESH CLUSTER considerations](#)
 - 3.2.2. [Publish/subscribe hierarchies](#)
 - 3.2.2.1. [Connecting a queue manager to a hierarchy](#)
 - 3.2.2.2. [Disconnecting a queue manager from a hierarchy](#)
 - 3.3. [Controlling the flow of publications and subscriptions](#)
 - 3.3.1. [Publication scope](#)
 - 3.3.2. [Subscription scope](#)
 - 3.3.3. [Combining publication and subscription scopes](#)
 - 3.3.4. [Topic spaces](#)
 - 3.3.5. [Combining topic spaces](#)
 - 3.3.5.1. [Create a single topic space in a publish/subscribe cluster](#)
 - 3.3.5.2. [Add a version 7 queue manager to existing version 6 topic spaces](#)
 - 3.3.5.3. [Combine the topic spaces of multiple clusters](#)
 - 3.3.5.4. [Combine and isolate topic spaces in multiple clusters](#)
 - 3.3.5.5. [Publish and subscribe to topic spaces in multiple clusters](#)
 - 3.3.6. [Overlapping topics](#)
 - 3.4. [Loop detection](#)
 - 3.4.1. [Loop detection format](#)
 - 3.5. [Retained publications in a distributed publish/subscribe topology](#)
 - 3.6. [Distributed publish/subscribe security](#)
 - 3.7. [Distributed publish/subscribe system queues](#)
 - 3.7.1. [System queue errors](#)
4. [Writing publish/subscribe applications](#)
 - 4.1. [Writing publisher applications](#)
 - 4.1.1. [Example 1: Publisher to a fixed topic](#)
 - 4.1.2. [Example 2: Publisher to a variable topic](#)
 - 4.2. [Writing subscriber applications](#)
 - 4.2.1. [Example 1: MQ Publication consumer](#)
 - 4.2.2. [Example 2: Managed MQ subscriber](#)
 - 4.2.3. [Example 3: Unmanaged MQ subscriber](#)
 - 4.3. [Publish/subscribe lifecycles](#)
 - 4.4. [Publish/subscribe message properties](#)
 - 4.5. [Message ordering](#)
 - 4.6. [Intercepting publications](#)
 - 4.6.1. [Subscription levels](#)
 - 4.6.2. [Publish exit – MQ PUBLISH_EXIT](#)
 - 4.6.2.1. [MOPSPX - Publish exit data structure](#)
 - 4.6.2.2. [MOPBC - Publication context data structure](#)
 - 4.6.2.3. [MQSBC - Subscription context data structure](#)
 - 4.6.2.4. [Publish exit configuration](#)
 - 4.7. [Publishing options](#)
 - 4.8. [Subscription options](#)
 - 4.8.1. [Subscriptions and message persistence](#)
 - 4.8.2. [Subscriptions and retained publications](#)
 - 4.8.3. [Grouping subscriptions](#)
5. [Publish/subscribe security](#)
 - 5.1. [Example publish/subscribe security setup](#)
 - 5.1.1. [Grant access to subscribe to a topic](#)
 - 5.1.2. [Grant access to subscribe to a topic deeper within the tree](#)
 - 5.1.3. [Grant access to another user to subscribe to a topic deeper within the tree](#)
 - 5.1.4. [Grant access control to avoid additional messages](#)
 - 5.1.5. [Grant access to publish to a topic](#)
 - 5.1.6. [Grant access to publish to a topic deeper within the tree](#)
 - 5.1.7. [Grant access to publish/subscribe](#)
 - 5.2. [Subscription security](#)
 - 5.2.1. [MQSO_ANY_USERID](#)
6. [Queued publish/subscribe compatibility](#)
 - 6.1. [Streams and topics](#)

- 6.2. [Subscription points and topics](#)
- 6.3. [Coexistence with queued publish/subscribe](#)
- 6.4. [Interoperation with queued publish/subscribe](#)
 - 6.4.1. [Heterogeneous broker topologies](#)
- 6.5. [Controlling queued publish/subscribe](#)
 - 6.5.1. [Setting publish/subscribe message attributes](#)
 - 6.5.2. [Starting queued publish/subscribe](#)
 - 6.5.3. [Stopping queued publish/subscribe](#)
 - 6.5.4. [Adding a stream](#)
 - 6.5.5. [Deleting a stream](#)
 - 6.5.6. [Adding a subscription point](#)
 - 6.5.7. [Connecting a queue manager to a hierarchy](#)
 - 6.5.8. [Disconnecting a queue manager from a hierarchy](#)
- 6.6. [MORFH2 header](#)
 - 6.6.1. [MORFH2 structure](#)
 - 6.6.2. [Message service folders](#)
- 6.7. [Command messages](#)
 - 6.7.1. [Delete Publication message](#)
 - 6.7.2. [Deregister Subscriber message](#)
 - 6.7.3. [Publish message](#)
 - 6.7.4. [Register Subscriber message](#)
 - 6.7.5. [Request Update message](#)
 - 6.7.6. [Queue Manager Response message](#)
 - 6.7.7. [Reason codes](#)
 - 6.7.8. [MQMD settings in command messages to the queue manager](#)
 - 6.7.9. [MQMD settings for publications forwarded by a queue manager](#)
 - 6.7.10. [MQMD settings in queue manager response messages](#)
- 7. [WebSphere MQ V6 publish/subscribe migration](#)
 - 7.1. [strmqbrk \(Migrate WebSphere MQ Version 6.0 broker to Version 7.0\)](#)
 - 7.2. [Publish/subscribe command messages](#)
 - 7.2.1. [Delete publication](#)
 - 7.2.2. [Deregister publisher](#)
 - 7.2.3. [Deregister subscriber](#)
 - 7.2.4. [Publish](#)
 - 7.2.5. [Register publisher](#)
 - 7.2.6. [Register subscriber](#)
 - 7.2.7. [Request update](#)
 - 7.3. [New queue manager attributes for publish/subscribe](#)
 - 7.4. [WebSphere MQ publish/subscribe topology migration](#)
 - 7.4.1. [Migrate a WebSphere MQ version 6 publish/subscribe hierarchy to a version 7 hierarchy](#)
 - 7.4.2. [Convert a WebSphere MQ version 6 publish/subscribe hierarchy to a version 7 publish/subscribe cluster](#)
 - 7.5. [Differences from WebSphere MQ Version 6 publish/subscribe](#)
 - 7.6. [Using publish/subscribe with WebSphere MQ classes for JMS](#)
 - 7.6.1. [Subscription name migration on a JMS client](#)
 - 7.7. [Implications of mapping an alias queue to a topic object](#)
- 8. [WebSphere Event Broker V6 or WebSphere Message Broker V6.0 or V6.1 publish/subscribe migration](#)
 - 8.1. [Access Control List \(ACL\) migration](#)
 - 8.2. [Migrating publish/subscribe from WebSphere Event Broker V6](#)
 - 8.3. [Migrating publish/subscribe from WebSphere Message Broker V6 and upgrading to WebSphere Message Broker V7](#)
 - 8.4. [Publish/subscribe migration log file](#)
 - 8.5. [Differences from WebSphere Message Broker V6 publish/subscribe](#)
 - 8.6. [Migrating a WebSphere Event Broker publish/subscribe collective to a WebSphere MQ publish/subscribe cluster](#)
 - 8.7. [Retained publications with headers in MORFH format](#)

Publish/Subscribe User's Guide

Notices

[What's new in publish/subscribe in WebSphere MQ Version 7.0?](#)

[Introduction to 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.

[Distributed publish/subscribe](#)

This section discusses how publish/subscribe messaging can be performed between queue managers, and the 2 different queue manager topologies that can be used to connect queue managers, clusters and hierarchies.

[▶Writing publish/subscribe applications◀](#)

Start writing publish/subscribe WebSphere MQ applications. We assume you have already written point to point WebSphere MQ applications before.

[Publish/subscribe security](#)

The components and interactions that are involved in publish/subscribe are described as an introduction to the more detailed explanations and examples that follow.

[▶Queued publish/subscribe compatibility◀](#)

WebSphere® MQ version 6 publish/subscribe and WebSphere Message and Event Broker version 6 publish/subscribe coexist and interoperate with WebSphere MQ version 7 publish/subscribe, with some restrictions. WebSphere MQ version 6 publish/subscribe is deprecated in WebSphere MQ version 7.

[WebSphere MQ version 6 publish/subscribe migration](#)

Publish/subscribe function in WebSphere MQ Version 7.0 is performed by the queue manager, rather than by a separate publish/subscribe broker. When you migrate your WebSphere MQ Version 6 systems to WebSphere MQ Version 7.0, publish/subscribe function is not automatically migrated. You must upgrade publish/subscribe information to WebSphere MQ Version 7.0 separately.

[WebSphere Event Broker Version 6.0 or WebSphere Message Broker Version 6.0 or 6.1 publish/subscribe migration to WebSphere MQ Version 7.0.1 and later versions](#)

You can migrate publish/subscribe configuration data from WebSphere Event Broker Version 6.0 or WebSphere Message Broker Version 6.0 or 6.1 to a WebSphere MQ Version 7.0.1 or later queue manager.

 This build: January 26, 2011 10:56:16

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps10120_

1. What's new in publish/subscribe in WebSphere MQ Version 7.0?

Publish/subscribe has been changed significantly for WebSphere® MQ Version 7.0. In previous versions, publish/subscribe messaging was controlled using a command message interface. This interface is deprecated in Version 7.0 publish/subscribe. Instead, publish/subscribe messaging is now controlled using new function in the WebSphere MQ API and as a result, publish/subscribe messaging is much more consistent with point-to-point messaging. This new way of doing publish/subscribe messaging is documented in the main body of this section of the information center.

Applications written using previous versions of WebSphere MQ publish/subscribe that make use of the command message interface are encouraged to move to the new WebSphere MQ publish/subscribe API – however, the command message interface continues to be supported on all platforms (including z/OS®). If you are already a user of publish/subscribe you can continue to use your current configuration after installing WebSphere MQ Version 7.0 without making extensive changes to your applications or configuration.

Similarly, JMS applications do not have to be modified, although if you do not chose to use the new Version 7.0 publish/subscribe you will not benefit from the simplified administration that is now available when using WebSphere MQ as the provider. Since the command message interface method of doing publish/subscribe is still supported in Version 7.0 using the PSMODE function, this interface continues to be documented in the WebSphere MQ Version 7.0 library. This information is grouped together here [Queued publish/subscribe compatibility](#).

Benefits of WebSphere Version 7.0 publish/subscribe

Publish/subscribe messaging is now performed using the WebSphere MQ API, there are a number of benefits of using this method.

Parent topic: [Publish/Subscribe User's Guide](#)

This build: January 26, 2011 10:56:56

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps22330_

1.1. Benefits of WebSphere Version 7.0 publish/subscribe

Publish/subscribe messaging is now performed using the WebSphere® MQ API, there are a number of benefits of using this method.

Benefits of the new method of publish/subscribe include:

- In WebSphere MQ Version 7.0, support has been added for publish/subscribe messaging on z/OS®.
- To perform some WebSphere MQ publish/subscribe functions in previous versions you required WebSphere Event Broker, WebSphere Message Broker or the MA0C WebSphere MQ SupportPac (if you were using WebSphere MQ Version 5.3), none of these applications are required now unless you want to route messages according to their content, in which case you can use WebSphere MQ in combination with WebSphere Message Broker.
- In WebSphere MQ Version 6.0, if you were using WebSphere MQ publish/subscribe you had to use PCFs or RFH1 headers, this is no longer the case.
- In WebSphere MQ Version 6.0 if you were using WebSphere Event Broker or WebSphere Message Broker you had to use RFH1 or RFH2 headers, this is no longer the case.
- In WebSphere MQ Version 7.0, the way you publish and subscribe is consistent with the rest of the WebSphere MQ API, making publish/subscribe more intuitive and easier to use.
- Native support for non-durable subscriptions has been added, allowing unconsumed messages and unnecessary subscriptions to be cleaned up at disconnection. This removes the need that existed in WebSphere MQ Version 6.0 for JMS to tidy up non-durable subscriptions in order to meet JMS specification requirements.
- By using non-durable subscriptions with JMS publish/subscribe messaging performance can be improved and resource usage made more efficient.
- The interlock between JMS and the queue manager is improved by using the new WebSphere MQ publish/subscribe API (rather than a command message interface).

Parent topic: [What's new in publish/subscribe in WebSphere MQ Version 7.0?](#)

This build: January 26, 2011 10:56:58

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps22890_

2. Introduction to WebSphere MQ publish/subscribe messaging

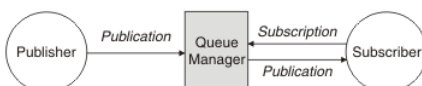
Publish/subscribe messaging allows you to decouple the provider of information, from the consumers of that information. The sending application and receiving application do not need to know anything about each other for the information to be sent and received.

Before a point-to-point 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.

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 WebSphere MQ message, containing the information that it wants, and assign it a topic, that denotes the subject of the information, and let 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 1](#) 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 1. 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.

Example of a single queue manager publish/subscribe configuration**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.


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.

Topics

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

Parent topic: [Publish/Subscribe User's Guide](#)

 This build: January 26, 2011 10:56:46

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps20010_

2.1. 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.

Parent topic: [Introduction to WebSphere MQ publish/subscribe messaging](#)

 This build: January 26, 2011 10:56:54

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps22220_

2.2. Example of a single queue manager publish/subscribe configuration

[Figure 1](#) 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:

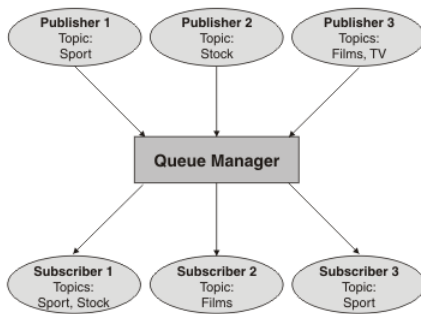
- Publisher 1 is publishing information about sports results using a topic of Sport
- Publisher 2 is publishing information about stock prices using a topic of Stock
- Publisher 3 is publishing information about film reviews using a topic of Films, and about television listings using a topic of TV

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 1. Single queue manager publish/subscribe example. This shows the relationship between publishers, subscribers, and queue managers.



Parent topic: [Introduction to WebSphere MQ publish/subscribe messaging](#)

This build: January 26, 2011 10:56:16

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps10270_

2.3. 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.

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.

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 should be retained so that it can be sent to future subscribers who register an interest in the topic.

Parent topic: [Introduction to WebSphere MQ publish/subscribe messaging](#)

This build: January 26, 2011 10:56:56

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps22270_

2.3.1. 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 up, 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.

Parent topic: [Publishers and publications](#)

This build: January 26, 2011 10:56:56

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps22300_

2.3.2. 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 should be 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 re-published).

The queue manager can retain only one publication for each topic, so a topic's existing retained publication is deleted when a new retained publication arrives at the queue manager. However, the deletion of the existing publication may 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 will not allow, new subscriptions, there is no need to retain publications because they will be delivered to the complete set of subscribers the first time they are published.
- If publications occur very 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 quite large, you could 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 will not be published and the call will fail with MQRC_PUT_NOT_RETAINED.

If a message is a retained publication this will be indicated by the MQIsRetained message property.

Parent topic: [Publishers and publications](#)

 This build: January 26, 2011 10:56:56

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps22310_

2.4. 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 SBSTATUS command.

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).

Managed queues and publish/subscribe

When you create a subscription you can choose to use managed queuing. If you use managed queuing 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.


Subscription durability

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

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.

Parent topic: [Introduction to WebSphere MQ publish/subscribe messaging](#)

 This build: January 26, 2011 10:56:55

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps22260_

2.4.1. Managed queues and publish/subscribe

When you create a subscription you can choose to use managed queuing. If you use managed queuing 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.

Parent topic: [Subscribers and subscriptions](#)

 This build: January 26, 2011 10:56:57

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps22750_

2.4.2. Subscription durability

Subscriptions can be configured to be durable or non-durable. Subscription durability determines what happens to subscriptions when subscribing applications disconnect for 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 will be set to the original expiry of the subscription and not the remaining expiry time.

A queue manager will continue to send publications to satisfy a durable subscription even if that subscriber application is not connected. This will lead 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 [MQSO_PUBLICATIONS_ON_REQUEST](#) 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 will no longer be shown if you display subscriptions using the DISPLAY SBSTATUS command. No more messages will be 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, will be used by the queue manager. Subscription names must be unique within the queue manager so that it can be used to identify a subscription.

Parent topic: [Subscribers and subscriptions](#)

 This build: January 26, 2011 10:56:54

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps22060_

2.4.3. 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.

Parent topic: [Subscribers and subscriptions](#)

Related information

[Selectors](#)

 This build: January 26, 2011 10:56:54

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps22180_

2.5. 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 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](#), 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](#).

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](#) 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 haven't defined any parent topics at all, it inherits from `SYSTEM.BASE.TOPIC`. Administrative topic objects are described in [Administrative topic objects](#).

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 instance, 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 or referring to a topic object. You can also do both, in which case the topic string you supply defines a subtopic. The queue manager identifies the topic by adding the subtopic to the "master" topic named in the topic object, inserting an additional '/' in between the two topic strings. [Constructing topic names](#) 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.

[Constructing topic names](#)

A topic is constructed from the subtopic identified in a topic object, and a subtopic provided by an application. You can use either subtopic as the topic name, or combine them to form a new topic name.


[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.

[Administrative topic objects](#)

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

Parent topic: [Introduction to WebSphere MQ publish/subscribe messaging](#)

 This build: January 26, 2011 10:56:54

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps22230_

2.5.1. 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.

[Syntax diagram format](#) Railroad diagram Dotted decimal
'\n';break; default:document.write('

'\n'


```
+ 'Topic string\n'
+ '\n'
+ '-----, \n'
+ ' V | \n'
+ '>>-----<<\n'
+ '\n'
+ '
```

```
);} //]]>
```

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](#).

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](#) 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.

Parent topic: [Topics](#)

 This build: January 26, 2011 10:56:55

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
>#22231_

2.5.1.1. 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.

MQSO_WILDCARD_CHAR

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

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.


[Character-based wildcard scheme](#)

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

Parent topic: [Topic strings](#)

 This build: January 26, 2011 10:56:55

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
>#22232_

2.5.1.1.1. 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 shown above, 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](#) 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](#) 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](#) 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.

[Syntax diagram format](#) Railroad diagram Dotted decimal

```
);break; default:document.write('
```

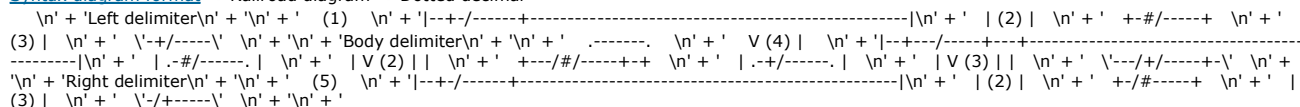


```
); //]]>
```

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.

[Syntax diagram format](#) Railroad diagram Dotted decimal



```
); //]]>
```

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.

- 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:

- level0/level1/+/level4/#
- 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
```

Parent topic: [Wildcard schemes](#)

 This build: January 26, 2011 10:56:55

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps22234_

2.5.1.1.2. 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
.-----
V        |
>>--+| V6 literal |-----<<
  +-§--+*-----+
  |  +?-----+ |
  |  | (1) | |
  |  '-§-----' |
  |  (2) | |
  +-*-----+
  |  (3) | |
  | -?-----' |

V6 literal
|----Any Unicode character except *, ? and §-----|
```


Notes:

- Means "Escape the following character", so that it is treated as a literal. '§' must be followed by either '*', '?' or '§'. See [Examples of topic strings](#).
- Means "Match zero or more characters" in a subscription.
- Means "Match exactly one character" in a subscription.

Examples

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

Parent topic: [Wildcard schemes](#)

 This build: January 26, 2011 10:56:55

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps22233_

2.5.2. Constructing topic names

A topic is constructed from the subtopic identified in a topic object, and a subtopic provided by an application. You can use either subtopic as the topic name, or combine them to form a new topic name.

In an MQI program the full topic name is created by MQOPEN. It is composed of two fields:

1. The **TOPICSTR** attribute of the topic object.
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 neither a blank nor a 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_TOPIC_STRING_ERROR**.

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

Table 1 shows examples of topic string concatenation:

Table 1. Topic string concatenation examples

TOPICSTR	ObjectString	Full topic name	Comment
Football/Scores	' '	Football/Scores	The TOPICSTR is used alone
' '	Football/Scores	Football/Scores	The ObjectString 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
/Football	Scores	/Football/Scores	The topic starts with an 'empty node'

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);
```

Parent topic: [Topics](#)

This build: January 26, 2011 10:56:57

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
 This topic's URL:
 ps22780_

2.5.3. 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.

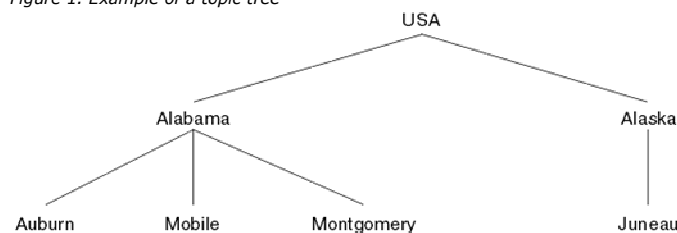
Although you can use any character string to define a topic's topic string, it is advisable to choose a topic string that fits into a hierarchical tree structure. Thoughtful design of topic strings and topic trees can help you with the following operations:

- 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.

Figure 1 shows an example of a topic tree with one root topic.

Figure 1. 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 1 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?

Parent topic: [Topics](#)

 This build: January 26, 2011 10:56:53

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps22040_



2.5.3.1. 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, with an exceptionally large number of topics that 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?](#) 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.

The section, [How does a large and growing number of unwanted topics arise?](#), discusses how designs that use unique identifiers in topic strings, and wildcard subscriptions, can lead to an unnecessarily large and growing number of topics.

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) ACTCON`.

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) ACTCON`.

Note: A subscription that contains a wildcard prevents all child nodes that match the wildcard being flagged as unused. Nodes are effectively marked "in use" by a wildcard, and the nodes are not removed when the queue manager periodically removes unused nodes.

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

How does a large and growing number of unwanted topics arise?

Cases of large numbers of topics leading to performance problems are typically associated with using wildcard subscriptions. A wildcard subscription prevents potentially many topics being flagged as unused, and so the topics remain in the topic tree, even if they are never used again.

Your solution design might use a wildcard subscription that persists for a long time. The longevity of the subscription might be either because the subscription is durable, or because the application keeps the subscription open for a long period. During this time, new publications match the subscription and might create new topics. Even if the topics are not used again, the wildcard subscription prevents the topics being flagged as unused.

The longevity of the subscription, the use of wildcards, and publications on new topics, are not problems in themselves. A problem only arises if the pattern of the publications leads to a continually increasing number of nodes that are not referred to again.

An example of a publication pattern that leads to many unused topics is to publish unique data in each topic string. For example, each publication might include a timestamp in the topic string. Subscribers can match the topic by using a wildcard in the subscription in place of the timestamp. When a publication is received, the subscriber gets its timestamp from the topic string. The topic is never used again, but because of the wildcard in the subscription, the topic remains in the topic space.

An alternative design to eliminate the topic is to use message properties, rather than the topic string, to carry unique data associated with a publication.

Use of unique identification in a publication topic string is not always a problem, but it is indicative. It does not necessarily lead to an increasing number of topics that are never used again. For example, it is common to distribute information to subscribers based on using the identifier of the subscriber in the topic string. There is one topic for each subscriber.

If the identifiers of the subscribers are permanent, and reused, the number of topics in the topic tree is unlikely to grow without bound. But if the identification is temporary, like a session-cookie that is never used again after a short period of use, then the number of topics in the topic tree might well grow in an unlimited way if the topics are not removed. How might it happen that topics associated with temporary subscribers are not removed?

The thinking of the designer might be as follows. A web application has registered and guest users. The registered users have durable subscriptions that do not expire. A guest subscription is set to expire some time after the guest session expires. The system is sized to manage the subscriptions of registered users, and perhaps ten times that number of subscriptions from temporary guest users at any one time. But once a guest user has not interacted for a time, the subscription expires. Once a guest subscription expires, the associated topic nodes are no longer wanted by that guest, and with no subscription using the nodes, the nodes become unused and are removed by the queue manager. So in the system, as initially conceived by the designer, there is no problem of a growing number of unwanted nodes.

Suppose requirements change, and an audit log is needed. The design is modified to include an audit log. The audit application uses a wildcard subscription to copy each publication to its log. Suppose in creating the audit log, the designer did not distinguish between logging topics that have been created for registered users, and topics that were created for guests. The audit subscription prevents the topics associated with expired guest subscriptions from being flagged as unused.

The designer might solve the problem by creating different audit subscriptions for the guest users. Corresponding to each guest user subscription might be an audit subscription with the same expiry time as the guest subscription. Now, when the guest user session ends, the final expiry time is set for both subscriptions. At the end of the expiry period the topics are no longer wanted, the subscriptions both expire, and the topics become unused.

In summary, scrutinize carefully any design that includes identification information in a topic string, and a wildcard subscription. It is possible that the design might contain some element that leads to a large and growing number of topics.

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.

Design your publish/subscribe applications so that unwanted nodes become unused nodes, which are removed by the queue manager

A major cause of unwanted nodes is using the topic string to pass publication data to subscribers. Another cause is passing unique identification data, such as a session cookie, that is only used for a short time. If these topic usages are coupled to long-lived wildcard subscriptions, the unwanted nodes do not become unused nodes.

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.

Parent topic: [Topic trees](#)

 This build: January 26, 2011 10:56:54

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps22042_

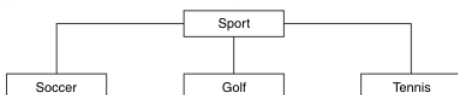


2.5.4. Administrative topic objects

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

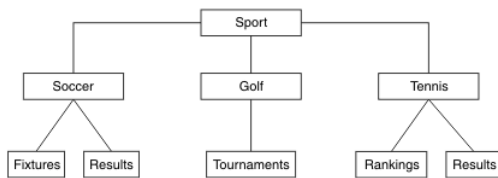
[Figure 1](#) shows how a high-level topic of `Sport` divided into separate topics covering different sports can be visualized as a topic tree:

Figure 1. Visualization of a topic tree



[Figure 2](#) shows how the topic tree can be divided further, to separate different types of information about each sport:

Figure 2. Extended topic tree



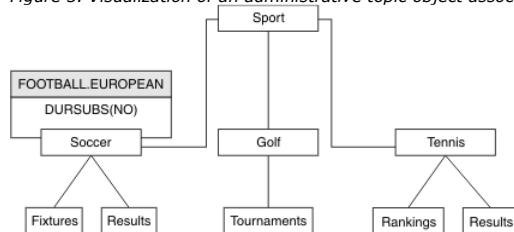
To create the topic tree illustrated, no administrative topic objects need be defined. Each of the nodes in this tree are defined by a topic string created in a publish or subscribe operation. Each topic in the tree inherits its attributes from its parent. Attributes are inherited from the parent topic 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`.

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 `DURSUB` of 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 3. Visualization of an administrative topic object associated with the Sport/Soccer topic



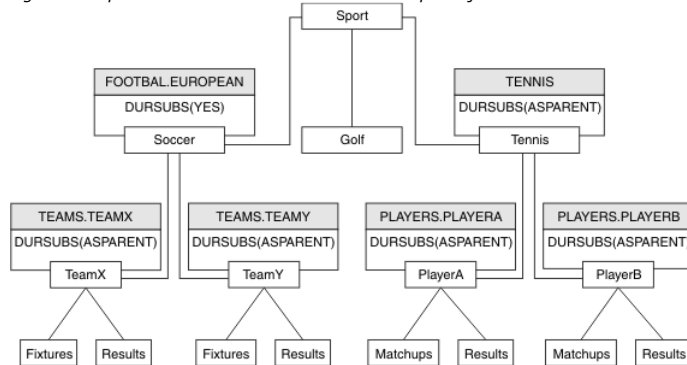
Any applications subscribing to topics beneath Soccer in the tree can still use the topic strings they used before the administrative topic object was added. However, an application can now be written to subscribe using the object name `FOOTBALL.EUROPEAN`, instead of the string `/Sport/Soccer`. For example, 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 4](#):

Figure 4. 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 will receive publications made to this topic and to topic strings more specific than this topic. This is the default value.

BLOCK

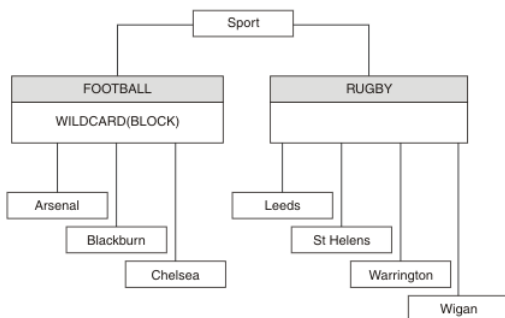
Subscriptions made to a wildcarded topic less specific than the topic string at this topic object will not receive publications made to this topic or to topic strings more specific than this topic.

► This 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 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 reevaluated for existing subscriptions, you must restart the queue manager. ◀



In the example, [Example: Create the Sport publish/subscribe cluster](#), you can follow the steps to create the topic tree structure shown in [Figure 5](#).

Figure 5. 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 9](#). Run the publish command in [Figure 12](#) to create some publications.

Figure 6. Publish to QMA

```
pub QMA
```

The results are shown in [Table 1](#). Notice how setting the WILDCARD property value BLOCK, prevents subscriptions with wildcards from receiving publications to topics within the scope of the wildcard.

Table 1. 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 5](#), 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 topic 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](#), QMB has the same set of subscriptions as QMA, yet QMB received no publications after the publisher published to QMA, see [Figure 6](#). 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, 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 6](#).

[Table 2](#) 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 3](#).

Table 2. 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 3](#) 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 3. 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](#). 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](#). 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 5](#). Cluster `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.

Figure 7. Delete and create topics: `topics.tst`

```
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')
```

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

Create subscriptions with wildcards. The wildcards corresponding to the topics with topic objects in [Figure 5](#). 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.

Figure 8. Create wildcard subscriptions: `wildsubs.tst`

```
DEFINE QLOCAL(QSPORTS) REPLACE
DEFINE QLOCAL(QARSENAL) REPLACE
DEFINE QLOCAL(QSLEEDS) REPLACE
CLEAR QLOCAL(QSPORTS)
CLEAR QLOCAL(QARSENAL)
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(QARSENAL)
DEFINE SUB (SLEEDS) TOPICSTR('Sports/#/Leeds') DEST(QSLEEDS)
```

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.

Figure 9. Delete and create subscriptions: `fullsubs.tst`

```
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)
```

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.

Figure 10. Create queue managers: `qmgrs.bat`

```

@echo off
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

```

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

Figure 11. Update subscriptions: *upsubs.bat*

```

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

```

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**.

Figure 12. Publish: *pub.bat*

```

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

```

SYSTEM.BASE.TOPIC


Base topic for ASPARENT resolution. If a particular topic has no parent administrative topic objects, or those parent objects also have ASPARENT, any remaining ASPARENT attributes are inherited from this object.

Parent topic: [Topics](#)

Related tasks

[Adding a stream](#)

[Adding a subscription point](#)

 This build: January 26, 2011 10:56:26

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps12490_

2.5.4.1. SYSTEM.BASE.TOPIC

Base topic for ASPARENT resolution. If a particular topic has no parent administrative topic objects, or those parent objects also have ASPARENT, any remaining ASPARENT attributes are inherited from this object.

The default values of the SYSTEM.BASE.TOPIC are:

Table 1. Default values of SYSTEM.BASE.TOPIC

Parameter	Value
TOPICSTR	"
DEFPRTY	0
DEFPRESP	SYNC
DEFPERSIST	NO
DESCR	'Base topic for resolving attributes'
DURSUB	YES
MDURMDL	SYSTEM.DURABLE.MODEL.QUEUE
MNDURMDL	SYSTEM.NDURABLE.MODEL.QUEUE
MASTER	YES
NPMMSGDLV	ALLAVAIL
PMSGDLV	ALLDUR
PUB	ENABLE
SUB	ENABLE

If this object does not exist, its default values are still used by WebSphere® MQ for ASPARENT attributes that are not resolved by parent topics further up the topic tree.

Parent topic: [Administrative topic objects](#)

This build: January 26, 2011 10:56:27

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps12495_

3. Distributed publish/subscribe

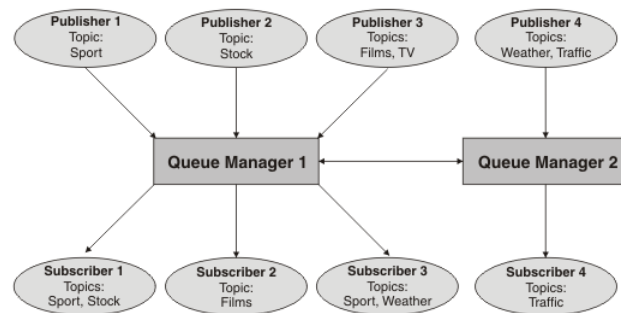
This section discusses how publish/subscribe messaging can be performed between queue managers, and the 2 different queue manager topologies that can be used to connect queue managers, clusters and hierarchies.

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 1](#).

[Figure 1](#) 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 1. 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.

Publish/subscribe topologies

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

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.

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.

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.

Distributed publish/subscribe security

Distributed publish/subscribe internal messages such as proxy subscriptions and publications are put to distributed publish/subscribe system queues (SYSTEM.INTER.QMGR.CONTROL, for example) by the receiving channel 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.

Distributed publish/subscribe system queues

Four system queues are used by queue managers when they do publish/subscribe messaging. You normally need to be aware of their existence only for problem determination or capacity planning purposes.

Parent topic: [Publish/Subscribe User's Guide](#)

This build: January 26, 2011 10:56:17

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps10320_

3.1. 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. A subscriptions flows to all queue managers in the network, where proxy subscriptions are created to ensure that publications get routed back to the queue manager where the subscription was created originally. This is shown in [Figure 1](#).

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. In turn, the queue manager creates subscriptions for these topics with its neighbors, unless a subscription already exists. This is shown in [Figure 2](#).

When an application publishes information, the receiving queue manager forwards it (possibly through one or more intermediate queue managers) using transmission queues to any applications that have valid subscriptions on remote queue managers. This is shown in [Figure 3](#).

Figure 1. Propagation of subscriptions through a queue manager network. 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).

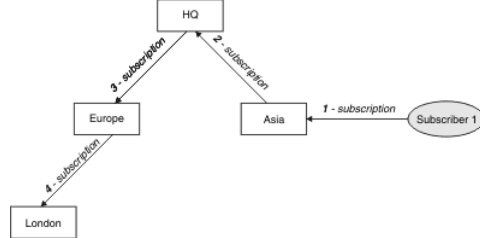


Figure 2. Multiple subscriptions. Subscriber 2 registers a subscription, to the same topic as in [Figure 1](#), on the HQ queue manager (5). The subscription for this topic is forwarded to the Asia queue manager, so that it is aware that subscriptions exist elsewhere on the network (6). The subscription does not have to be forwarded to the Europe queue manager, because a subscription for this topic has already been registered (step 3 in [Figure 1](#)).

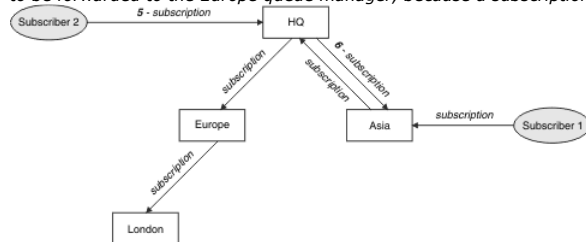
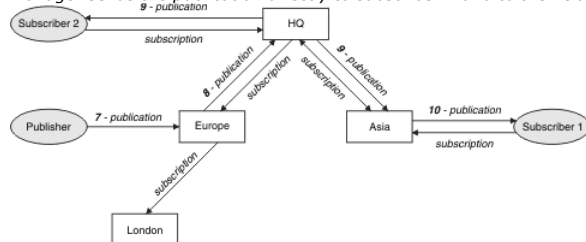


Figure 3. Propagation of publications through a queue manager network. A publisher sends a publication, on the same topic as in [Figure 2](#), to the Europe queue manager (7). A subscription for this topic exists from HQ to Europe, so the publication is forwarded to the HQ queue manager (8). However, 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), from where it is forwarded to subscriber 1 (10).



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

The interconnected nature of publish/subscribe queue managers means that it takes some time for the proxy subscription to propagate around all nodes in the network. The consequence of this is that once a subscription has been made, remote publications are not necessarily received immediately; this can be addressed by using PROXYSUB(FORCE) as described in [More on routing mechanisms](#).

The subscription operation completes when the proxy subscription has been put on the appropriate transmission queue for each directly connected queue manager, and will not include the propagation of the proxy subscription out to the rest of the topology. Proxy subscriptions are associated with the queue manager name that created them. If one queue manager is attached, by a hierarchical connection or as part of a publish/subscribe cluster, to more than one queue manager with the same queue manager name, this can result in publications failing to reach one or all of the identically named remote queue managers. 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.

[More on routing mechanisms](#)

Publish everywhere is an alternative routing mechanism to proxy subscription-forwarding. Publish everywhere works by publishing to all directly connected queue managers regardless of proxy subscriptions. Publish everywhere is not supported in publish/subscribe clusters or hierarchies, but a similar technique is available by using the PROXYSUB attribute for a high-level topic object.

[Wildcard rules](#)

Wildcards in proxy subscriptions are converted to use topic wildcards.

Parent topic: [Distributed publish/subscribe](#)

Related concepts

[Controlling the flow of publications and subscriptions](#)

[Publication scope](#)

[Subscription scope](#)

[Subscription scope and publication scope in publish/subscribe clusters](#)

[Combining publication and subscription scopes](#)

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps15910_

3.1.1. 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.

Proxy subscription aggregation

Proxy subscriptions are aggregated using a simple duplicate elimination system. For a given resolved topic string, a proxy subscription is sent to directly connected publish/subscribe queue managers on the first local subscription or received proxy subscription.

Subsequent subscriptions make use of this existing proxy subscription. The proxy subscription is cancelled only after the last local subscription or received proxy subscription is cancelled.

Note: If PROXYSUB(FORCE) is set, a proxy subscription might be sent before the first local subscription or received proxy subscription, and will not be cancelled even after the last local subscription or received proxy subscription is cancelled.

Publication aggregation

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.

Parent topic: [How does distributed publish/subscribe work?](#)

This build: January 26, 2011 10:56:39

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps15990_

3.1.2. More on routing mechanisms

Publish everywhere is an alternative routing mechanism to proxy subscription-forwarding. Publish everywhere works by publishing to all directly connected queue managers regardless of proxy subscriptions. Publish everywhere is not supported in publish/subscribe clusters or hierarchies, but a similar technique is available by using the PROXYSUB attribute for a high-level topic object.

PROXYSUB attribute for a high-level topic object is explained in the following comparison:

Publish everywhere

If publish everywhere routing is available in a publish/subscribe cluster, there is no need for any proxy subscriptions and all publications are published to every member of the publish/subscribe clusters.

The advantages of publish everywhere are the removal of latency introduced by the propagation of proxy subscriptions, and the removal of the network traffic caused by proxy subscription propagation where the subscription is frequently created and deleted.

Proxy-subscription forwarding

To achieve a similar behavior to publish everywhere, alter the topic object, as follows:

```
ALTER TOPIC ("SYSTEM.BASE.TOPIC") PROXYSUB (FORCE)
```

This forces the sending of a wildcard proxy subscription, for the topic string associated with this topic object, to every directly connected member of the publish/subscribe topology, regardless of whether any local subscriptions have been made.

When this forced proxy subscription has been propagated throughout the topology, any new subscriptions immediately receive any publications from other connected queue managers, without suffering latency. Proxy subscriptions for these new subscriptions are still propagated to each of the directly connected publish/subscribe queue managers; preventing a break in flow of publications if this behavior is turned off later.

Parent topic: [How does distributed publish/subscribe work?](#)

This build: January 26, 2011 10:56:32

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps15970_

3.1.3. 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 will receive publications made to this topic and to topic strings more specific than this topic. This is the default value.

BLOCK

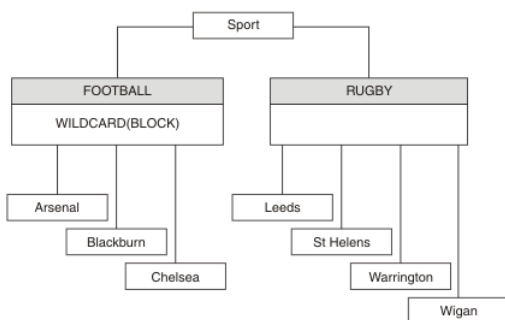
Subscriptions made to a wildcarded topic less specific than the topic string at this topic object will not receive publications made to this topic or to topic strings more specific than this topic.

►This 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 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 reevaluated for existing subscriptions, you must restart the queue manager.◀

◀

In the example, [Example: Create the Sport publish/subscribe cluster](#), you can follow the steps to create the topic tree structure shown in [Figure 5](#).

Figure 1. 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 9](#). Run the publish command in [Figure 12](#) to create some publications.

Figure 2. Publish to QMA

```
pub QMA
```

The results are shown in [Table 1](#). Notice how setting the WILDCARD property value BLOCK, prevents subscriptions with wildcards from receiving publications to topics within the scope of the wildcard.

Table 1. 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 5](#), 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 topic 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](#), QMB has the same set of subscriptions as QMA, yet QMB received no publications after the publisher published to QMA, see [Figure 6](#). Although the Sports/Football and Sports/Rugby topics are cluster topics, the subscriptions defined in [fullsubs.txt](#) 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, 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 6](#).

[Table 2](#) 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 3](#).

Table 2. 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 3](#) 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 3. 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](#). 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](#). 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 5](#). Cluster `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.

Figure 3. Delete and create topics: `topics.tst`

```
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')
```

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 5](#). 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.

Figure 4. Create wildcard subscriptions: `wildsubs.tst`

```
DEFINE QLOCAL(QSPORTS) REPLACE
DEFINE QLOCAL(QARSENAL) REPLACE
DEFINE QLOCAL(QSLEEDS) REPLACE
CLEAR QLOCAL(QSPORTS)
CLEAR QLOCAL(QARSENAL)
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(QARSENAL)
DEFINE SUB (SLEEDS) TOPICSTR('Sports/#/Leeds') DEST(QSLEEDS)
```

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.

Figure 5. Delete and create subscriptions: `fullsubs.tst`

```
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)
```

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.

Figure 6. Create queue managers: *qmgrs.bat*

```
@echo off
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
```

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

Figure 7. Update subscriptions: *upsubs.bat*

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

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**.

Figure 8. Publish: *pub.bat*

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

Parent topic: [How does distributed publish/subscribe work?](#)

This build: January 26, 2011 10:56:33

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps15980_

3.2. 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 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 information center.

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

[Publish/subscribe clusters](#)

You can improve the performance of your publish/subscribe network by arranging your queue managers in a publish/subscribe cluster. A publish/subscribe cluster consists of a set of queue managers connected together, with direct channel links between all members, to form all or part of a publish/subscribe network.

[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.

Parent topic: [Distributed publish/subscribe](#)

This build: January 26, 2011 10:56:56

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps22740_

3.2.1. Publish/subscribe clusters

You can improve the performance of your publish/subscribe network by arranging your queue managers in a publish/subscribe cluster. A publish/subscribe cluster consists of a set of queue managers connected together, with direct channel links between all members, to form all or part of a publish/subscribe network.

A *publish/subscribe cluster* is a set of queue managers that are fully interconnected and form part of a multi-queue manager network for publish/subscribe applications. A cluster that is used for publish/subscribe messaging is no different from a standard 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 connected together by a pair of channels. For information about how to plan and configure a WebSphere MQ cluster refer to [first tasks for clustering](#).

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

- Messages destined for 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.
- There is no single point of failure in this topology. If one queue manager is not available, publications and subscriptions are still able to flow through the rest of the publish/subscribe system because each queue manager is directly connected with each other.
- If your clients are geographically dispersed, you can set up a cluster in each location, and connect the clusters (by joining a single queue manager in each cluster) to optimize the flow of publications and subscriptions through the network.
- You can group clients according to the topics to which they publish and subscribe. Clients that share common topics can connect to queue managers within a cluster. The common publications are transported efficiently within the cluster, because they pass through only queue managers that have at least one client with an interest in those common topics.
- 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 per queue manager can be reduced by adding more queue manager to the cluster to share workload. This makes a publish/subscribe cluster topology highly scalable.

When you create a cluster it is possible to create a loop causing messages to cycle forever within the network, nothing will prevent you from doing this but you will be made aware of it because of the fingerprint that is added by the queue manager (stored as a message property).

A publish/subscribe cluster is created when a clustered topic is defined. This definition is shared with all members of the cluster. This means that publications on the clustered topic are shared with all members of the cluster.

When at least one clustered topic object is defined, all queue managers within the cluster will be notified about each other.

If you have several queue managers in your publish/subscribe system, many channels are required to connect these queue managers together. However, the connections between queue managers can be created automatically to reduce the administrative work load.

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.

Key roles for publish/subscribe cluster queue managers

There are two key roles for queue managers in publish/subscribe clusters; as full repositories and as cluster topic hosts.

Overlapping cluster support and publish/subscribe

With WebSphere MQ clusters, a single queue manager can be a member of more than one cluster. ►Overlapped publish/subscribe clusters result in some unexpected behaviors. Subscriptions are not propagated from one cluster to another, and the publications received by some subscribers might be unexpected. ◀


Subscription scope and publication scope in publish/subscribe clusters

The scope of publications and subscriptions is defined in the cluster topic object.

REFRESH CLUSTER considerations

The REFRESH CLUSTER command can cause temporary disruption to publish/subscribe traffic in a publish/subscribe cluster. We therefore recommend only to run REFRESH CLUSTER command when under the guidance of your IBM Support Center.

Parent topic: [Publish/subscribe topologies](#)

 This build: January 26, 2011 10:56:56

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps22650_



3.2.1.1. 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, the cluster topic is published to the full repositories. The full repositories then push 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.

At each queue manager a single topic space is constructed from the local and cluster topic definitions. When an application subscribes to a topic that resolves to a clustered topic, WebSphere® MQ creates a proxy subscription and sends it from the queue manager to which the subscriber is connected, to all the other members of the cluster. The subscriber receives publications to the cluster topic from publishers connected to any of the queue managers in the cluster.

Multiple cluster topic definitions

A local topic definition overrides a cluster topic definition of the same name, but needs to be used with caution. Create multiple definitions of the same cluster topic only in special circumstances.

Cluster topic performance

The performance characteristics of cluster topics require special consideration as they differ from the performance characteristics of cluster queues, and are potentially a source of performance problems in large or unbalanced clusters. Some performance problems can be remedied, without adding additional servers, by creating multiple overlapping clusters.


Parent topic: [Publish/subscribe clusters](#)

Related concepts

[»Topics«](#)

Related information

[»Reset cluster«](#)

 This build: January 26, 2011 10:56:27

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps15670_



3.2.1.1.1. Multiple cluster topic definitions

A local topic definition overrides a cluster topic definition of the same name, but needs to be used with caution. Create multiple definitions of the same cluster topic only in special circumstances.

If a local and cluster topic definition exists for a single topic string, the local definition is used.

It is acceptable to define two or more cluster topic definitions with identical attributes for a single topic string. Unlike having one local and one cluster topic definition, having multiple definitions of a cluster topic leads to the possibility you might change one of the definitions and not the other, and create a conflict. Consequently, avoid creating multiple cluster topic definitions, except when there is a need to do so, as in the examples below.

Where two or more cluster topic definitions, for a single topic string, have differing attributes or exist in more than one cluster, a message is written to the [»error«](#) log and the most recently received cluster topic definition is used.

Overriding a cluster topic definition locally

You can define a local topic object to override a cluster topic object with the same name and use it to publish only to locally connected subscribers.

You need to specify **PUBSCOPE** (`QMGR`) on the local topic object if you want publisher applications connected to this queue manager to publish only to local subscribers. If **PUBSCOPE** resolves to `(ALL)`, then remote subscribers are also sent publications published to the topic defined on this queue manager.

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

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`. Before you do step [3](#), you need to wait awhile to be sure that the changes have been pushed to all members of the cluster.

1. Duplicate the cluster topic definition from `QM1` on `QM2` with the same attributes.
2. Delete the cluster topic definition from `QM1`.
3. You can now alter the topic definition on `QM2` without conflicting with the definition that was on `QM1`.

Replacing a cluster topic definition on a failed queue manager

Your problem might be that `QM1` cannot be restarted quickly, and therefore you cannot carry out step [2](#) to delete the original cluster topic definition before changing the new topic definition on `QM2`. If you cannot wait for `QM1` to be restarted, what do you do?

The answer is to add the new topic definition to `QM2` anyway. When you restart `QM1` the new cluster topic definition prevails over the old definition, because the new definition is more recent. Errors are written to the error logs of the queue managers that there is a conflicting cluster topic definition. Resolve the error as soon as possible by removing the duplicate cluster topic definition from one of the queue managers.

Parent topic: [Cluster topics](#)

Related information

[»Reset cluster«](#)

 This build: January 26, 2011 10:56:27

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps15672_



3.2.1.1.2. Cluster topic performance

The performance characteristics of cluster topics require special consideration as they differ from the performance characteristics of cluster queues, and are potentially a source of performance problems in large or unbalanced clusters. Some performance problems can be remedied, without adding additional servers, by creating multiple overlapping clusters.

Reducing the impact 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 to manage the cluster. In a typical point-to-point cluster, you can largely discount the performance impact of managing the cluster when considering queue manager performance. In a publish/subscribe cluster you do need to consider the impact of cluster management on queue manager performance, both in its timing, and its size.

To reduce the impact 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 thinking about adding publish/subscribe topics to an existing large cluster, just because the cluster is already there, consider if you can define a much smaller subset of queue managers involved in publish/subscribe and make that an "overlapping" cluster. Although some queue managers are now in two clusters, the overall impact 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 impact 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.

Performance characteristics of publish/subscribe clusters

It is important to consider how changing attributes of a publish/subscribe cluster, such as adding a new queue manager, topic, or subscription to the cluster impacts 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 repositories, and only sent to other cluster members when they first reference a cluster queue.
2. Adding a new 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, or a new subscription created to a cluster topic, the information is pushed to all members of the cluster straightaway, possibly causing channels to be started to each member of the cluster from the full repositories.
2. When a new queue manager joins an existing cluster, it resynchronizes with all members of the cluster, possibly causing channels to be created and started to each member of the cluster from the new queue manager.

In short, 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.

Parent topic: [Cluster topics](#)

 This build: January 26, 2011 10:56:27

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps15674_



3.2.1.2. Key roles for publish/subscribe cluster queue managers

There are two key roles for queue managers in publish/subscribe clusters; as full repositories and as cluster topic hosts.

Note: ▶ You can use a cluster for queued messaging, publish/subscribe messaging, or both. Create a publish/subscribe cluster by creating the first cluster topic object in a cluster. ◀



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 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 out to all the other queue managers in the cluster where it is cached for use by publishers and subscribers running on other queue managers in the cluster.



Availability and management

You should define at least two full repositories in a cluster to improve availability.

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 and to use one of them as a cluster topic host.

In publish/subscribe clusters with many highly available computers you might decide, for performance and management reasons, to host repositories and cluster topic hosts on different computers.

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. In contrast, in publish/subscribe clusters, a clustered topic is *available* on all queue managers in the cluster. 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.

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 happens after 90 days from when the topic definition became unavailable.

With the 90 day period of time 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 90 day period is sufficient to cater for technical problems; the 90 day period is likely to be exceeded only because of administrative errors. To mitigate that possibility, if the cluster topic host is unavailable, every 27 days all members of the cluster write error log messages that their cached cluster topic object has not been refreshed. You should 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 90 days, only its version of the cluster topic object is removed from the other hosts. The other versions of the cluster topic object remain. 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.



Parent topic: [Publish/subscribe clusters](#)

Related concepts

[Selecting queue managers to hold full repositories](#)

This build: January 26, 2011 10:56:28

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps15680_

3.2.1.3. Overlapping cluster support and publish/subscribe

With WebSphere® MQ clusters, a single queue manager can be a member of more than one cluster. ►Overlapped publish/subscribe clusters result in some unexpected behaviors. Subscriptions are not propagated from one cluster to another, and the publications received by some subscribers might be unexpected. ◀

A reason for making a single queue manager a member of more than one cluster is to create a cluster gateway between two clusters, so that messages originating in one cluster can be routed to another cluster. ►Publish/subscribe clusters have inherited the capability of being overlapped from traditional queue manager clusters. ◀

A WebSphere MQ queue manager can be a member of more than one publish/subscribe cluster. Publications are not passed from one cluster to another using the queue manager as a gateway. The scope of proxy subscriptions is limited to the single cluster in which the clustered topic is defined.

►In designing a publish/subscribe cluster, cluster topics are typically defined on one queue manager, called the cluster topic host. Often the cluster topic host is one of the full repositories. Having one cluster topic host avoids the difficulty made by the situation that the same topic might be defined twice on different queue managers in the cluster, with different attributes. ◀

With overlapping publish/subscribe clusters, there are bound to be multiple cluster topic hosts. Queue managers in the overlap receive cluster topic definitions from multiple cluster topic hosts. Having only one cluster topic host avoided the difficulty of dealing with conflicting cluster topic definitions.

It is not possible to choose one queue manager, which is in the overlap, to be the single cluster topic host for multiple clusters. The cluster attribute of a topic definition is a cluster name, and not a cluster namelist.

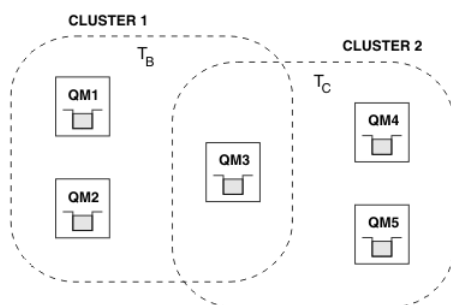
Overlapping clusters, non-overlapping topics

One way to use overlapping clusters with publish/subscribe clustering, is to ensure that the overlapping clusters have non-overlapping topic spaces. You can then use queue managers in the overlap to publish and subscribe to both clusters predictably.

In [Figure 1](#), 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 both clusters. A subscriber connected to QM3 in the overlap is able to subscribe to topics in both clusters. Publishers and subscribers to queue managers that are not in the overlap only publish and subscribe to topics in their cluster.

An alternative way of thinking about [Figure 1](#) 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 both queue managers QM1 and QM2. 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 both queue managers QM4 and QM5.

Figure 1. Overlapping clusters: two clusters each subscribing to different topics



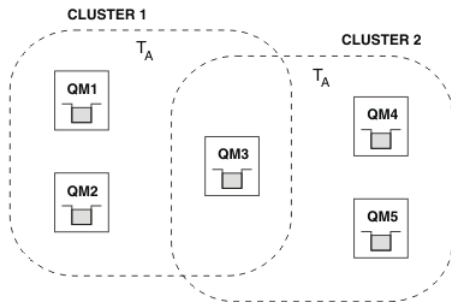
Overlapping clusters, overlapping topics

Another way to use overlapping clusters with publish/subscribe clustering is not to publish or subscribe using queue managers in the cluster overlap. The overlap might be used as a gateway for queued messaging, with the traditional clusters also being used for publish/subscribe. In this topology, if you cannot ensure that the overlapping clusters have non-overlapping topics, do not publish or subscribe using applications connected to queue managers in the overlap.

In [Figure 2](#) two topic objects T_A are defined, one in CLUSTER 1 and the other in CLUSTER 2. Messages are output to the log of queue manager QM3 informing users that the topic object T_A exists in two clusters. A publisher to QM3, in the cluster overlap, cannot control whether it is publishing to T_A in CLUSTER 1 or T_A in CLUSTER 2. The publications might flow to either cluster. Similarly, a subscriber to QM3, in the cluster overlap, cannot control whether it is subscribing to T_A in CLUSTER 1 or T_A in CLUSTER 2. It might receive publications from one or other of the clusters. Publishers and subscribers that are not in the overlap only publish and subscribe to topics in their cluster.

An application connected to queue manager QM3, subscribing on a topic that resolved to topic object T_A results in proxy subscriptions being sent to one cluster only – either to queue managers QM1 and QM2 or to queue managers QM4 and QM5. The cluster chosen depends on which cluster topic object was added last to the cluster cache in queue manager QM3.

Figure 2. Overlapping clusters: two clusters each subscribing to the same topic



Overlapping topics - example 1: Different topic spaces

If the two topic spaces in the two clusters are different, ensure that the topic spaces are not inadvertently overlapped by associating a topic object with the same name in different clusters to different topics.

For example, suppose you have a topic definition with the same name in each cluster. In [Figure 2](#), suppose the clusters are queue managers running systems to subscribe to published articles for different sporting journals. Topic object T_A might be "Latest scores".

- Topic: T_A in CLUSTER 1 with TopicString: /Wimbledon scores
- Topic: T_A in CLUSTER 2 with TopicString: /Masters scores

A subscription for T_A made on queue manager QM3 might resolve to either the golf or tennis scores depending which had been updated most recently. The latest tennis scores publication on queue manager QM3 might go to either the golf or tennis journal subscribers depending which topic definition had been updated most recently.

►If any queue manager receives multiple definitions on the same topic string which differ in any detail including cluster name, the behavior of publications and subscriptions on those topics or topic string is undefined. An informational message is issued to alert the administrator to the duplicate definition. ◀

Overlapping topics - example 2: Overlapping topic spaces

If the two topic spaces in the two clusters are subsets of the same overall topic space, ensure that you do not overlap the scope of subscribers.

Administrative topic objects are used to provide the root name of a topic, and can be used to define non-overlapping topic spaces. Publishers and subscribers do not need to know the absolute topic string path to a topic. They append their subtopic string to the root topic name to form a full topic name. You can use this mechanism with cluster topic definitions to divide a topic space into discrete topic subspaces, mapped to different publish/subscribe clusters.

For example, if the complete topic space is Sports, you might define subspaces for football and tennis. The football subspace is Sport/Football and is mapped to CLUSTER 1 using the topic definition Football. The tennis subspace is Sport/Tennis and is mapped to CLUSTER 2 using the cluster topic definition Tennis.

Subscribers in CLUSTER 1 receive only publications on football, and subscribers in CLUSTER 2 receive only publications on tennis.

Suppose, however that the subspaces overlapped. In this revised example, CLUSTER 1 subscribers might have access to all sports, using the cluster topic definition Sports that maps to Sport. Their topic space now overlaps with the topic space of tennis.

An unexpected result would be a tennis subscriber receiving a publication on another sport. This result could arise if there are any publishers connected to QM3, in the cluster overlap. A publisher in QM3 can publish to any sport, because it has access to both topic definitions, Sport and Tennis. A Sport/Football publication could flow into CLUSTER 2 from a publisher connected to QM3 intending it only to flow into CLUSTER 1. If subscribers in CLUSTER 2 subscribed to everything published, expecting to only receive topics published on tennis, they would also receive the football publication.

Parent topic: [Publish/subscribe clusters](#)

This build: January 26, 2011 10:56:28

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps15690_

3.2.1.4. Subscription scope and publication scope in publish/subscribe clusters

The scope of publications and subscriptions is defined in the cluster topic object.

►If a cluster topic object is defined with SUBSCOPE(QMGR), the definition is shared with the cluster, but the scope of subscriptions based on that topic is local only and publications are not received from the cluster. ◀

►If a cluster topic object is defined with PUBSCOPE(QMGR), the definition is shared with the cluster, but the scope of publications based on that topic is local only and they are not 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. Note that this does not prevent publication or subscription if topic objects are defined on subtopics. ◀

Parent topic: [Publish/subscribe clusters](#)

This build: January 26, 2011 10:56:30

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps15950_

3.2.1.5. REFRESH CLUSTER considerations

►The REFRESH CLUSTER command can cause temporary disruption to publish/subscribe traffic in a publish/subscribe cluster. We therefore recommend only to run REFRESH CLUSTER command when under the guidance of your IBM Support Center.◄

The disruption can occur as follows:

- 10 second, or longer, pauses in message delivery.
- MQOPEN and MQPUT failures, for example, MQRC_NO_DESTINATIONS_AVAILABLE.

Parent topic: [Publish/subscribe clusters](#)

This build: January 26, 2011 10:56:28

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps15700_

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

[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.

[Disconnect a queue manager from a broker hierarchy](#)

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

Parent topic: [Publish/subscribe topologies](#)

Related concepts

[Distributed publish/subscribe security](#)

Related tasks

[Connect a queue manager to a broker hierarchy](#)

[Disconnect a queue manager from a broker hierarchy](#)

This build: January 26, 2011 10:56:28

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps15800_

3.2.2.1. 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 a WebSphere MQ connection. There are two ways to establish the connection.
 - Connect the queue managers to a WebSphere MQ 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 example, suppose you are connecting to a queue manager called PARENT. Define a queue manager alias for PARENT that resolves to the transmission queue to parent. To place messages for PARENT on the transmission queue `PARENT.XMITQ`, use the following MQSC command to define the queue manager alias.

```
DEFINE QREMOTE (PARENT) RNAME(' ') RQMNAME(PARENT) XMITQ(PARENT.XMITQ)
```

About this task

In WebSphere® MQ Version 6.0, when the appropriate channels and queues are defined, brokers connect to one another as defined by parameters provided on the **strmqbrk** command.

The **strmqbrk** command works differently in WebSphere MQ Version 7.0 and you can no longer use it to connect children to parents. Instead you use the **ALTER QMGR PARENT (PARENT_NAME) runmqsc** command.

In WebSphere MQ Version 7, distributed publish/subscribe is typically implemented by using queue manager clusters and clustered topic definitions. For interoperability with WebSphere MQ Version 6 and WebSphere Message Broker V6.1 and WebSphere Event Broker V6.1 and earlier, you can also connect version 7 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, 2008. 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 type(All)
  14 : display pubsub type(All)
AMQ8723: Display pub/sub status details.
      QMNAME (QM2)                TYPE (LOCAL)
AMQ8723: Display pub/sub status details.
      QMNAME (QM1)                TYPE (PARENT)
```


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, 2008. ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM1.
display pubsub type(all)
  1 : display pubsub type(all)
AMQ8723: Display pub/sub status details.
      QMNAME (QM1)                TYPE (LOCAL)
AMQ8723: Display pub/sub status details.
      QMNAME (QM2)                TYPE (CHILD)
```

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.

Parent topic: [Publish/subscribe hierarchies](#)

 This build: January 26, 2011 10:56:28

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps15810_

3.2.2.2. 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

In WebSphere® MQ Version 6.0, queue managers were disconnected from one another using the **dltmqbrk** command, and required that all child queue managers were disconnected first. In WebSphere MQ Version 7, the **dltmqbrk** command is used to discard WebSphere MQ Version 6 broker resources after migration to version 7 using the **strmqbrk** command.

You disconnect a version 7 queue manager from a broker hierarchy using the **ALTER QMGR** command. Unlike version 6, you can disconnect version 7 queue managers in any order and 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, 2008. 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.

Parent topic: [Publish/subscribe hierarchies](#)

 This build: January 26, 2011 10:56:29

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps15820_

3.3. 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.

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.

[Subscription scope](#)

The scope of a subscription controls whether a queue manager forwards a subscription to remote queue managers. Use the **SUBSCOPE** topic attribute to administer the scope of subscriptions.

[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.

[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.

[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.

[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.

Parent topic: [Distributed publish/subscribe](#)

Related concepts

[Publication scope](#)

[Subscription scope](#)

 This build: January 26, 2011 10:56:29

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps15920_

3.3.1. 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.

Parent topic: [Controlling the flow of publications and subscriptions](#)

 This build: January 26, 2011 10:56:29

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps15930_

3.3.2. Subscription scope

The scope of a subscription controls whether a queue manager forwards a subscription to remote queue managers. Use the **SUBSCOPE** topic attribute to administer the scope of subscriptions.

If a subscription is not forwarded to remote queue managers, a subscriber can receive only local publications.

The **SUBSCOPE** topic attribute is used to determine the scope of subscriptions made to a specific topic. You can set the attribute to one of the following values:

QMGR

A proxy subscription is not propagated to remote queue managers, and therefore the subscriber receives only local publications.

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.

Subscribers can override the **SUBSCOPE** setting by specifying either the `MQSO_SCOPE_QMGR` or `MQSO_SCOPE_ALL` subscription option when creating a subscription.

Parent topic: [Controlling the flow of publications and subscriptions](#)

 This build: January 26, 2011 10:56:30

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps15940_

3.3.3. 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. For the difference in the interpretation of scope between the version 6 and version 7 publish/subscribe, see [Local and global publications and subscriptions](#).

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.

Parent topic: [Controlling the flow of publications and subscriptions](#)

 This build: January 26, 2011 10:56:30

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps15960_

3.3.4. 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](#) for a discussion of some 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 below.

- [Case 1. Publish/subscribe clusters.](#)
- [Case 2. Publish/subscribe hierarchies in version 7.](#)
- [Case 3. Publish/subscribe hierarchies and streams in version 6.](#)

In separate topics, the following tasks describe how to combine topic spaces.

- [Create a single topic space in a publish/subscribe cluster.](#)
- [Add a version 7 queue manager to existing version 6 topic spaces.](#)
- [Combine the topic spaces of multiple clusters.](#)
- [Combine and isolate topic spaces in multiple clusters](#)
- [Publish and subscribe to topic spaces in multiple clusters](#)

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 recreate 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](#) for a further discussion of 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 is made up 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.

Parent topic: [Controlling the flow of publications and subscriptions](#)

Related concepts

[How does distributed publish/subscribe work?](#)


[Publish/subscribe clusters](#)

[Publish/subscribe hierarchies](#)

[Multiple cluster topic definitions](#)

Related tasks

[Adding a stream](#)

 This build: January 26, 2011 10:56:31

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps15966_

3.3.5. 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.

[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.

[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.


[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.

[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.

Parent topic: [Controlling the flow of publications and subscriptions](#)

 This build: January 26, 2011 10:56:31

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps15964_

3.3.5.1. 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.


Parent topic: [Combining topics spaces](#)

Related concepts

[Publish/subscribe clusters](#)

Related information

[Setting up your first cluster](#)

 This build: January 26, 2011 10:56:30

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps15961_

3.3.5.2. 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.

Parent topic: [Combining topics spaces](#)

Related concepts


[Queued publish/subscribe compatibility](#)

[Controlling queued publish/subscribe](#)

[WebSphere MQ publish/subscribe topology migration](#)

Related tasks

[Connect a queue manager to a broker hierarchy](#)

 This build: January 26, 2011 10:56:30

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps15963_

3.3.5.3. 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

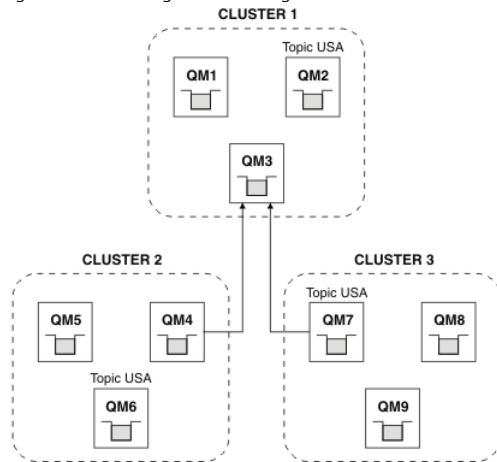
You have existing publish/subscribe clusters, and you want to propagate some cluster topics into all the clusters.

About this task

To propagate publications from one cluster to another, you need to join the clusters together in a hierarchy; see [Figure 1](#). 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 1. 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 1](#). 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.

Parent topic: [Combining topics spaces](#)

This build: January 26, 2011 10:56:31

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps15965_

3.3.5.4. 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.

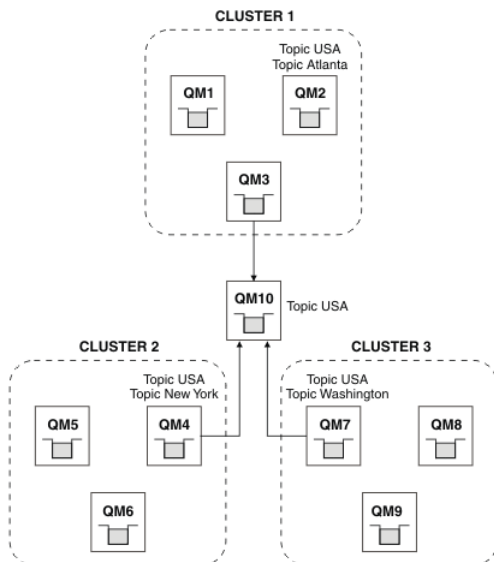
Before you begin

Examine the topic [Combine the topic spaces of multiple clusters](#). 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 1](#) in [Combine the topic spaces of multiple clusters](#) 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 1](#). Use the bridging queue manager to filter which publications and subscriptions can flow from one cluster to another.

Figure 1. Bridged clusters



Use the bridge to isolate cluster topics that you do not want exposed across the bridge on the other clusters. In [Figure 1](#), 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 1](#), 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.

Parent topic: [Combining topics spaces](#)

This build: January 26, 2011 10:56:31

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps15967_

3.3.5.5. 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

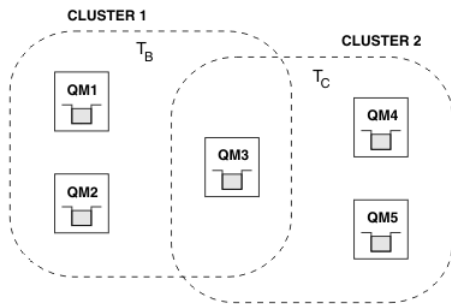
You might have chosen to overlap clusters for various different reasons.

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 1](#). 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 1. 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.

- DEFINE TOPIC(B) TOPICSTR('B') CLUSTER('CLUSTER 1') ...
- DEFINE TOPIC(C) TOPICSTR('C') CLUSTER('CLUSTER 2') ...

Example

In [Figure 1](#) 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.

Parent topic: [Combining topics spaces](#)

Related concepts

[Overlapping cluster support and publish/subscribe](#)

This build: January 26, 2011 10:56:31

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps15969_

3.3.6. 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.

Parent topic: [Controlling the flow of publications and subscriptions](#)

 This build: January 26, 2011 10:56:30

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps15962_

3.4. 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](#) 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.

Parent topic: [Distributed publish/subscribe](#)

Related concepts

[Loop detection fingerprint format](#)

 This build: January 26, 2011 10:56:39

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps16000_

3.4.1. 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.

Parent topic: [How loop detection works](#)

This build: January 26, 2011 10:56:46

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps21010_

3.5. 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.

Parent topic: [Distributed publish/subscribe](#)

This build: January 26, 2011 10:56:46

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps21020_

3.6. Distributed publish/subscribe security

Distributed publish/subscribe internal messages such as proxy subscriptions and publications are put to distributed publish/subscribe system queues (SYSTEM.INTER.QMGR.CONTROL, for example) by the receiving channel 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® and on distributed systems, no local topic object is required to establish access control, so administrators can choose to apply access control to clustered topic objects irrespective of whether they exist in the cluster yet.

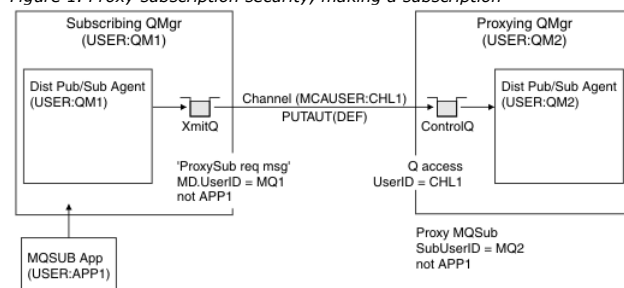
System administrators are responsible for access control on their local system and trust other members of the hierarchy or cluster collectives to which they are attached to be responsible for their own access control policy. It might not be necessary to impose any access control, or access control can be defined on high-level objects in the topic tree, or fine level access control can be defined for each subdivision of the topic namespace. Because access control is defined for each separate machine it is likely to be burdensome if fine level control is needed.

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 then allowed to do distributed publish/subscribe, an authority check is done when the channel puts the message to a distributed publish/subscribe queue; for example, SYSTEM.INTER.QMGR.CONTROL. 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 so on, depending on value and platform). For more information about channel security, see [Security](#).

Proxy subscriptions are made with the user ID of the distributed publish/subscribe agent on the remote queue manager (QM2 in [Figure 1](#)) which can then easily be granted access to local topic object profiles, because that user ID is defined in the system and there are therefore no domain conflicts.

Figure 1. Proxy subscription security, making a subscription



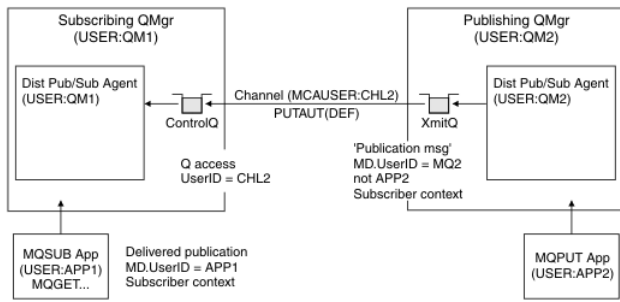
Sending back remote publications

When a publication is made on the publishing queue manager, a copy satisfies the proxy subscription that was made, and the context of that message contains the context of the user ID which made the subscription, QM2 in [Figure 2](#). The proxy subscription is made with a destination queue that is a remote queue, so the publication message is resolved onto a transmission queue.

Again, 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 so on, depending on value and platform). For more information about channel security, see [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 2. 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. Where there are multiple different remote queue managers with channels running under different MCAUSER IDs (for instance, when multiple hierarchical connections are configured on a single queue manager), then all those user IDs need to be granted access to the SYSTEM.INTER.QMGR.* queues.

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 by the user ID of the distributed publish/subscribe agent from the queue manager that is sending the internal message, or publication message (see Figure 2), 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.

►System queue security◄ provides a list of queues and the access that is required to securely set up your distributed publish/subscribe environment. If any internal messages or publications fail to be put due to security violations, the channel writes a message to the log in the normal manner and the messages can be sent to the dead-letter queue according to normal channel error processing.

All inter-queue manager messaging for the purposes of distributed publish/subscribe runs using normal channel security. No special casing is required in the security manager on behalf of the distributed publish/subscribe component.

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 MQRD_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
- *ADMCRD *ADMCLT *ADMCHG authority on the SYSTEM.BROKER.CONTROL.QUEUE queue.

◄ The default user IDs are as follows:

Windows	MUSR_MQADMIN
UNIX systems	mqm
i5/OS	QMQM
z/OS	The channel initiator address space user ID

►Create and grant access to the 'mqm' user ID if hierarchically attached to a queue manager on i5/OS for Queue Managers on Windows, UNIX, and z/OS platforms.◄

►Create and grant access to the 'mqm' user ID if hierarchically attached to a queue manager on Windows or UNIX for Queue Managers on i5/OS 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, and i5/OS platforms.◄

►User IDs can be case sensitive. The originating queue manager (if Windows, UNIX, or i5/OS) forces the user ID to be all uppercase. The receiving queue manager (if Windows or UNIX) forces the user ID to be all lowercase. Therefore, all user IDs created on UNIX need to be created in their lowercase form. However, if a message exit has been installed, the forcing to uppercase or lowercased does not take place and 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 and Windows, ensure the user IDs are specified in lowercase.
- On i5/OS and z/OS, ensure the user IDs are specified in uppercase.

Parent topic: [Distributed publish/subscribe](#)

This build: January 26, 2011 10:56:48

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps21070_

3.7. Distributed publish/subscribe system queues

Four system queues are used by queue managers when they do publish/subscribe messaging. You normally need to be aware of their existence only for problem determination or capacity planning purposes.

Table 1. Publish/subscribe system queues

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

► On z/OS, you set up the necessary system objects when you create the queue manager, by including the CSQINS4R and CSQINS4G samples in the CSQINP2 initialization input data set. For more information, see [Customize the initialization input data sets](#). ◀

The attributes of the distributed publish/subscribe system queues are as displayed in [Table 2](#).

Table 2. Attributes of publish/subscribe system queues

Attribute	Value
DEFPERSIST	Yes
DEFSOPT	This takes the value EXCL.
MAXMSGL	On AIX®, HP-UX, Linux, i5/OS®, Solaris and Windows this takes the value of MAXMSGL parameter of the ALTER QMGR command. On z/OS® this takes the value 100 MB (104 857 600 bytes).
MAXDEPTH	▶◀ This takes the value 999 999 999.
SHARE	This is a keyword that specifies that the queue can be shared for GET.
STGCLASS	On z/OS this takes the value 'SYSTEM'. On other platforms this attribute is not used.


[Publish/subscribe system queue errors](#)

Errors can occur when distributed publish/subscribe queue manager queues are unavailable.

Parent topic: [Distributed publish/subscribe](#)

Related reference

[Publish/subscribe system queue errors](#)

 This build: January 26, 2011 10:56:47

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps21030_

3.7.1. 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.

Parent topic: [Distributed publish/subscribe system queues](#)

 This build: January 26, 2011 10:56:47

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps21040_

4. Writing publish/subscribe applications

Start writing publish/subscribe WebSphere MQ applications. We assume you have already written point to point WebSphere MQ applications before.

► [Writing publisher applications](#) ◀

Get started with writing publisher applications by studying two examples. The first is modelled as closely as possible on a point to point application putting messages on a queue, and the second demonstrates creating topics dynamically - a more common pattern for publisher applications.

► [Writing subscriber applications](#) ◀

There are many more patterns of subscriber application than publisher. Three are illustrated: a WebSphere MQ application consuming messages from

a queue, an application that creates a subscription and requires no knowledge of queuing, and finally an example that uses both queuing and subscriptions.

►Publish/subscribe lifecycles◀

Consider the lifecycles of topics, subscriptions, subscribers, publications, publishers and queues in designing publish/subscribe applications.

►Publish/subscribe message properties◀

Several message properties relate to Websphere MQ publish/subscribe messaging.

►Message ordering◀

For a particular topic, messages are published by the queue manager in the same order as they are received from publishing applications (subject to reordering based on message priority).

►Intercepting publications◀

You can intercept a publication, modify it, and then republish it before it reaches any other subscriber.

►Publishing options◀

Several options are available that control the way messages are published.

►Subscription options◀

Parent topic: [Publish/Subscribe User's Guide](#)

This build: January 26, 2011 10:56:23

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps10438_

4.1. Writing publisher applications

Get started with writing publisher applications by studying two examples. The first is modelled as closely as possible on a point to point application putting messages on a queue, and the second demonstrates creating topics dynamically - a more common pattern for publisher applications.

Writing a simple WebSphere MQ publisher application is just like writing a WebSphere MQ point to point application that puts messages to a queue ([Table 1](#)). The difference is you MQPUT messages to a topic, not to a queue.

Table 1. Point to point vs. publish/subscribe WebSphere MQ program pattern.

Step	Point to point MQ Call	Publish MQ Call
Connect to a queue manager	MQCONN	MQCONN
Open queue	MQOPEN	
Open topic		MQOPEN
Put message(s)	MQPUT	MQPUT
Close topic		MQCLOSE
Close queue	MQCLOSE	
Disconnect from queue manager	MQDISC	MQDISC

To make that concrete, there are two examples of applications to publish stock prices. In the first example ([Example 1: Publisher to a fixed topic](#)), that is modelled very closely on putting messages to a queue, the administrator creates a topic definition in a similar way to creating a queue. The programmer codes MQPUT to write messages to the topic instead of writing them to a queue. In the second example ([Example 2: Publisher to a variable topic](#)), the pattern of interaction of the program with WebSphere MQ is similar. The difference is the programmer provides the topic to which the message is written, rather than the administrator. In practice this usually means the topic string is content defined, or provided "out of band", that is, provided by human input, or by another source of information.

►Example 1: Publisher to a fixed topic◀

A WebSphere MQ program to illustrate publishing to an administratively defined topic.

►Example 2: Publisher to a variable topic◀

A Websphere MQ program to illustrate publishing to a programmatically defined topic.

Parent topic: [Writing publish/subscribe applications](#)

Related concepts

[Writing subscriber applications](#)

This build: January 26, 2011 10:56:17

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps10431_

4.1.1. Example 1: Publisher to a fixed topic

A WebSphere MQ program to illustrate publishing to an administratively defined topic.

Note: The compact coding style is intended for readability not production use.

Figure 1. Simple WebSphere MQ publisher to a fixed topic.

See the output in [Figure 2](#)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    char    topicNameDefault[] = "IBMSTOCKPRICE";
    char    publicationDefault[] = "129";
```

```

MQCHAR48 qmName = "";

MQHCONN Hconn = MQHC_UNUSABLE_HCONN; /* connection handle */
MQHOBJ Hobj = MQHO_NONE; /* object handle sub queue */
MQLONG CompCode = MQCC_OK; /* completion code */
MQLONG Reason = MQRC_NONE; /* reason code */
MQOD td = {MQOD_DEFAULT}; /* Object descriptor */
MQMD md = {MQMD_DEFAULT}; /* Message Descriptor */
MQPMO pmo = {MQPMO_DEFAULT}; /* put message options */
MQCHAR resTopicStr[151]; /* Returned vale of topic string */
char * topicName = topicNameDefault;
char * publication = publicationDefault;
memset (resTopicStr, 0 , sizeof(resTopicStr));

switch(argc){ /* replace defaults with args if provided */
  default:
    publication = argv[2];
  case(2):
    topicName = argv[1];
  case(1):
    printf("Optional parameters: TopicObject Publication\n");
}
do {
  MQCONN(qmName, &Hconn, &CompCode, &Reason);
  if (CompCode != MQCC_OK) break;
  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.ResObjectString.VSPtr = resTopicStr;
  td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
  MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF QUIESCING, &Hobj, &CompCode, &Reason);
  if (CompCode != MQCC_OK) break;
  pmo.Options = MQPMO_FAIL_IF QUIESCING | MQPMO_RETAIN;
  MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
  if (CompCode != MQCC_OK) break;
  MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
  if (CompCode != MQCC_OK) break;
  MQDISC(&Hconn, &CompCode, &Reason);
} while (0);
if (CompCode == MQCC_OK)
  printf("Published \"%s\" using topic \"%s\" to topic string \"%s\"\\n",
    publication, td.ObjectName, resTopicStr);
printf("Completion code %d and Return code %d\\n", CompCode, Reason);
}

```

Figure 2. Sample output from first publisher example

```

X:\Publish1\Debug>PublishStock
Optional parameters: TopicObject Publication
Published "129" using topic "IBMSTOCKPRICE" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

X:\Publish1\Debug>PublishStock IBMSTOCKPRICE 155
Optional parameters: TopicObject Publication
Published "155" using topic "IBMSTOCKPRICE" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

```

The lines of code selected below illustrate aspects of writing a publisher application for WebSphere MQ.

```
char topicNameDefault[] = "IBMSTOCKPRICE";
```

A default topic name is defined in the program. You can override it by providing the name of a different topic object as the first argument to the program.

```
MQCHAR resTopicStr[151];
```

`resTopicStr` is pointed at by `td.ResObjectString.VSPtr` and is used by `MQOPEN` to return the resolved topic string. Make the length of `resTopicStr` one larger than the length passed in `td.ResObjectString.VSBufSize` to give space for null termination.

```
memset (resTopicStr, 0, sizeof(resTopicStr));
```

Initialize `resTopicStr` to nulls to ensure the resolved topic string returned in an `MQCHARV` is null terminated.

```
td.ObjectType = MQOT_TOPIC
```

There is a new type of object for publish/subscribe: the *topic object*.

```
td.Version = MQOD_VERSION_4;
```

To use the new type of object, you must use at least *version 4* of the object descriptor.

```
strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
```

The `topicName` is the name of a topic object, sometimes called an administrative topic object. In the example the topic object needs to be created beforehand, using WebSphere MQ Explorer or this MQSC command,

```
DEFINE TOPIC (IBMSTOCKPRICE) TOPICSTR(NYSE/IBM/PRICE) REPLACE;
```

```
td.ResObjectString.VSPtr = resTopicStr;
```

The resolved topic string is echoed in the final `printf` in the program. Set up the `MQCHARV ResObjectString` structure for WebSphere MQ to return the resolved string back to the program.

```
MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF QUIESCING, &Hobj, &CompCode, &Reason);
```

Open the topic for output; just like opening a queue for output.

```
pmo.Options = MQPMO_FAIL_IF QUIESCING | MQPMO_RETAIN;
```

You want new subscribers to be able receive the publication, and by specifying `MQPMO_RETAIN` in the publisher, when we start a subscriber it receives the latest publication, published before the subscriber started, as its first matching publication. The alternative is to provide subscribers with publications published only after the subscriber started. In addition a subscriber has the option to decline to receive a retained publication by specifying `MQSO_NEW_PUBLICATIONS_ONLY` in its subscription.

```
MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
```

Add 1 to the length of the string passed to `MQPUT` to pass the null termination character to WebSphere MQ as part of the message buffer.

►What does the first example demonstrate? The example imitates as closely as possible the tried and tested traditional pattern for writing point to point WebSphere MQ programs. An important feature of the WebSphere MQ programming pattern is that the programmer is not concerned where messages are sent. The programmer's task is to connect to a queue manager, and pass to it the messages that are to be distributed to recipients. In the point-to-point paradigm, the programmer opens a queue (probably an alias queue) that the administrator has configured. The alias routes messages to a target queue, either on the local queue manager, or to a remote queue manager. While the messages are waiting to be delivered, they are stored on queues somewhere between the source and the destination. ◀

In the publish/subscribe pattern, instead of opening a queue, the programmer opens a topic. In our example, the topic is associated with a topic string by an administrator. The queue manager forwards the publication, using queues, to local or remote subscribers that have subscriptions that match the publication's topic string. In the case of retained publications the queue manager keeps the latest copy of the publication, even if it has no subscribers at present. The retained publication is available to forward to future subscribers. The publisher application plays no part in selecting or routing the publication to a destination; its task is to create and put publications to the topics defined by the administrator.

This fixed topic example is atypical of many publish/subscribe applications: it is static. It requires an administrator to define the topic strings and change the topics that are published on. Commonly publish/subscribe applications need to have knowledge of some or all of the topic tree. Perhaps topics change frequently, or perhaps although the topics do not change much, the number of topic combinations is very large and it is too onerous for an administrator to define a topic node for every topic string that might need to be published on. Perhaps topic strings are not known in advance of publication; a publisher application might use information from the publication content to specify a topic string, or it might have out of band information about topic strings to publish on, such as input from a browser. To cater for more dynamic styles of publishing, the next example shows how to create topics dynamically, as part of the publisher application.


Topics couple publishers and subscribers together. Designing the rules, or architecture, for naming topics, and organizing them in topic trees is a very important step in developing a publish/subscribe solution. Look carefully at the extent to which organization of the topic tree binds of publisher and subscriber programs together, and binds them to the content of the topic tree. Ask yourself the question whether changes in the topic tree will impact publisher and subscriber applications, and how you can minimize the impact. Built into the architecture of the WebSphere MQ publish/subscribe model is the notion of an administrative topic object that provides the root part, or root subtree, of a topic. The topic object gives you the option of defining the root part of the topic tree administratively that simplifies application programming and operations, and consequently improves maintainability. For example, if you are deploying multiple publish/subscribe applications that have isolated topic trees, then by administratively defining the root part of the topic tree, you can guarantee the isolation of topic trees, even if there is no consistency in the topic naming conventions adopted by the different applications.

In practice, publisher applications cover a spectrum from solely using fixed topics, as in this example, and variable topics, as in the next. [Example 2: Publisher to a variable topic](#) also demonstrates combining the use of topics and topic strings.

Parent topic: [Writing publisher applications](#)

Related concepts

[Example 2: Publisher to a variable topic](#)
[Writing subscriber applications](#)

 This build: January 26, 2011 10:56:21

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
 ps10433_

4.1.2. Example 2: Publisher to a variable topic

A Websphere MQ program to illustrate publishing to a programmatically defined topic.

Note: The compact coding style is intended for readability not production use.

Figure 1. Simple WebSphere MQ publisher to a variable topic.

See the output in [Figure 2](#).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    char    topicNameDefault[] = "STOCKS";
    char    topicStringDefault[] = "IBM/PRICE";
    char    publicationDefault[] = "130";
    MQCHAR48 qmName = "";

    MQHCONN    Hconn = MQHC_UNUSABLE_HCONN; /* connection handle */
    MQHOBJ    Hobj = MQHO_NONE; /* object handle sub queue */
    MQLONG    CompCode = MQCC_OK; /* completion code */
    MQLONG    Reason = MQRC_NONE; /* reason code */
    MQOD    td = {MQOD_DEFAULT}; /* Object descriptor */
    MQMD    md = {MQMD_DEFAULT}; /* Message Descriptor */
    MQPMO    pmo = {MQPMO_DEFAULT}; /* put message options */
    MQCHAR    resTopicStr[151]; /* Returned value of topic string */
    char *    topicName = topicNameDefault;
    char *    topicString = topicStringDefault;
    char *    publication = publicationDefault;
    memset    (resTopicStr, 0 , sizeof(resTopicStr));

    switch(argc){
        /* Replace defaults with args if provided */
        default:
            publication = argv[3];
        case(3):
            topicString = argv[2];
        case(2):
            if (strcmp(argv[1],"/")) /* "/" invalid = No topic object */
                topicName = argv[1];
            else
                *topicName = '\0';
        case(1):
            printf("Provide parameters: TopicObject TopicString Publication\n");
    }

    printf("Publish \"%s\" to topic \"%-48s\" and topic string \"%s\"\n", publication, topicName, topicString);
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        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);
```



```

    if (CompCode != MQCC_OK) break;
    pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
    MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
    if (CompCode != MQCC_OK) break;
    MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
    if (CompCode != MQCC_OK) break;
    MQDISC(&Hconn, &CompCode, &Reason);
} while (0);
if (CompCode == MQCC_OK)
    printf("Published \"%s\" to topic string \"%s\"\\n", publication, resTopicStr);
printf("Completion code %d and Return code %d\\n", CompCode, Reason);
}

```

Figure 2. Sample output from second publisher example

```

X:\Publish2\Debug>PublishStock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

X:\Publish2\Debug>PublishStock / NYSE/IBM/PRICE 131
Provide parameters: TopicObject TopicString Publication
Publish "131" to topic "" and topic string "NYSE/IBM/PRICE"
Published "131" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

```

There are a few points to note about this example.

```
char topicNameDefault[] = "STOCKS";
```

The default topic name `STOCKS` defines part of the topic string. You can override this topic name by providing it as the first argument to the program, or eliminate the use of the topic name by supplying `/` as the first parameter.

```
char topicString[101] = "IBM/PRICE";
```

`IBM/PRICE` is the default topic string. You can override this topic string by providing it as the second argument to the program.

The queue manager combines the topic string provided by the `STOCKS` topic object, `"NYSE"`, with the topic string provided by the program `"IBM/PRICE"` and inserts a `"/"` between the two yielding topic strings as the resolved topic string `"NYSE/IBM/PRICE"`. The resulting topic string is the same as the one defined in the `IBMSTOCKPRICE` topic object, and has precisely the same effect.

The administrative topic object associated with the resolved topic string is not necessarily the same topic object as passed to `MQOPEN` by the publisher. WebSphere MQ uses the tree implicit in the resolved topic string to work out which administrative topic object defines the attributes associated with the publication.

Suppose there are two topic objects `A` and `B`, and `A` defines topic `"a"`, and `B` defines topic `"a/b"` (Figure 3). If the publisher program refers to topic object `A` and provides topic string `"b"`, resolving the topic to the topic string `"a/b"`, then the publication inherits its properties from topic object `B` because the topic matches the topic string `"a/b"` defined for `B`.

```
if (strcmp(argv[1], "/"))
```

`argv[1]` is the optionally provided `topicName`. `"/"` is invalid as a topic name; here it signifies that there is no topic name, and the topic string is provided entirely by the program. The output in Figure 2 shows the whole topic string being supplied dynamically by the program.

```
strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
```

For the default case, the optional `topicName` needs to be created beforehand, using WebSphere MQ Explorer or this MQSC command:

```
DEFINE TOPIC(STOCKS) TOPICSTR(NYSE) REPLACE;
```

```
td.ObjectString.VSPtr = topicString;
```

The topic string is a `MQCHARV` field in the topic descriptor

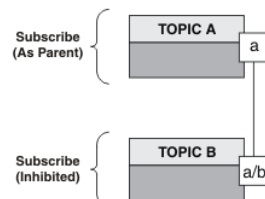


Figure 3. Topic object associations

What does the second example demonstrate? Although the code is very similar to the first example - effectively there are only two lines difference - the result is a significantly different program to the first. The programmer controls the destinations to which publications are sent. In conjunction with minimal administrator input used to design subscriber applications, no topics or queues need to be predefined to route publications from publishers to subscribers.

In the point-to-point messaging paradigm, queues have to be defined before messages are able to flow. For publish/subscribe, they do not, although WebSphere MQ implements publish/subscribe using its underlying queuing system; the benefits of guaranteed delivery, transactionality and loose coupling associated with messaging and queueing are inherited by publish/subscribe applications.

A designer has to decide whether publisher, and subscriber, programs are to be aware of the underlying topic tree or not, and also whether subscriber programs are aware of queueing or not. Study the subscriber example applications next. They are designed to be used with the publisher examples, typically publishing and subscribing to `NYSE/IBM/PRICE`.

Parent topic: [Writing publisher applications](#)

Related concepts

[Example 1: Publisher to a fixed topic](#)

[Writing subscriber applications](#)

This build: January 26, 2011 10:56:21

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps10434_

4.2. Writing subscriber applications

There are many more patterns of subscriber application than publisher. Three are illustrated: a WebSphere MQ application consuming messages from a queue, an application that creates a subscription and requires no knowledge of queueing, and finally an example that uses both queueing and subscriptions.

In [Table 1](#) the three styles of consumer or subscriber are listed, together with the sequences of WebSphere MQ function calls that characterize them.

1. The first style, MQ Publication Consumer, is identical to a point to point MQ program that only does MQGET. The application has no knowledge that it is consuming publications - it is simply reading messages from a queue. The subscription that causes publications to get routed to the queue is created administratively using WebSphere MQ Explorer or a command.
2. The second style is the preferred pattern for most subscriber applications. The subscriber application creates the subscription, and then gets publications. The queue management is all performed by the queue manager.
3. In the third style, the subscriber application elects to open and close the underlying queue that is used for publications as well as issue subscriptions to fill the queue with publications.

One way to understand these styles is to study the example c programs listed in [Table 1](#) for each of the styles. The examples are designed to be run in conjunction with the publisher example found in [Writing publisher applications](#).

Table 1. Point to point vs. subscribe WebSphere MQ program patterns.

Step	MQ message consumer	Example 1: MQ Publication consumer	Example 2: Managed MQ subscriber	Example 3: Unmanaged MQ subscriber
Connect to a queue manager	MQCONN	MQCONN	MQCONN	MQCONN
Open queue	MQOPEN	MQOPEN		MQOPEN
Subscribe			MQSUB	MQSUB
Put message(s)	MQGET	MQGET	MQGET	MQGET
Close queue	MQCLOSE	MQCLOSE	(MQCLOSE)	MQCLOSE
Close subscription			MQCLOSE	MQCLOSE
Disconnect from queue manager	MQDISC	MQDISC	MQDISC	MQDISC

Using MQCLOSE is always optional, either to release resources, pass MQCLOSE options, or just for symmetry with MQOPEN. Since you are unlikely to need to specify the MQCLOSE options when the subscription queue is closed in the Managed MQ subscriber case, and the symmetry argument is not relevant, the subscription queue is not explicitly closed in the Managed MQ subscriber example below.

Another way to understand publish/subscribe application patterns is to look at the interactions between the different entities involved. Lifeline, or UML sequence diagrams are a good way to study interactions. Three lifeline examples are described in [Publish/subscribe lifecycles](#).

►[Example 1: MQ Publication consumer](#)◄

The MQ Publication consumer is a WebSphere MQ message consumer that does not subscribe to topics itself.

►[Example 2: Managed MQ subscriber](#)◄

The managed MQ subscriber is the preferred pattern for most subscriber applications. The example requires *no* administrative definition of queues, topics or subscriptions.

►[Example 3: Unmanaged MQ subscriber](#)◄


The unmanaged subscriber is an important class of subscriber application. With it, you combine the benefits of publish/subscribe with *control* of queuing and consumption of publications. The example demonstrates different ways of combining subscriptions and queues.

Parent topic: ►[Writing publish/subscribe applications](#)◄

Related concepts

[Writing publisher applications](#)

[Publish/subscribe lifecycles](#)

 This build: January 26, 2011 10:56:22

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps10435_

4.2.1. Example 1: MQ Publication consumer

The MQ Publication consumer is a WebSphere MQ message consumer that does not subscribe to topics itself.

To create the subscription and publication queue for this example run the following commands, or define the objects using WebSphere MQ Explorer.

```
DEFINE QLOCAL(STOCKTICKER) REPLACE;
DEFINE SUB(IBMSTOCKPRICESUB) DEST(STOCKTICKER) TOPICOBJ(IBMSTOCKPRICE) REPLACE;
```

The IBMSTOCKPRICESUB subscription references the IBMSTOCK topic object created for the publisher example and the local queue STOCKTICKER. The topic object IBMSTOCK defines the topic string that is used in the subscription, NYSE/IBM/PRICE. Note that the topic object and the queue used to receive publications need to be defined before the subscription is created.

There are a number of valuable facets to the MQ publication consumer pattern:

1. Multiprocessing: sharing out of the work of reading publications. The publications all go onto the single queue associated with the subscription topic. Multiple consumers can open the queue using MQOO_INPUT_SHARED.
2. Centrally managed subscriptions. Applications do not construct their own subscription topics or subscriptions; the administrator is responsible for where publications are sent.
3. Subscription concentration: multiple different subscriptions can be sent to a single queue.
4. Subscription durability: the queue receives all publications whether or not consumers are active.
5. Migration and coexistence: the consumer code works equally well for a point-to-point and a publish/subscribe scenario.

The subscription creates a relationship between the topic string NYSE/IBM/PRICE and the queue STOCKTICKER. Publications, including any currently retained publication, are forwarded to STOCKTICKER from the moment the subscription is created.

An administratively created subscription can be managed or unmanaged. A managed subscription takes effect as soon as it has been created, just like an unmanaged subscription. Not all the pattern facets are available to a managed subscription. See [Example 3: Unmanaged MQ subscriber](#)

Note: The compact coding style is intended for readability not production use.

Figure 1. MQ publication consumer.

The results are shown in Figure 2.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    MQCHAR      publicationBuffer[101];
    MQCHAR48    subscriptionQueueDefault = "STOCKTICKER";
    MQCHAR48    qmName = "";
                /* Use default queue manager */

    MQHCONN    Hconn = MQHC_UNUSABLE_HCONN;
                /* connection handle */
    MQHOBJ     Hobj = MQHO_NONE;
                /* object handle sub queue */
    MQLONG     CompCode = MQCC_OK;
                /* completion code */
    MQLONG     Reason = MQRC_NONE;
                /* reason code */
    MQLONG     messlen = 0;

    MQOD       od = {MQOD_DEFAULT};
                /* Unmanaged subscription queue */
    MQMD       md = {MQMD_DEFAULT};
                /* Message Descriptor */
    MQGMO      gmo = {MQGMO_DEFAULT};
                /* Get message options */
    char *     publication=publicationBuffer;
    char *     subscriptionQueue = subscriptionQueueDefault;

    switch(argc){
        /* Replace defaults with args if provided */
        default:
            subscriptionQueue = argv[1]
        case(1):
            printf("Optional parameter: subscriptionQueue\n");
    }

    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        strncpy(od.ObjectName, subscriptionQueue, MQ_Q_NAME_LENGTH);
        MQOPEN(Hconn, &od, MQOO_INPUT_AS_Q_DEF | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
        gmo.WaitInterval = 10000;
        printf("Waiting %d seconds for publications from %s\n", gmo.WaitInterval/1000, subscriptionQueue);
        do {
            memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
            memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
            md.Encoding = MQENC_NATIVE;
            md.CodedCharSetId = MQCCSI_Q_MGR;
            memset(publication, 0, sizeof(publicationBuffer));
            MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1, publication, &messlen,
                &CompCode, &Reason);
            if (Reason == MQRC_NONE)
                printf("Received publication \"%s\"\n", publication);
        }
        while (CompCode == MQCC_OK);
        if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
}
```

Figure 2. Output from MQ publication consumer

```
X:\Subscriber\Debug>Subscriber1
Optional parameter: subscriptionQueue
Waiting 10 seconds for publications from STOCKTICKER
Received publication "129"
Completion code 0 and Return code 0
```

There are a couple of standard WebSphere MQ C language programming tips to be aware of:

memset(publication, 0, sizeof(publicationBuffer));

Ensure the message has a trailing null for easy formatting using `printf`. The publisher example includes the trailing null in the message buffer passed to `MQPUT` by adding 1 to `strlen(publication)`. Setting `MQCHAR` buffers to null is good programming style for WebSphere MQ C programs that use the buffers to store strings, ensuring a null follows an array of characters that does not completely fill the buffer.

MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1, publication, &messlen, &CompCode, &Reason);

Reserve one null at the end of the message buffer to ensure the returned message has trailing null in case `"if (messlen == strlen(publication));"` is true. This tip complements the preceding one, and ensures that there is at least one null in `publicationBuffer` that is not overwritten by the contents of `publication`.


Parent topic: [Writing subscriber applications](#)

Related concepts

[Example 2: Managed MQ subscriber](#)

[Example 3: Unmanaged MQ subscriber](#)

[Writing publisher applications](#)

 This build: January 26, 2011 10:56:22

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps10436_

4.2.2. Example 2: Managed MQ subscriber

The managed MQ subscriber is the preferred pattern for most subscriber applications. The example requires *no* administrative definition of queues, topics or subscriptions.

This simplest kind of managed subscriber typically makes use of a *non-durable* subscription. The example focuses on a non-durable subscription. The subscription only lasts only as long as the lifetime of the subscription handle from MQSUB. Any publications that match the topic string during the lifetime of the subscription are sent to the subscription queue (and possibly a retained publication if the flag MQSO_NEW_PUBLICATIONS_ONLY is not set or defaulted, an earlier publication matching the topic string was retained, and the publication was persistent or the queue manager has not terminated, since the publication was created).

You can also use a *durable* subscription with this pattern. Typically if a managed durable subscription is used it is done for reliability reasons, rather than to establish a subscription that, without any errors occurring, would outlive the subscriber. See the discussion of different lifecycles associated with managed, unmanaged, durable and non-durable subscriptions in the related topics section.

Durable subscriptions are often associated with persistent publications, and non-durable subscriptions with non-persistent publications, but there is no necessary relationship between subscription durability and publication persistence. All four combinations of persistence and durability are possible.

For the managed non-durable case we are considering, the queue manager creates a subscription queue that is purged and deleted when the queue is closed. The publications are removed from the queue when the non-durable subscription is closed.

The valuable facets of the managed non-durable pattern exemplified by this code are listed below.

1. On demand subscription: the subscription topic string is dynamic. It is provided by the application when it runs.
2. Self managing queue: the subscription queue is self defining and managing.
3. Self managing subscription lifecycle: *non-durable* subscriptions only exist for the duration of the subscriber application.
 - o If you define a *durable* managed subscription, then it results in a permanent subscription queue and publications continue to be stored on it with no subscriber programs being active. The queue manager deletes the queue (and clears any unretrieved publications from it) only after the application or administrator has chosen to delete the subscription. The subscription can be deleted using an administrative command, or by closing the subscription with the MQCO_REMOVE_SUB option.
 - o Consider setting SubExpiry for durable subscriptions so that publications cease to be sent to the queue and the subscriber can consume any remaining publications before removing the subscription and causing the queue manager to delete the queue and any remaining publications on it.
4. Flexible topic string deployment: Subscription topic management is simplified by defining the root part of the subscription using an administratively defined topic. The root part of the topic tree is then hidden from the application. By hiding the root part an application can be deployed without the application inadvertently creating a topic tree that overlaps with another topic tree created by another instance, or another application.
5. Administered topics: by using a topic string in which the first part matches an administratively defined topic object, publications are managed according to the attributes of the topic object.
 - o For example, if the first part of the topic string matches the topic string associated with a clustered topic object, then the subscription can receive publications from other members of the cluster
 - o The selective matching of administratively defined topic objects and programmatically defined subscriptions enables you to combine the benefits of both. The administrator provides attributes for topics, and the programmer dynamically defines "sub-topics" without being concerned about the management of topics.
 - o It is the resultant topic string which is used to match the topic object that provides the attributes associated with the topic, and not necessarily the topic object named in sd.Objectname, though they usually turn out to be one and the same. See the discussion in [Example 2: Publisher to a variable topic](#).

By making the subscription durable in the example, publications continue to be sent to the subscription queue after the subscriber has closed the subscription with the MQCO_KEEP_SUB option. The queue continues to receive publications when the subscriber is not active. You can override this behavior by creating the subscription with the MQSO_PUBLICATIONS_ON_REQUEST option and using MQSUBRQ to request the retained publication.

The subscription can be resumed later by opening the subscription with the MQCO_RESUME option.

You can use the queue handle, Hobj, returned by MQSUB in a number of ways. The queue handle is used in the example to inquire on the name of the subscription queue. Managed queues are opened using the default model queues SYSTEM.NDURABLE.MODEL.QUEUE or SYSTEM.DURABLE.MODEL.QUEUE. You can override the defaults by providing your own durable and non-durable model queues on a topic by topic basis as properties of the topic object associated with the subscription.

Regardless of the attributes inherited from the model queues, you cannot reuse a managed queue handle to create an additional subscription. Nor can you obtain another handle for the managed queue by opening the managed queue a second time using the returned queue name. The queue behaves as if it has been opened for exclusive input.

Unmanaged queues are more flexible than managed queues. You can, for example share unmanaged queues, or define multiple subscriptions on the one queue. The next example, [Example 3: Unmanaged MQ subscriber](#), demonstrates how to combine subscriptions with an unmanaged subscription queue.

Note: The compact coding style is intended for readability not production use.

Figure 1. Managed MQ subscriber - part 1: declarations and parameter handling.

The results are shown in [Figure 3](#).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>

void inquireQname(MQHCONN HConn, MQHOBJ Hobj, MQCHAR48 qName);

int main(int argc, char **argv)
{
    MQCHAR48 topicNameDefault = "STOCKS";
    char topicStringDefault[] = "IBM/PRICE";
    MQCHAR48 qmName = ""; /* Use default queue manager */
    MQCHAR48 qName = ""; /* Allocate to query queue name */
    char publicationBuffer[101]; /* Allocate to receive messages */
    char resTopicStrBuffer[151]; /* Allocate to resolve topic string */

    MQHCONN Hconn = MQHC_UNUSABLE_HCONN; /* connection handle */
    MQHOBJ Hobj = MQHO_NONE; /* publication queue handle */
    MQHOBJ Hsub = MQSO_NONE; /* subscription handle */
    MQLONG CompCode = MQCC_OK; /* completion code */
    MQLONG Reason = MQRC_NONE; /* reason code */
    MQLONG messlen = 0;
    MQSD sd = {MQSD_DEFAULT}; /* Subscription Descriptor */
    MQMD md = {MQMD_DEFAULT}; /* Message Descriptor */
    MQGMO gmo = {MQGMO_DEFAULT}; /* get message options */
}
```

```

char *   topicName   = topicNameDefault;
char *   topicString = topicStringDefault;
char *   publication = publicationBuffer;
char *   resTopicStr = resTopicStrBuffer;
memset(resTopicStr, 0, sizeof(resTopicStrBuffer));

switch(argc){
    /* Replace defaults with args if provided */
    default:
        topicString = argv[2];
    case(2):
        if (strcmp(argv[1],"/")
            /* "/" invalid = No topic object */
            topicName = argv[1];
        else
            *topicName = '\0';
    case(1):
        printf("Optional parameters: topicName, topicString\nValues \"%s\" \"%s\"",
            topicName, topicString);
}

```

There are some additional comments to make about the declarations in this example.

MQHOBJ Hobj = MQHO_NONE;

You cannot explicitly open a non-durable managed subscription queue to receive publications, but you do need to allocate storage for the object handle the queue manager returns when it opens the queue for you. It is important to initialize the handle to MQHO_OBJECT. This indicates to the queue manager that it needs to return a queue handle to the subscription queue.

MQSD sd = {MQSD_DEFAULT};

The new subscription descriptor, used in MQSUB.

MQCHAR48 qName;

Although the example doesn't require knowledge of the subscription queue, we do inquire the name of the subscription queue - the MQINQ binding is a little awkward in the C language, so you may find this part of the example useful to study.

Figure 2. Managed MQ subscriber - part 2: code body.

```

do {
    MQCONN(qmName, &Hconn, &CompCode, &Reason);
    if (CompCode != MQCC_OK) break;
    strncpy(sd.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
    sd.ObjectString.VSPtr = topicString;
    sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
    sd.Options = MQSO_CREATE | MQSO_MANAGED | MQSO_NON_DURABLE | MQSO_FAIL_IF QUIESCING ;
    sd.ResObjectString.VSPtr = resTopicStr;
    sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
    MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
    if (CompCode != MQCC_OK) break;
    gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
    gmo.WaitInterval = 10000;
    inquireQname(Hconn, Hobj, qName);
    printf("Waiting %d seconds for publications matching \"%s\" from \"%s-0.48s\"",
        gmo.WaitInterval/1000, resTopicStr, qName);
    do {
        memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
        memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
        md.Encoding = MQENC_NATIVE;
        md.CodedCharSetId = MQCCSI_Q_MGR;
        memset(publicationBuffer, 0, sizeof(publicationBuffer));
        MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1,
            publication, &messlen, &CompCode, &Reason);
        if (Reason == MQRC_NONE)
            printf("Received publication \"%s\"", publication);
    }
    while (CompCode == MQCC_OK);
    if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
    MQCLOSE(Hconn, &Hsub, MQCO_REMOVE_SUB, &CompCode, &Reason);
    if (CompCode != MQCC_OK) break;
    MQDISC(&Hconn, &CompCode, &Reason);
} while (0);
printf("Completion code %d and Return code %d\n", CompCode, Reason);
return;
}

void inquireQname(MQHCONN Hconn, MQHOBJ Hobj, MQCHAR48 qName) {
#define _selectors 1
#define _intAttrs 1

    MQLONG select[_selectors] = {MQCA_Q_NAME}; /* Array of attribute selectors */
    MQLONG intAttrs[_intAttrs]; /* Array of integer attributes */
    MQLONG CompCode, Reason;
    MQINQ(Hconn, Hobj, _selectors, select, _intAttrs, intAttrs, MQ_Q_NAME_LENGTH, qName,
        &CompCode, &Reason);
    if (CompCode != MQCC_OK) {
        printf("MQINQ failed with Condition code %d and Reason %d\n", CompCode, Reason);
        strcpy(qName, "unknown queue");
    }
    return;
}

```

Figure 3. Output from managed MQ subscriber

```

W:\Subscribe2\Debug>solution2
Optional parameters: topicName, topicString
Values "STOCKS" "IBM/PRICE"
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from "SYSTEM.MANAGED.NDURABLE.48A0AC7403300020"
Received publication "150"
Completion code 0 and Return code 0

W:\Subscribe2\Debug>solution2 / NYSE/IBM/PRICE
Optional parameters: topicName, topicString
Values "" "NYSE/IBM/PRICE"
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from "SYSTEM.MANAGED.NDURABLE.48A0AC7403310020"
Received publication "150"
Completion code 0 and Return code 0

```

There are some additional comments to make about the code in this example.

```
strncpy(sd.ObjectName, topicName, MQ_Q_NAME_LENGTH);
```

If `topicName` is null or blank (*default value*), the topic name is not used to compute the resolved topic string.

```
sd.ObjectString.VSPtr = topicString;
```

Rather than solely use a predefined topic object, in this example the programmer provides a topic object and a topic string, that are combined by `MQSUB`. Notice the topic string is a `MQCHARV` structure.

```
sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
```

An alternative to setting the length of a `MQCHARV` field.

```
sd.Options = MQSO_CREATE | MQSO_MANAGED | MQSO_NON_DURABLE | MQSO_FAIL_IF_QUIESCING;
```

After defining the topic string, the `sd.Options` flags need the most careful attention. There are many options, we shall specify only the most commonly used ones in this example, the others are left to default.

1. As the subscription is *non-durable*, in other words, it has a lifetime of the open subscription in the application, set the `MQSO_CREATE` flag. You can also set the (*default*) `MQSO_NON_DURABLE` flag for readability.
2. Complementing `MQSO_CREATE` is `MQSO_RESUME`. Both flags may be set together; the queue manager either creates a new subscription or resumes an existing subscription, whichever is appropriate. However, if you do specify `MQSO_RESUME` you must also initialize the `MQCHARV` structure for `sd.SubName`, even if there is no subscription to resume. Failure to initialize `SubName` results in a return code of 2440: `MQRC_SUB_NAME_ERROR` from `MQSUB`.
Note: `MQSO_RESUME` is always ignored for a non-durable managed subscription: but specifying it without initializing the `MQCHARV` structure for `sd.SubName` does cause the error.
3. In addition there is a third flag affecting how the subscription is opened, `MQSO_ALTER`. Given the right permissions, the properties of a resumed subscription are changed to match other attributes specified in `MQSUB`.
Note: At least one of the `MQSO_CREATE`, `MQSO_RESUME` and `MQSO_ALTER` flags must be specified. See the discussion in [MQSD Options](#) [Application programming reference](#). There are examples of using all three flags in [Example 3: Unmanaged MQ subscriber](#).
4. Set `MQSO_MANAGED` for the queue manager to manage the subscription for you automatically.

```
sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
```

Optionally, omit setting the length of `MQCHARV` for null terminated strings and use the null terminator flag instead.

```
sd.ResObjectString.VSPtr = resTopicStr;
```

The resulting topic string is echoed in first `printf` in the program. Set up `MQCHARV ResObjectString` for WebSphere MQ to return the resolved string back to our program.

Note: We initialized `resTopicStringBuffer` to nulls in `memset(resTopicStr, 0, sizeof(resTopicStrBuffer))`. Returned topic strings do not end with a trailing null.

```
sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
```

Set the buffer size of the `sd.ResObjectString` to one less than its actual size. This prevents overwriting the null terminator we provided, in case the resolved topic string fills the entire buffer.

Note: No error is returned if the topic string is longer than `sizeof(resTopicStrBuffer)-1`. Even if `VSLength > VSBufSize` the length returned in `sd.ResObjectString.VSLength` is the length of the complete string and not necessarily the length of the returned string. Test `sd.ResObjectString.VSLength < sd.ResObjectString.VSBufSize` to confirm the topic string is complete.

```
MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
```

The `MQSUB` function creates a subscription. If it is non-durable you are probably not interested in its name, though you can inspect its status in WebSphere MQ Explorer. You can provide the `sd.SubName` parameter as input, so you know what name to look for; you obviously have to avoid name clashes with other subscriptions.

```
MQCLOSE(Hconn, &Hsub, MQCO_REMOVE_SUB, &CompCode, &Reason);
```

Closing both the subscription and the subscription queue is optional. In the example the subscription is closed, but not the queue. The `MQCLOSE MQCO_REMOVE_SUB` option is the default in this case anyway as the subscription is non-durable. Using `MQCO_KEEP_SUB` is an error.

Note: the subscription *queue* is not closed by `MQSUB`, and its handle, `Hobj`, remains valid until the queue is closed by `MQCLOSE` or `MQDISC`. If the application terminates prematurely, the queue and subscription are cleaned up by the queue manager sometime after application termination.


Parent topic: [Writing subscriber applications](#)

Related concepts


[Example 1: MQ Publication consumer](#)

[Example 3: Unmanaged MQ subscriber](#)

[Writing publisher applications](#)

 This build: January 26, 2011 10:56:23

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps10437_

4.2.3. Example 3: Unmanaged MQ subscriber

The unmanaged subscriber is an important class of subscriber application. With it, you combine the benefits of publish/subscribe with *control* of queuing and consumption of publications. The example demonstrates different ways of combining subscriptions and queues.

The unmanaged pattern is more commonly associated with *durable* subscriptions than *non-durable*. Typically the lifecycle of a subscription created by an unmanaged subscriber is independent of the lifecycle of the subscribing application itself. By making the subscription durable the subscription receives publications even when no subscribing application is active.

You can create durable *managed* subscriptions to achieve the same result, but some applications require more flexibility and control over queues and messages than is possible with a managed subscription. For a durable managed subscription, the queue manager creates a permanent queue for the publications that match the subscription topic. It deletes the queue and associated publications when the subscription is deleted.

Typically durable *managed* subscriptions are used if the lifecycle of the application and the subscription is essentially the same, but hard to guarantee. By making the subscription durable, and having the publisher create persistent publications, there are no lost messages should the queue manager or subscriber terminate prematurely and need to be recovered.

The queue manager implicitly opens the durable managed subscription queue for a subscriber in such a way that shared processing of the queue is not

possible. In addition, you cannot create more than one subscription for each managed queue and you may find the queues harder to manage because you have less control over the names of the queues. For these reasons, consider whether the *unmanaged* MQ subscriber is a better fit for applications requiring durable subscriptions than the *managed* MQ subscriber.

The code in [Figure 3](#) demonstrates an unmanaged durable subscription pattern. For illustration the code also creates unmanaged, non-durable subscriptions. The pattern facets exemplified by this code are,

1. On demand subscriptions: the subscription topic strings are dynamic. They are provided by the application when it runs.
2. Simplified subscription topic management: subscription topic management is simplified by defining the root part of the subscription topic string using an administratively defined topic. This hides the root part of the topic tree from the application. By hiding the root part a subscriber can be deployed to different topic trees.
3. Flexible subscription management: you can define a subscription either administratively, or create it on-demand in a subscriber program. There is no difference between administratively and programmatically created subscriptions, except an attribute that shows how the subscription was created. There is a third type of subscription that is created automatically by the queue manager for distribution of subscriptions. All subscriptions are displayed in the WebSphere MQ Explorer.
4. Flexible association of subscriptions with queues: a predefined local queue is associated with a subscription by the MQSUB function. There are different ways to use MQSUB to associate subscriptions with queues:
 - a. Associate a subscription with a queue having *no* existing subscriptions, MQSO_CREATE + (Hobj from MQOPEN).
 - b. Associate a *new* subscription with a queue having existing subscriptions, MQSO_CREATE + (Hobj from MQOPEN).
 - c. Move a existing subscription to a different queue, MQSO_ALTER + (Hobj from MQOPEN).
 - d. Resume an existing subscription associated with an existing queue, MQSO_RESUME + (Hobj = MQHO_NONE), or MQSO_RESUME + (Hobj = from MQOPEN of queue with existing subscription).
 - o By combining MQSO_CREATE | MQSO_RESUME | MQSO_ALTER in different combinations, you can cater for different input states of the subscription and the queue without having to code multiple versions of MQSUB with different sd.Options values.
 - o Alternatively, by coding a specific choice of MQSO_CREATE | MQSO_RESUME | MQSO_ALTER the queue manager returns an error ([Table 1](#)) if the states of the subscription and queue provided as input to MQSUB are inconsistent with the value of sd.Options. [Figure 9](#) shows the results of issuing MQSUB for Subscription X with different individual settings of the sd.Options flag, and passing it three different object handles.

Explore different inputs to the example program in [Figure 2](#) to become familiar with these different kinds of error. One common error, RC = 2440, that is not included in the cases listed in the table, is a subscription name error. It is commonly caused by passing a null or invalid subscription name with MQSO_RESUME or MQSO_ALTER.

5. Multiprocessing: you can share out of the work of reading publications to multiple consumers. The publications all go onto the single queue associated with the subscription topic. Consumers have a choice of opening the queue directly using MQOPEN or resuming the subscription using MQSUB.
6. Subscription concentration: multiple subscriptions can be created on the same queue. Be cautious with this capability as it can lead to "overlapping" subscriptions, and receiving the same publication multiple times. The MQSO_GROUP_SUB option eliminates duplicate publications caused by overlapping subscriptions.
7. Subscriber and consumer separation: As well as the three consumer models illustrated in the examples, another model is to separate the consumer from the subscriber. It is a variation of the unmanaged MQ Subscriber, but rather than issue the MQOPEN and MQSUB in the same program, one program subscribes to publications, and another program consumes them. For example, the subscriber might be part of a publish/subscribe cluster and the consumer attached to a queue manager outside the queue manager cluster. The consumer receives publications through standard distributed queuing by defining the subscription queue as a remote queue definition.

Understanding the behavior of MQSO_CREATE | MQSO_RESUME | MQSO_ALTER is important, especially if you plan to simplify your code by using combinations of these options. Study the table [Table 1](#) that shows the results of passing different queue handles to MQSUB, and the results of running the example program shown in [Figure 4](#) to [Figure 9](#).

The scenario used to construct the table has one subscription X and two queues, A and B. The subscription name parameter sd.SubName is set to X, the name of a subscription attached to queue A. Queue B has no subscription attached to it.

Examine the top left cell. MQSUB is passed subscription X and the queue handle to queue A .

1. MQSO_CREATE fails because the queue handle corresponds to the queue A which already has a subscription to X. Contrast this behavior to the cell to the right: there, the call succeeds because queue B does not have a subscription to X attached to it.
2. MQSO_RESUME succeeds because the queue handle corresponds to the queue A which already has a subscription to X. In contrast, the call fails in the cell to the right.
3. MQSO_ALTER behaves in a similar way to MQSO_RESUME with respect to opening the subscription and queue. However if the attributes contained within the subscription descriptor passed to MQSUB differ from the attributes of the subscription, MQSO_RESUME fails, whereas MQSO_ALTER succeeds as long as the program instance has permission to alter the attributes. Note that you can never change the topic string in a subscription; but rather than return an error, MQSUB ignores the topic name and topic string values in the subscription descriptor and uses the values in the existing subscription.

Next, look at the cell below. MQSUB is passed subscription X and the queue handle to queue B.

1. MQSO_CREATE succeeds and creates subscription X on queue B because this is a new subscription on queue B.
2. MQSO_RESUME fails. MQSUB looks for subscription X on queue B and does not find it, but rather than returning RC = 2428 - subscription X does not exist, it returns RC = 2019 - Subscription queue does not match queue object handle. The behavior of the third option MQSO_ALTER suggests the reason for this unexpected error. MQSUB expects the queue handle to point to a queue with a subscription. It checks this first before checking whether the subscription named in sd.SubName exists.
3. MQSO_ALTER succeeds, and moves the subscription from queue A to queue B.
4. A case that is not shown in the table is if the subscription name of the subscription on queue A does not match the subscription name in sd.SubName. That call fails with a RC = 2428 - subscription X does not exist on Queue A.

Table 1. Errors from MQSUB with different queue handles and subscription combinations

	Queue A Subscription X Queue B No subscription	Queue A No subscription Queue B No subscription
Hobj for Queue A passed to MQSUB	MQSO_CREATE RC = 2432 - Subscription X already exists on Queue A MQSO_RESUME Resumes subscription X on Queue A MQSO_ALTER Resumes subscription X on Queue A and makes permitted alterations	MQSO_CREATE Creates subscription X on Queue A MQSO_RESUME RC = 2428 - Subscription X does not exist on Queue A MQSO_ALTER RC = 2428 - Subscription X does not exist on Queue A
Hobj for Queue B passed to MQSUB	MQSO_CREATE Creates new subscription X on Queue B	MQSO_CREATE Creates new subscription X on Queue B

	MQSO_RESUME RC = 2019 - Subscription queue does not match queue object handle MQSO_ALTER Move subscription X from Queue A to Queue B	MQSO_RESUME RC = 2428 - subscription X does not exist on Queue B MQSO_ALTER RC = 2428 - subscription X does not exist on Queue B
MQHO_NONE passed to MQSUB	MQSO_CREATE RC = 2019 - Bad object handle: set MQSO_MANAGED flag to create a managed subscription and create a managed queue MQSO_RESUME Resumes subscription X on Queue A and returns Hobj to Queue A MQSO_ALTER Resumes subscription X on Queue A, returns Hobj to Queue A and makes permitted alterations	MQSO_CREATE RC = 2019 - Bad object handle: set MQSO_MANAGED flag to create a managed subscription and create a managed queue MQSO_RESUME RC = 2428 - No subscription X MQSO_ALTER RC = 2019 - Bad object handle: No queue A or B

Note: The compact coding style is intended for readability not production use.

Figure 1. Unmanaged MQ subscriber - part 1: declarations.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>

void inquireQname(MQHCONN HConn, MQHOBJ Hobj, MQCHAR48 qName);

int main(int argc, char **argv)
{
    MQCHAR48 topicNameDefault      = "STOCKS";
    char      topicStringDefault[]  = "IBM/PRICE";
    char      subscriptionNameDefault[] = "IBMSTOCKPRICESUB";
    char      subscriptionQueueDefault[] = "STOCKTICKER";
    char      publicationBuffer[101]; /* Allocate to receive messages */
    char      resTopicStrBuffer[151]; /* Allocate to resolve topic string */
    MQCHAR48 qmName = "";           /* Default queue manager */
    MQCHAR48 qName = "";           /* Allocate storage for MQINQ */

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle */
    MQHOBJ   Hobj = MQHO_NONE;           /* subscription queue handle */
    MQHOBJ   Hsub = MQSO_NONE;          /* subscription handle */
    MQLONG   CompCode = MQCC_OK;        /* completion code */
    MQLONG   Reason = MQRC_NONE;        /* reason code */
    MQLONG   messlen = 0;
    MQOD     od = {MQOD_DEFAULT};       /* Unmanaged subscription queue */
    MQSD     sd = {MQSD_DEFAULT};       /* Subscription Descriptor */
    MQMD     md = {MQMD_DEFAULT};       /* Message Descriptor */
    MQGMO    gmo = {MQGMO_DEFAULT};     /* get message options */
    MQLONG   sdOptions = MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE | MQSO_FAIL_IF QUIESCING;

    char *   topicName = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   subscriptionName = subscriptionNameDefault;
    char *   subscriptionQueue = subscriptionQueueDefault;
    char *   publication = publicationBuffer;
    char *   resTopicStr = resTopicStrBuffer;
    memset(resTopicStrBuffer, 0, sizeof(resTopicStrBuffer));

```

Figure 2. Unmanaged MQ subscriber - part 2: parameter handling.

```
    switch(argc){
        /* Replace defaults with args if provided */
        default:
            switch((argv[5][0])) {
                case('A'): sdOptions = MQSO_ALTER | MQSO_DURABLE | MQSO_FAIL_IF QUIESCING;
                    break;
                case('C'): sdOptions = MQSO_CREATE | MQSO_DURABLE | MQSO_FAIL_IF QUIESCING;
                    break;
                case('R'): sdOptions = MQSO_RESUME | MQSO_DURABLE | MQSO_FAIL_IF QUIESCING;
                    break;
                default: ;
            }
        case(5):
            if (strcmp(argv[4],"/") /* "/" invalid = No subscription */
                subscriptionQueue = argv[4];
            else {
                *subscriptionQueue = '\0';
                if (argc > 5) {
                    if (argv[5][0] == 'C') {
                        sdOptions = sdOptions + MQSO_MANAGED;
                    }
                }
                else
                    sdOptions = sdOptions + MQSO_MANAGED;
            }
        case(4):
            if (strcmp(argv[3],"/") /* "/" invalid = No subscription */
                subscriptionName = argv[3];
            else {
                *subscriptionName = '\0';
                sdOptions = sdOptions - MQSO_DURABLE;
            }
        case(3):
            if (strcmp(argv[2],"/") /* "/" invalid = No topic string */
                topicString = argv[2];
            else
                *topicString = '\0';
        case(2):
            if (strcmp(argv[1],"/") /* "/" invalid = No topic object */
                topicName = argv[1];

```


resumed without explicitly opening it with an MQOPEN.

```
MQCLOSE(Hconn, &Hsub, MQCO_NONE, &CompCode, &Reason);
```

The subscription is closed using the subscription handle. Depending on whether the subscription is durable or not, the subscription is closed with an implicit MQCO_KEEP_SUB or MQCO_REMOVE_SUB. You can close a durable subscription with MQCO_REMOVE_SUB, but you *cannot* close a non-durable subscription with MQCO_REMOVE_SUB. The action of MQCO_REMOVE_SUB is to remove the subscription which stops any further publications being sent to the subscription queue.

```
MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
```

No special action is taken if the subscription is unmanaged. If the queue is managed and the subscription closed with either an explicit or implicit MQCO_REMOVE_SUB, then all publications are purged from the queue and queue deleted at this point.

Results from the example illustrate aspects of publish/subscribe.

1. In [Figure 4](#) the example starts by publishing 130 on the NYSE/IBM/PRICE topic.

Figure 4. Publish 130 to NYSE/IBM/PRICE

```
W:\Subscribe3\Debug>..\Publish2\Debug\publishstock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0
```

2. In [Figure 5](#) execution of the example using default parameters receives the retained publication 130.

The provided topic object and topic string are ignored, as shown in [Figure 9](#). The topic object and topic string are always taken from the subscription object, when one is provided, and the topic string is immutable. The actual behavior of the example depends on the choice or combination of MQSO_CREATE, MQSO_RESUME, and MQSO_ALTER. In this example MQSO_RESUME is the option selected.

Figure 5. Receive the retained publication

```
W:\Subscribe3\Debug>solution3
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8206
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0
```

3. In [Figure 6](#) no publications are received, because the durable subscription has already received the retained publication. In this example, the subscription is resumed by providing only the subscription name without the queue name. If the queue name was provided, the queue would be opened first and the handle passed to MQSUB.

Note: The 2038 error from MQINQ is due to the implicit MQOPEN of STOCKTICKER by MQSUB not including the MQOO_INQUIRE option. Avoid the 2038 return code from MQINQ by opening the queue explicitly.

Figure 6. Resume subscription

```
W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE IBMSTOCKPRICESUB / Resume
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "IBMSTOCKPRICESUB" "" sd.Options=8204
MQINQ failed with Condition code 2 and Reason 2038
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from unknown queue
Completion code 0 and Return code 0
```

4. In [Figure 7](#), the example creates a non-durable unmanaged subscription using STOCKTICKER as the destination. Because this is a new subscription, it receives the retained publication.

Figure 7. Receive retained publication with new unmanaged non durable subscription

```
W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE / STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "" "STOCKTICKER" sd.Options=8194
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0
```

5. To demonstrate overlapping subscriptions, another publication is sent, changing the retained publication. Next, a new non-durable, unmanaged subscription is created by not providing a subscription name ([Figure 8](#)). The retained publication is received twice, once for the new subscription, and once for the durable IBMSTOCKPRICESUB subscription that is still active on the STOCKTICKER queue.

The example is an illustration it is the queue that has subscriptions, and not the application. Despite not referring to the IBMSTOCKPRICESUB subscription in this invocation of the application, the application receives the publication twice: once from the durable subscription that was created administratively, and once from the non-durable subscription created by the application itself.

Figure 8. Overlapping subscriptions

```
W:\Subscribe3\Debug>..\Publish2\Debug\publishstock
Provide parameters: TopicObject TopicString Publication
Publish "130" to topic "STOCKS" and topic string "IBM/PRICE"
Published "130" to topic string "NYSE/IBM/PRICE"
Completion code 0 and Return code 0

W:\Subscribe3\Debug>solution3 STOCKS IBM/PRICE / STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "STOCKS" "IBM/PRICE" "" "STOCKTICKER" sd.Options=8194
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Received publication "130"
Completion code 0 and Return code 0
```

6. In [Figure 9](#) the example demonstrates that providing a new topic string and an existing subscription does not result in a changed subscription.

- a. In the first case, Resume resumes the existing subscription, as you might expect, and ignores the changed topic string.
- b. In the second case, Alter causes an error, RC = 2510, Topic not alterable.
- c. In the third example, Create causes an error RC = 2432, Sub already exists.

Figure 9. Subscription topics cannot be changed

```
W:\Subscribe3\Debug>solution3 "" NASDAD/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Resume
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAD/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8204
Waiting 10 seconds for publications matching "NYSE/IBM/PRICE" from STOCKTICKER
Received publication "130"
Completion code 0 and Return code 0
```

```

W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Alter
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8201
Completion code 2 and Return code 2510

W:\Subscribe3\Debug>solution3 "" NASDAC/IBM/PRICE IBMSTOCKPRICESUB STOCKTICKER Create
Optional parameters: topicName, topicString, subscriptionName, subscriptionQueue, A(lter)|C(reate)|R(esume)
Values "" "NASDAC/IBM/PRICE" "IBMSTOCKPRICESUB" "STOCKTICKER" sd.Options=8202
Completion code 2 and Return code 2432

```

Parent topic: [Writing subscriber applications](#)

Related concepts

[Example 1: MQ Publication consumer](#)

[Example 2: Managed MQ subscriber](#)

[Writing publisher applications](#)

This build: January 26, 2011 10:56:21

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

cs10432_

4.3. Publish/subscribe lifecycles

Consider the lifecycles of topics, subscriptions, subscribers, publications, publishers and queues in designing publish/subscribe applications.

The lifecycle of an object, such as a subscription, starts with its creation and ends with its deletion. It may also include other states and changes that it goes through, such as temporary suspension, having parent and children topics, expiration and deletion.

Traditionally WebSphere MQ objects such as queues are created administratively, or by administrative programs using Programmable Command Format (PCF). Publish/subscribe is different in providing the `MQSUB` and `MQCLOSE` API verbs to create and delete subscriptions, having the concept of managed subscriptions that not only create and delete queues, but also clean up unconsumed messages, and having associations between administratively created topic objects and programmatically or administratively created topic strings.

This functional richness caters for a wide range of publish/subscribe requirements, and also simplifies designing some common patterns of publish/subscribe application. Managed subscriptions, for example, simplify both the programming and administration of a subscription that is intended to last only as long as the program that created it. Unmanaged subscriptions simplify programming where there is a looser connection between subscribing and consuming publications. Centrally created subscriptions are useful where the pattern is one of routing publication traffic to consumers based on a centralized model of control, for example sending flight information to automated gates, whereas programmatically created subscriptions might be used if gate staff are responsible for subscribing to the passengers records for that flight, by entering a flight number at a gate.

In this last example a managed durable subscription might be appropriate: managed, because the subscriptions are being created very often, and have a clear endpoint when the gate closes and the subscription can be programmatically removed; durable, to avoid losing a passenger record due to the gate subscriber program going down for one reason or another⁴. To initiate the publication of passenger records to the gate, a possible design would be for the gate application to both subscribe to the passenger records using the gate number, and publish the gate opening event using the gate number. The publisher responds to the gate opening event by publishing the passenger records - which might then also go to other interested parties, such as billing, to record the flight is taking place, and to customer services, to text notifications to passengers' mobile phones of the gate number.

The centrally managed subscription might use a durable unmanaged model, routing passenger lists to the gate using a predefined queue for each gate.

The following three examples of publish/subscribe lifecycles illustrate how managed non-durable, managed durable, and unmanaged durable subscribers interact with subscriptions, topics, queues, publishers and the queue manager, and how the responsibilities might be divided between administration and the subscriber programs.

Managed non-durable subscriber

[Figure 1](#) shows an application creating a managed non-durable subscription, getting two messages that are published to the topic identified in the subscription, and terminating. The interactions labeled in an italic grey font with dotted arrows are implicit.

Here are some points to note.

1. The application creates a subscription on a topic that has already been published to twice. When the subscriber receives its first publication, it receives the *second* publication which is the currently retained publication.
2. The queue manager creates a temporary subscription queue as well as creating a subscription for the topic.
3. The subscription has an expiry. When the subscription expires no more publications on the topic are sent to this subscription, but the subscriber continues to get messages published before the subscription expired. Publication expiry is not affected by subscription expiry.
4. The fourth publication is not placed on the subscription queue and consequently the last `MQGET` does not return a publication.
5. Although the subscriber closes its subscription, it does not close its connection to the queue or the queue manager.
6. The queue manager cleans up shortly after the application terminates. Because the subscription is managed and non-durable, the subscription queue is deleted.

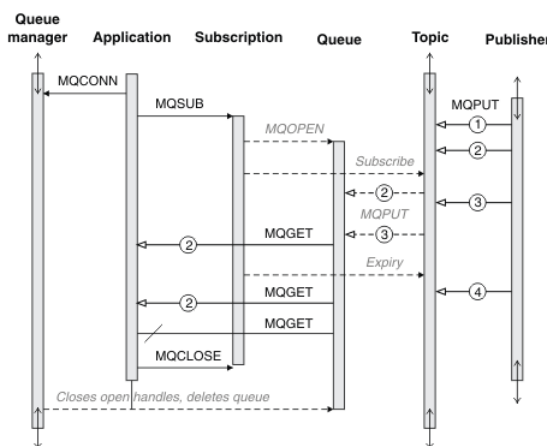


Figure 1. Managed non-durable subscriber lifelines

Managed durable subscriber

The managed durable subscriber takes the previous example a step further, and shows a managed subscription surviving the termination and restart of the subscribing application.

There are some new points to note.

1. In this example, unlike the last, the publication topic did not exist before it was defined in the subscription.
2. The first time the subscriber terminates, it closes the subscription with the option `MQCO_KEEP_SUB`. That is the default behavior for implicitly closing a managed durable subscription.
3. When the subscriber resumes the subscription, the subscription queue is reopened.
4. The new publication 2, placed on the queue before it is reopened, is available to `MQGET`, even after the subscription has been removed. Even though the subscription is durable, the subscriber reliably receives all messages sent by the publisher only if both the subscription is durable and the messages persistent. Message persistence depends on the setting of the `Persistent` field in the `MQMD` of the message sent by the publisher. A subscriber has no control over this.
5. Closing the subscription with the flag `MQCO_REMOVE_SUB` removes the subscription, stopping any further publications being placed on the subscription queue. When the subscription queue is closed, then the queue manager removes the unread publication 3, and then deletes the queue. The action is equivalent to administratively deleting the subscription.

Note: Do not delete the queue manually, or issue `MQCLOSE` with the option `MQCO_DELETE`, or `MQCO_PURGE_DELETE`. The visible implementation details of a managed subscription is not part of the supported WebSphere MQ interface. The queue manager manage cannot manage a subscription reliably unless it has complete control.

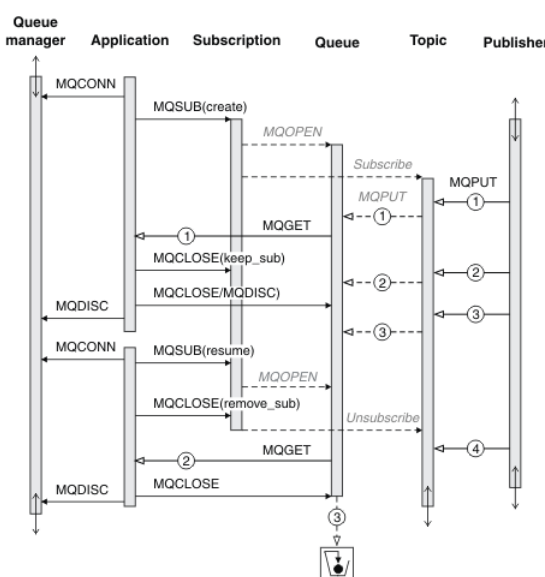


Figure 2. Managed durable subscriber lifelines

Unmanaged durable subscriber

An administrator is added in the third example: the unmanaged durable subscriber. It is a good example to show how the administrator might interact with a publish/subscribe application.

The points to note are listed.

1. The publisher puts a message, 1, to a topic that later becomes associated with the topic object that is used for subscription. The topic object defines a topic string that matches the topic that was published to by using wildcards.
2. The topic has a retained publication.
3. The administrator creates a topic object, a queue and a subscription. The topic object and queue need to be defined before the subscription.
4. The application opens the queue associated with the subscription and passes `MQSUB` the handle of the queue. It could, alternatively, simply open the subscription, passing it the queue handle `MQHO_NONE`. The converse is not true, it cannot resume a subscription by passing it only queue handle without a subscription name - a queue might have multiple subscriptions.
5. The application opens the subscription using the option `MQSO_RESUME` even though it is the first time it has opened the subscription. It is resuming an administratively created subscription.
6. The subscriber receives the retained publication, 1. Publication 2, although published before any publications were received by the subscriber, was published after the subscription started, and is the second publication on the subscription queue.

Note: If the retained publication is not published as a persistent message, then it is lost after queue manager restart.

- In this example the subscription is durable. It is possible for a program to create an unmanaged non-durable subscription; it should be obvious this is not something an administrator can do.
- The effect of the option `MQCO_REMOVE_SUB` on closing the subscription is to remove the subscription just as if the administrator had deleted it. This stops any further publications being sent to the queue, but does not affect publications that are already on the queue, even when the queue is closed, unlike a *managed* durable subscription.
- The administrator later deletes the remaining message, 3, and deletes the queue.

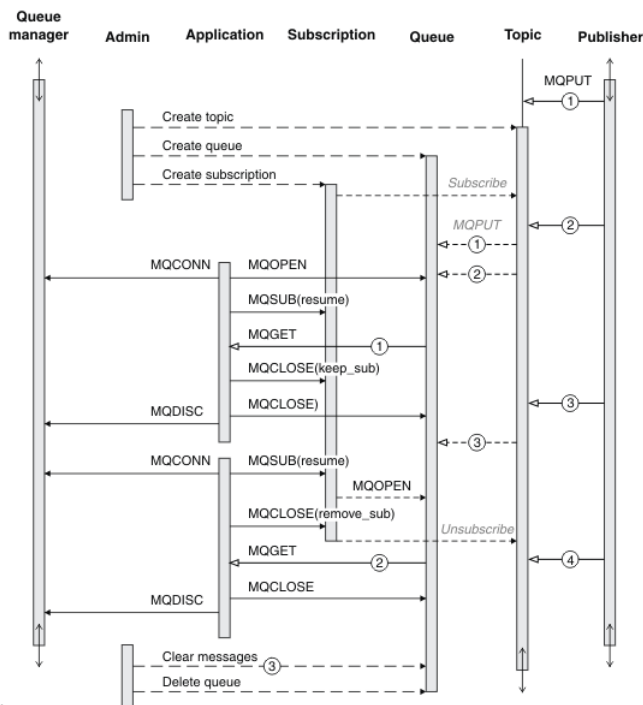


Figure 3. Unmanaged durable subscriber lifelines

A normal pattern for an unmanaged subscription is for queue and subscription housekeeping to be performed by the administrator. Typically one would not attempt to emulate the behavior of a managed subscriber and tidy up queues and subscriptions programmatically in application code. If you find yourself needing to write management logic, question whether you can achieve the same results using a managed pattern. It is not easy to write tightly synchronized, completely reliable management code. It is easier to tidy up later, either manually, or using an automated management program, when you can be sure that messages, subscriptions, and queues can be simply deleted, regardless of their state.

Parent topic: [Writing publish/subscribe applications](#)

The publisher must send the passenger records as persistent messages to avoid other possible failures, of course.

This build: January 26, 2011 10:56:23

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps10439_

4.4. Publish/subscribe message properties

Several message properties relate to Websphere MQ publish/subscribe messaging.

PubAccountingToken

This is the value that will be in the AccountingToken field of the Message Descriptor (MQMD) of all publication messages matching this subscription. AccountingToken is part of the identity context of the message. For more information about message context, see [Message context](#). For more information about the AccountingToken field in the MQMD, see [AccountingToken](#).

PubAppIdentityData

This is the value that will be in the AppIdentityData field of the Message Descriptor (MQMD) of all publication messages matching this subscription. AppIdentityData is part of the identity context of the message. For more information about message context, see [Message context](#). For more information about the AppIdentityData field in the MQMD, see [AppIdentityData](#).

If the option `MQSO_SET_IDENTITY_CONTEXT` is not specified, the AppIdentityData which will be set in each message published for this subscription is blanks, as default context information.

If the option `MQSO_SET_IDENTITY_CONTEXT` is specified, the PubAppIdentityData is being generated by the user and this field is an input field which contains the AppIdentityData to be set in each publication for this subscription.

PubPriority

This is the value that will be in the Priority field of the Message Descriptor (MQMD) of all publication messages matching this subscription. For more information about the Priority field in the MQMD, see [Priority](#).

The value must be greater than or equal to zero; zero is the lowest priority. The following special values can also be used:

- `MQPRI_PRIORITY_AS_Q_DEF` - When a subscription queue is provided in the Hobj field in the MQSUB call, and is not a managed handle, then the priority for the message is taken from the DefPriority attribute of this queue. If the queue so identified is a cluster queue or there is more than one definition in the queue-name resolution path then the priority is determined when the publication message is put to the queue as described for Priority in the MQMD in the *WebSphere® MQ Application Programming Reference*. If the MQSUB call uses a managed handle, the priority for the message is taken from the DefPriority attribute of the model queue associated with the topic subscribed to.
- `MQPRI_PRIORITY_AS_PUBLISHED` - The priority for the message is the priority of the original publication. This is the initial value of this field.

SelectionString

This variable length field will be returned on output from an MQSUB call using the MQSO_RESUME option, if a big enough buffer is provided. If the buffer provided on the call is not big enough (by the value in VSBufSize) only the length of the selection string will be returned in the VSLength field of the MQCHARV and the contents of the buffer will not be altered. The MQSUB call with the MQSO_RESUME option can then be issued again providing a buffer long enough to fit VSLength bytes.

SubCorrelId

Attention: ▶ a correlation identifier can only be passed between queue managers in a publish/subscribe cluster, not a hierarchy. ◀

All publications sent to match this subscription will contain this correlation identifier in the message descriptor. If multiple subscriptions use the same queue to get their publications from, using MQGET by correlation id allows only publications for a specific subscription to be obtained. This correlation identifier can either be generated by the queue manager or by the user.

If the option MQSO_SET_CORREL_ID is not specified, the correlation identifier is generated by the queue manager and this field is an output field which contains the correlation identifier which will be set in each message published for this subscription.

If the option MQSO_SET_CORREL_ID is specified, the correlation identifier is being generated by the user and this field is an input field which contains the correlation identifier to be set in each publication for this subscription. In this case, if the field contains MQCI_NONE, the correlation identifier which will be set in each message published for this subscription will be the correlation identifier created by the original put of the message.

If the option MQSO_GROUP_SUB is specified and the correlation identifier specified is the same as an existing grouped subscription using the same queue and an overlapping topic string, only the most significant subscription in the group is provided with a copy of the publication.

SubUserData

This is the subscription user data. The data provided on the subscription in this field will be included as the MQSubUserData message property of every publication sent to this subscription.

▶

Publication properties

[Table 1](#) lists the publication properties that are provided with a publication message.


You can access these properties directly from the **MQRFH2** folder, or retrieve them using **MQINQMP**. **MQINQMP** accepts either the property name or **MQRFH2** name as the name of the property to inquire on.

Table 1. Publication properties

Property name	MQRFH2 name	Type	Description
MQTopicString	mgps.Top	MQTYPE_STRING	Topic string
MQSubUserData	mgps.Sud	MQTYPE_STRING	Subscriber user data
MQIsRetained	mgps.Ret	MQTYPE_BOOLEAN	Retained publication
MQPubOptions	mgps.Pub	MQTYPE_INT32	Publication options
MQPubLevel	mgps.Pbl	MQTYPE_INT32	Publication level
MQPubTime	mgpse.Pts	MQTYPE_STRING	Publication time
MQPubSeqNum	mgpse.Seq	MQTYPE_INT32	Publication sequence number
MQPubStrIntData	mgpse.Sid	MQTYPE_STRING	String/Integer data added by the publisher
MQPubFormat	mgpse.Pfmt	MQTYPE_INT32	Message format: MQRFH1 MQRFH2 PCF

◀

Parent topic: ▶ [Writing publish/subscribe applications](#) ◀

 This build: January 26, 2011 10:56:58

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps22880_

4.5. Message ordering

For a particular topic, messages are published by the queue manager in the same order as they are received from publishing applications (subject to reordering based on message priority).

Message ordering normally means that each subscriber receives messages from a particular queue manager, on a particular topic, from a particular publisher in the order that they are published by that publisher.

However, as with all WebSphere® MQ messages, it is possible for messages, occasionally, to be delivered out of order. This can happen in the following situations:

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

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

Note: ▶ It is not possible to use grouped or segmented messages with Publish/Subscribe. ◀

Parent topic: ▶ [Writing publish/subscribe applications](#) ◀

 This build: January 26, 2011 10:56:23

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
 This topic's URL:
 ps10470_

4.6. Intercepting publications

You can intercept a publication, modify it, and then republish it before it reaches any other subscriber.

You might want to intercept a publication before it reaches a subscriber in order to do one of the following actions:

- Attach additional information to the message
- Block the message
- Transform the message

You can perform the same operation on each message or vary the operation depending on the subscription, the message, or the message header.

»Subscription levels◀

An intercepting subscriber is a normal subscriber and uses any of the normal publish/subscribe or WebSphere® MQ functions. Publications are delivered to subscribers with a higher subscription level first. Publications can be republished to reach subscribers with a lower subscription level.

»Publish exit - MQ PUBLISH_EXIT◀

You can configure a publish exit at the queue manager, to change the contents of a published message before it is received by subscribers. You can also change the message header or choose not to deliver the message to a subscription. Publish exits are not supported on z/OS®.

Parent topic: »Writing publish/subscribe applications◀

This build: January 26, 2011 10:56:45

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
 This topic's URL:
 ps20000_

4.6.1. Subscription levels

An intercepting subscriber is a normal subscriber and uses any of the normal publish/subscribe or WebSphere® MQ functions. Publications are delivered to subscribers with a higher subscription level first. Publications can be republished to reach subscribers with a lower subscription level.

After the message has been intercepted, it can be passed on to an application capable of doing message transformation, and then republished. The message is then sent to the original subscribers, or it is intercepted again by another subscription exit or by the Publish exit.

»To intercept a publication, use the **subscription level** subscription attribute. The interceptor then republishes the message so that it can be received by other subscribers. ◀

To ensure the interceptor receives the messages before any other subscribers, make sure it has the highest subscription level of all subscribers by using the *SubLevel* field in the MQSD. By default, subscribers have a *SubLevel* of 1.

- »If you have one intercepting subscriber on a topic it should be configured to subscribe at a *SubLevel* of 9.◀
- »If more than one intercepting application on a topic is required to receive the publication, set the *SubLevel* of each interceptor's subscription appropriately to determine the order in which these intercepting applications receive the publication. ◀

The interceptor with the highest *SubLevel* receives the publication first, after which it is republished and received by the subscription with the next highest *SubLevel*, and so on. When configuring multiple intercepting applications, there should be no more than one at each *SubLevel* value which republishes the message; otherwise duplicate publications will be sent to the final set of subscribing applications because more than one interceptor republishes the message to the next *SubLevel* down.

By default, applications publish to a topic using a *PubLevel* of 9. *PubLevel* is a field in the MQPMO. If there are any subscriptions with a *SubLevel* of 9, only those subscriptions are given a copy of the publication. If there are no subscriptions with a *SubLevel* of 9, the publications are given to all those subscriptions on this topic which have the highest *SubLevel*.

An intercepting application should make its subscription using the options described in [Table 1](#).

Table 1. Intercepting subscriber options

Subscription option	Notes
MQSO_SET_CORREL_ID and <i>SubCorrelId</i> set to MQCI_NONE	This ensures that the <i>CorrelId</i> in the publication message when it is re-published by the interceptor, is the one set by the original publisher, in case any subscriptions at a lower <i>SubLevel</i> has requested that. Attention: »a correlation identifier can only be passed between queue managers in a publish/subscribe cluster, not a hierarchy.◀
<i>PubPriority</i> set to MQPRI_PRIORITY_AS_PUBLISHED	This ensures that the <i>Priority</i> in the publication message when it is re-published by the interceptor, is the one set by the original publisher, in case any subscriptions at a lower <i>SubLevel</i> has requested that.

An intercepting application should process the publication message (for example, transform or encrypt it) and then republish it to the same topic at a *PubLevel* one lower than the *SubLevel* which intercepted it. For example, a subscription with a *SubLevel* of 9 should republish the message with a *PubLevel* of 8. To republish the message correctly, several pieces of information are required as shown in [Table 2](#), and the intercepting application should use the same MQMD as in the original message and use MQPMO_PASS_ALL_CONTEXT to ensure all information in that MQMD is preserved and passed on to the next application (ordinary subscriber or interceptor).

Table 2. MQMD values for republished messages

Republish message using MQPUT	Information in publication message
MQOD.ObjectString	Message property MQTopicString
MQPMO.Options should OR with the information in the message	Message property MQPubOptions

A maximum of 8 intercepting applications can be implemented (with subscription levels from 9 down to 2 inclusive). In this case the final recipient of the message will have a *SubLevel* of value 1.


You can have a subscriber with a *SubLevel* of value 0 that serves as a catchall if no other subscriber is interested in the message. This configuration can be useful because you can monitor the messages this subscriber receives and check why no other subscribers received it and whether it is correct that it not be received by anyone else.

Retained publications

►If the publication is put by the original application with put-message option MQPMO_RETAIN, it will not be retained if it is intercepted by a subscriber with a *SubLevel* that is greater than 1. To ensure that the instruction to retain this publication is preserved as the publication passes through an intercepting application, the MQPMO options are carried with the publication as a message property and must be used on the republishing MQPUT call by the intercepting application.◀

►A subscription that subscribes with a *SubLevel* other than the value 1 does not receive retained publications when it subscribes or as a result of an MQSUBRQ request.◀

Parent topic: ►[Intercepting publications](#)◀

 This build: January 26, 2011 10:56:46

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps20005_



4.6.2. Publish exit - MQ_PUBLISH_EXIT

You can configure a publish exit at the queue manager, to change the contents of a published message before it is received by subscribers. You can also change the message header or choose not to deliver the message to a subscription. Publish exits are not supported on z/OS®.

Purpose

The publish exit allows you to inspect and alter the messages delivered to subscribers. The publish exit can do the following operations:

- Examine the contents of the message published to each subscriber
- Modify the contents of the message published to each subscriber
- Alter the queue to which the message is put
- Choose not to deliver the message to the subscriber

Syntax

MQ_PUBLISH_EXIT(*ExitParms*, *PubContext*, *SubContext*)

Parameters

ExitParms

Type: MQPSXP - input/output

Exit parameter structure. This structure contains information relating to the invocation of the exit.

PubContext

Type: MQPBC - input

Publication context structure. This structure contains contextual information relating to the publisher of the publication.

SubContext

Type: MQSBC - input/output

Subscription context structure. This structure contains contextual information relating to the subscriber receiving the publication.

Refer to the definitions of the structures for full details about the information that is passed to the exit and which fields can be changed.

Usage notes

The function performed by the publish exit is defined by the provider of the exit. The exit, however, must conform to the rules defined in the associated control block MQPSXP.

No entry point called MQ_PUBLISH_EXIT is provided by the queue manager. However, a C language typedef declaration is provided for the name MQ_PUBLISH_EXIT which you can use to declare the user-written exit to ensure that the parameters are correct. The following example illustrates how to use the typedef declaration:

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

MQ_PUBLISH_EXIT MyPublishExit;

void MQENTRY MyPublishExit( PMQPSXP pExitParms,
                           PMQPBC  pPubContext,
                           PMQSBC  pSubContext )
{
    /* C language statements to perform the function of the exit */
}
```

The publish exit runs within the queue manager process, as a result of the following operations:

- A Publish operation where a message is delivered to one or more subscribers
- A Subscribe operation where one or more retained messages are delivered
- A Subscription Request operation where one or more retained messages are delivered

The first time that the publish exit is called for a connection, it is called with the *ExitReason* of MQXR_INIT. If the publish exit is invoked for MQXR_INIT then the publish exit is also invoked with the *ExitReason* for MQXR_TERM when the connection disconnects.

If the publish exit is configured but cannot be loaded when the queue manager is started, publish/subscribe message operations are inhibited for the queue manager. You must fix the problem or restart the queue manager before publish/subscribe messaging is re-enabled.

For each WebSphere® MQ connection that requires the publish exit, if the exit cannot be loaded or fails to initialize, publish/subscribe operations that require the publish exit are disabled and will fail with the WebSphere MQ reason code MQRC_PUBLISH_EXIT_ERROR.

Writing a publish exit

The context in which the publish exit is called is the connection by an application to the queue manager. In particular, the exit user area field is maintained

on a connection by connection basis for each application connection that is performing publish operations.

It is not valid for a publish exit to use any MQI calls other than those explicitly provided to manipulate message properties: MQBUFMH, MQCRTMH, MQDLTMH, MQDLTMP, MQMHBUF, MQINQMP, and MQSETMP.

If the publish exit changes the destination queue manager or queue name, no new authority check is carried out.

Compiling a publish exit

The publish exit is a dynamically loaded library; it can be thought of as a channel-exit. For information on how such exits are compiled, see [Channel-exit programs for messaging channels](#).

Sample publish exit

The sample exit program is called amqspse0.c. It writes an appropriate message to a log file depending on whether the exit was called for initialize, publish or terminate operations, and also demonstrates the use of the exit user area field to allocate and free storage appropriately.

Parameters

►MQPSXP - Publish exit data structure◄

The MQPSXP structure describes the information, relating to the invocation of the publish exit, that is passed to the publish exit.

►MQPBC - Publication context data structure◄

The MQPBC structure contains the contextual information, relating to the publisher of the publication, that is passed to the publish exit.


►MQSBC - Subscription context data structure◄

The MQSBC structure contains the contextual information, relating to the subscriber that is receiving the publication, that is passed to the publish exit.

►Publish exit configuration◄

Configure a publish exit by using three attributes in a stanza in the `qm.ini` file, or by setting appropriate registry keys on Windows platforms.

Parent topic: [►Intercepting publications◄](#)

 This build: January 26, 2011 10:57:14

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps29110_



4.6.2.1. MQPSXP - Publish exit data structure

The MQPSXP structure describes the information, relating to the invocation of the publish exit, that is passed to the publish exit.

The following table summarizes the fields in the structure:

Table 1. Fields in MQPSXP

Field	Description
StrucID	Structure identifier
Version	Structure version number
ExitId	Type of exit that is being called
ExitReason	Reason for invoking the exit
ExitResponse	Response from the exit
ExitResponse2	Secondary response from exit
Feedback	Feedback code
ExitUserArea	Exit user area
ExitData	Exit data
QMgrName	Name of local queue manager
Hconn	Connection handle
MsgDescPtr	Address of message descriptor (MQMD)
MsgHandle	Handle to message properties (MQHMSG)
MsgInPtr	Address of input message
MsgInLength	Length of input message
MsgOutPtr	Address of output message
MsgOutLength	Length of output message

Fields

StrucID (MQCHAR4)

Structure identifier. The value is as follows:

MQPSXP_STRUCID

Identifier for the publish exit parameter structure. For the C programming language, the constant MQPSXP_STRUC_ID_ARRAY is also defined; this has the same value as MQPSXP_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the exit.

Version (MQLONG)

Structure version number. The value is as follows:

MQPSXP_VERSION_1

Version 1 publish exit parameter structure. The constant MQPSXP_CURRENT_VERSION is also defined with the same value.

This is an input field to the exit.

ExitId (MQLONG)

Type of exit that is being called. The value is as follows:

MQXT_PUBLISH_EXIT

Publish exit.

This is an input field to the exit.

ExitReason (MQLONG)

Reason for invoking the exit. The possible values are:

MQXR_INIT

Exit initialization. This indicates that the exit for this connection is being called for initialization. It allows the exit to acquire and initialize the resources that it needs; for example, main storage.

MQXR_TERM

Exit termination. This indicates that the exit for this connection is about to be stopped. The exit must free any resources that it has acquired since it was initialized; for example, main storage.

This is an input field to the exit.

ExitResponse (MQLONG)

Response from the exit. This is set by the exit to specify that processing should continue normally. It must be one of the following values:

MQXCC_OK

Continue normally. This indicates that processing should continue normally. It is valid for all values of ExitReason. When ExitReason has the value MQXR_MSG, the DestinationQName and DestinationQMGrName fields of the MQSBC structure identify the destination to which the message should be sent.

MQXCC_FAILED

This indicates that the exit has failed. The publish operation will fail with completion code MQCC_FAILED and reason code [MQRC_PUBLISH_EXIT_ERROR](#).

MQXCC_SUPPRESS_FUNCTION

Suppress function. This indicates that the normal processing of the message should be discontinued. It is valid only when ExitReason has the value MQXR_PUBLICATION. The processing performed on the message is determined by the MQRO_DISCARD_MSG option in the Report field of the message descriptor of the message.

- If the MQRO_DISCARD_MSG option is specified, the message is not delivered to the subscriber.
- If the MQRO_DISCARD_MSG option is not specified, the message is placed on the dead-letter queue (undelivered-message queue). If there is no dead-letter queue, or the message cannot be placed successfully on the dead-letter queue, the publish operation fails for the subscriber. In this case, the failure to deliver the message to the subscriber is handled in accordance with the value of the PMSGDLV and NPMSGDLV topic object attributes as appropriate. For an explanation of these attributes, see the parameter descriptions for the [DEFINE_TOPIC](#) command.

This is an output field from the exit.

ExitResponse2 (MQLONG)

Reserved for future use.

Feedback (MQLONG)

Feedback code to be used if the exit returns MQXCC_SUPPRESS_FUNCTION in the ExitResponse field.

On input to the exit, this field always has the value MQFB_NONE. If the exit returns MQXCC_SUPPRESS_FUNCTION, the exit should set Feedback to the value to be used for the message when the queue manager places it on the dead-letter queue.

If the exit returns MQXCC_SUPPRESS_FUNCTION, but Feedback still has the value MQFB_NONE, the following feedback code is used: MQFB_STOPPED_BY_PUBSUB_EXIT - Message stopped by publish exit.

This is an input/output field to the exit.

ExitUserArea (MQBYTE16)

Exit user area. This is a field that is available for the exit to use. It is initialized to MQXUA_NONE (binary zero) on the first invocation of the exit for a connection, and thereafter any changes made to this field by the exit are preserved across invocations of the exit. The first invocation of the exit is indicated by the ExitReason field having the value MQXR_INIT. There is a separate ExitUserArea for each connection. The length of this field is given by MQ_EXIT_USER_AREA_LENGTH.

This is an input/output field to the exit.

ExitData (MQCHAR32)

Fixed exit data defined by the PublishExitData parameter of the stanza in the queue manager's initialization file. The data is padded with blanks to the full length of the field. If there is no fixed exit data defined in the initialization file, this field is completely blank. The length of this field is given by MQ_EXIT_DATA_LENGTH.

This is an input field to the exit.

QMGrName (MQCHAR48)

Name of the local queue manager. The name is padded with blanks to the full length of the field. The length of this field is given by MQ_Q_MGR_NAME_LENGTH.

This is an input field to the exit.

Hconn (MQHCONN)

Handle representing a connection to the queue manager. It is to be used only in association with manipulating message properties, and should be supplied to the set, inquire, or delete message property function calls as the required connection handle parameter.

This is an input field to the exit.

MsgDescPtr (PMQMD)

Address of message descriptor. This is the address of the message descriptor (MQMD) of the message being processed. The exit can change the contents of the message descriptor, so use it with care. In particular, in the case where the SubType field of the MQSBC structure is of value MQSUBTYPE_PROXY, the CorrelId field in the message descriptor must not be changed.

No message descriptor is passed to the exit if ExitReason is MQXR_INIT or MQXR_TERM; in these cases, MsgDescPtr is the null pointer.

This field is an input field to the exit.

MsgHandle (MQHMSG)

Handle to message properties. The message handle is used to set, inquire, or delete the message properties.

This is an input field to the exit.

MsgInPtr (PMQVOID)

The contents of this buffer can be modified by the exit; see the MsgOutPtr field.

This is an input field to the exit.

MsgInLength (MQLONG)

Length of input message data. This is the length in bytes of the message data passed to the exit. The address of the data is given by the MsgInPtr field.

This is an input field to the exit.

MsgOutPtr (PMQVOID)

Address of output message data. This is the address of a buffer containing the message data that is output from the exit. On input to the exit, this field is always null. On output from the exit, if the value is still null, the queue manager sends the message specified by the MsgInPtr field, with the length given by the MsgInLength field.

If the exit must modify the message data, use one of the following procedures:

- If the length of the data does not change, the data can be modified in the buffer addressed by the MsgInPtr field. In this case, do not change MsgOutPtr and MsgOutLength.
- If the modified data is shorter than the original data, the data can be modified in the buffer addressed by MsgInPtr. In this case MsgOutPtr must be set to the address of the input message buffer, and MsgOutLength set to the new length of the message data.
- If the modified data is, or might be, longer than the original data, the exit must obtain a buffer of the required size and copy the modified data into it. In this case MsgOutPtr must be set to the address of the new buffer, and MsgOutLength set to the new length of the message data. The exit is responsible for freeing the buffer when the exit is next called.

Note: Because MsgOutPtr is always the null pointer on input to the exit, the exit must save the address of the buffer it obtains, either in ExitUserArea, or in a control block whose address is saved in ExitUserArea, so that the exit can reuse or free and buffer allocated.

This field is an input/output field to the exit.

MsgOutLength (MQLONG)

Length of output message data. This is the length in bytes of the message data returned by the exit. On input to the exit, this field is always zero. On output from the exit, this field is ignored if MsgOutPtr is null. See the description of the MsgOutPtr field for information about modifying the message data.

This field is an input/output field to the exit.


C language declaration - MQPSXP

```
typedef struct tagMQPSXP {
    MQCHAR4    StrucId;           /* Structure identifier */
    MQLONG     Version;         /* Structure version number */
    MQLONG     ExitId;          /* Type of exit */
    MQLONG     ExitReason;      /* Reason for invoking exit */
    MQLONG     ExitResponse;    /* Response from exit */
    MQLONG     ExitResponse2;   /* Reserved */
    MQLONG     Feedback;       /* Feedback code */
    MQBYTE16   ExitUserArea;   /* Exit user area */
    MQCHAR32   ExitData;       /* Exit data */
    MQCHAR48   QMgrName;       /* Name of local queue manager */
    MQHCONN    Hconn;         /* Connection handle */
    MQHMSG     MsgHandle;      /* Handle to message properties */
    PMQMD      MsgDescPtr;     /* Address of message descriptor */
    PMQVOID    MsgInPtr;       /* Address of input message data */
    MQLONG     MsgInLength;    /* Length of input message data */
    PMQVOID    MsgOutPtr;     /* Address of output message data */
    MQLONG     MsgOutLength;   /* Length of output message data */
} MQPSXP;
```

Parent topic: [Publish exit - MQ_PUBLISH_EXIT](#)

Related information

[2557 \(09FD\) \(RC2557\): MQRC_PUBLISH_EXIT_ERROR](#)

 This build: January 26, 2011 10:57:14

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps29120_

**4.6.2.2. MQPBC - Publication context data structure**

The MQPBC structure contains the contextual information, relating to the publisher of the publication, that is passed to the publish exit.

The following table summarizes the fields in the structure:

Table 1. Fields in MQPBC

Field	Description
StrucID	Structure identifier
Version	Structure version number
PubTopicString	Publish topic string

Fields**StrucID (MQCHAR4)**

Structure identifier. The value is as follows:

MQPBC_STRUCID

Identifier for the publication context structure. For the C programming language, the constant MQPBC_STRUC_ID_ARRAY is also defined; this has the same value as MQPBC_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the exit.

Version (MQLONG)

Structure version number. The value is as follows:

MQPBC_VERSION_1

Version 1 publish exit parameter structure. The constant MQPBC_CURRENT_VERSION is also defined with the same value.

This is an input field to the exit.

PubTopicString (MQCHARV)


The topic string being published to.

This is an input field to the exit.

C language declaration - MQPBC

```
typedef struct tagMQPBC {
    MQCHAR4    StructId;           /* Structure identifier */
    MQLONG    Version;           /* Structure version number */
    MQCHARV   PubTopicString;    /* Publish topic string */
} MQPBC;
```

Parent topic: [Publish exit - MQ_PUBLISH_EXIT](#)

 This build: January 26, 2011 10:57:14

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps29130_



4.6.2.3. MQSBC - Subscription context data structure

The MQSBC structure contains the contextual information, relating to the subscriber that is receiving the publication, that is passed to the publish exit.

The following table summarizes the fields in the structure:

Table 1. Fields in MQSBC

Field	Description
StructID	Structure identifier
Version	Structure version number
DestinationQMgrName	Name of destination queue manager
DestinationQName	Name of destination queue
SubType	Type of subscription
SubOptions	Subscription options
ObjectName	Object name
ObjectString	Object string
SubTopicString	Subscription topic string
SubName	Subscription name
SubId	Subscription identifier
SelectionString	Address of selection string
SubLevel	Subscription level
PSProperties	Publish/subscribe properties

Fields

StructID (MQCHAR4)

Structure identifier. The value is as follows:

MQSBC_STRUCID

Identifier for the publish exit parameter structure. For the C programming language, the constant MQSBC_STRUC_ID_ARRAY is also defined; this has the same value as MQSBC_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the exit.

Version (MQLONG)

Structure version number. The value is as follows:

MQSBC_VERSION_1

Version 1 publish exit parameter structure. The constant MQSBC_CURRENT_VERSION is also defined with the same value.

This is an input field to the exit.

DestinationQMgrName (MQCHAR48)

Name of destination queue. This is the name of the queue to which the message is being sent. The name is padded with blanks to the full length of the field. The name can be altered by the exit. The length of this field is given by MQ_Q_NAME_LENGTH.

This is an input/output field to the exit.

DestinationQName (MQCHAR48)

Name of destination queue. This is the name of the queue to which the message is being sent. The name is padded with blanks to the full length of the field. The name can be altered by the exit. The length of this field is given by MQ_Q_NAME_LENGTH.

Note: All authority checks performed by the queue manager to determine if the message can be published have already been performed on the original values of the DestinationQMgrName and DestinationQName before the publish exit is called.

This is an input/output field to the exit.

SubType (MQLONG)

Type of subscription. Indicates how the subscription was created. Valid values are MQSUBTYPE_API, MQSUBTYPE_ADMIN and MQSUBTYPE_PROXY.

This is an input field to the exit.

SubOptions (MQLONG)

Subscription options. Refer to [Subscription options](#) for a description of values this field can take.

This is an input field to the exit.

ObjectName (MQCHAR48)

Name of the topic object as defined on the local queue manager. The length of this field is given by MQ_TOPIC_NAME_LENGTH.

This is an input field to the exit.

ObjectString (MQCHARV)

The object string, which is used to form part of the full topic name. Refer to the [ObjectString](#) field in the description of the MQSD subscription descriptor structure for further details.

This is an input field to the exit.

SubTopicString (MQCHARV)

The topic string as supplied by the subscriber.

This is an input field to the exit.

SubName (MQCHARV)

The subscription name.

This is an input field to the exit.

SubId (MQBYTE 24)

Specifies the unique internal subscription identifier.

This is an input field to the exit.

SelectionString (MQCHARV)

This is the string used to provide the selection criteria.

This is an input field to the exit.

SubLevel (MQLONG)

The level associated with the subscription. Refer to the [SubLevel](#) field in the description of the MQSD subscription descriptor structure for further details.

This is an input field to the exit.

PSProperties (MQLONG)


Publish/subscribe properties. Specifies how publish/subscribe related message properties are added to messages sent to this subscription. Possible values are MQPSPROP_NONE, MQPSPROP_COMPAT, MQPSPROP_RFH2, MQPSPROP_MSGPROP. See the [optional parameters for the Change, Copy, and Create Subscription programmable command format](#) for a description of these values.

This is an input field to the exit.

C language declaration - MQSBC

```
typedef struct tagMQSBC {
    MQCHAR4  StructId;           /* Structure identifier */
    MQLONG   Version;           /* Structure version number */
    MQCHAR48 DestinationQMgrName; /* Destination queue manager */
    MQCHAR48 DestinationQName;  /* Destination queue name */
    MQLONG   SubType;           /* Type of subscription */
    MQLONG   SubOptions;        /* Subscription options */
    MQCHAR48 ObjectName;        /* Object name */
    MQCHARV  ObjectString;      /* Object string */
    MQCHARV  SubTopicString;    /* Subscription topic string */
    MQCHARV  SubName;           /* Subscription name */
    MQBYTE24 SubId;             /* Subscription identifier */
    MQCHARV  SelectionString;   /* Subscription selection string */
    MQLONG   SubLevel;          /* Subscription level */
    MQLONG   PSProperties;      /* Publish/subscribe properties */
} MQSBC;
```

Parent topic: [Publish exit - MQ_PUBLISH_EXIT](#)

 This build: January 26, 2011 10:57:15

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps29140_



4.6.2.4. Publish exit configuration

Configure a publish exit by using three attributes in a stanza in the `qm.ini` file, or by setting appropriate registry keys on Windows platforms.

To configure the publish exit in the `qm.ini` file, specify the attributes and values under a stanza called `PublishSubscribe`. This stanza has the following attributes:

PublishExitPath=[path] | module_name

Module name and path containing the publish exit code. The maximum length of this field is MQ_EXIT_NAME_LENGTH. The default is no publish exit.

PublishExitFunction=function_name


Name of the function entry point into the module that contains the publish exit code. The maximum length of this field is MQ_EXIT_NAME_LENGTH.

On AS/400, if a program is used this field can be omitted.

PublishExitData=string

If the queue manager is calling a publish exit, it passes an MQPSXP structure as input. The data specified using the `PublishExitData` attribute is provided in the `ExitData` field of the structure. The string can be up to MQ_EXIT_DATA_LENGTH characters in length. The default is 32 blank characters.

Parent topic: [Publish exit - MQ_PUBLISH_EXIT](#)

Related information[2557 \(09FD\) \(RC2557\): MQRC_PUBLISH_EXIT_ERROR](#) This build: January 26, 2011 10:57:15[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps29150_



4.7. Publishing options

Several options are available that control the way messages are published.

Withholding reply-to information from subscribers

If you do not want subscribers to be able to reply to publications they receive, it is possible to withhold information in the ReplyToQ and ReplyToQgr fields of the MQMD by using the MQPMO_SUPPRESS_REPLYTO put-message option. If this option is used, the queue manager removes that information from the MQMD when it receives the publication before forwarding it to any subscribers.


This option cannot be used in combination with a report option that needs a ReplyToQ, if this is attempted the call will fail with MQRC_MISSING_REPLY_TO_Q.

Publication level

Using publication levels is a way of controlling which subscribers receive the publication. The publication level denotes the level of subscription targeted by the publication. Only subscriptions with the highest subscription level less than or equal to the publication's publication level, will receive the publication. This value must be in the range zero to nine; zero is the lowest publication level. The initial value of this field is 9. One of the uses of publication and subscription levels is to [intercept publications](#).

**Checking if a publication is not delivered to any subscribers**

To check if a publication has not been delivered to any subscribers, use the MQPMO_WARN_IF_NO_SUBS_MATCHED put-message option with the MQPUT call. If a completion code of MQCC_WARNING and a reason code MQRC_NO_SUBS_MATCHED are returned by the put operation, the publication was not delivered to any subscriptions. If the MQPMO_RETAIN option is specified on the put operation, the message is retained and delivered to any subsequently defined matching subscription. In a distributed publish/subscribe system, the MQRC_NO_SUBS_MATCHED reason code is returned only if there are no proxy subscriptions registered for the topic on the queue manager.

**Parent topic:** [Writing publish/subscribe applications](#) This build: January 26, 2011 10:56:57[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps22790_


4.8. Subscription options

[Subscriptions and message persistence](#)[Subscriptions and retained publications](#)

To control when retained publications are received, subscribers can use two subscription options.

[Grouping subscriptions](#)

You can group subscriptions to eliminate receiving duplicate publications from overlapping subscriptions.

Parent topic: [Writing publish/subscribe applications](#) This build: January 26, 2011 10:56:57[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps22800_


4.8.1. Subscriptions and message persistence



Queue managers maintain the persistence of the publications they forward to subscribers as set by the publisher. The publisher sets the persistence to be one of the following options:

- 0 Nonpersistent
- 1 Persistent
- 2 Persistence as queue/topic definition

For publish/subscribe, the publisher resolves the topic object and the topicString to a resolved topic object. If the publisher specifies Persistence as queue/topic definition, then the default persistence from the resolved topic object is set for the publication.

**Parent topic:** [Subscription options](#) This build: January 26, 2011 10:56:57[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps22805_

4.8.2. Subscriptions and retained publications

To control when retained publications are received, subscribers can use two subscription options.

Publish on request only, MQSO_PUBLICATIONS_ON_REQUEST

If you want a subscriber to have control of when it receives publications you can use the MQSO_PUBLICATIONS_ON_REQUEST subscription option. A subscriber can then control when it receives publications by using the MQSUBRQ call (specifying the Hsub handle that was returned from the original MQSUB call) to request that it is sent a topic's retained publication. Subscribers using the MQSO_PUBLICATIONS_ON_REQUEST subscription option, do not receive any non-retained publications.

►If you specify MQSO_PUBLICATIONS_ON_REQUEST you must use MQSUBRQ to retrieve any publication. If you do not use MQSO_PUBLICATIONS_ON_REQUEST you get messages as they are published.◀

If a subscriber uses the MQSUBRQ call and uses wildcards in the subscription's topic, the subscription might match multiple topics or nodes on a topic tree, all of whose retained messages (if any exist) will be sent to the subscriber.

This option can be particularly helpful when used with durable subscriptions because a queue manager will continue to send publications to a subscriber if it subscribed durably even if that subscriber application is not running. This could lead to a buildup of messages on the subscriber queue. This build up can be avoided if the subscriber registers using the MQSO_PUBLICATIONS_ON_REQUEST option. Alternatively, you can use non-durable subscriptions if appropriate to your application to avoid a build up of unwanted messages.

If a subscription is durable and a publisher uses retained publications the subscriber application can use the MQSUBRQ call to refresh its state information after a restart. The subscriber must then refresh its state periodically using the MQSUBRQ call.

No publications will be sent as a result of the MQSUB call using this option. A durable subscription that has been resumed following disconnection will use the MQSO_PUBLICATIONS_ON_REQUEST option if the original subscription was configured to use this option.

New publications only, MQSO_NEW_PUBLICATIONS_ONLY

If a retained publication exists on a topic, any subscribers that make a subscription after the publication was made will receive a copy of that publication. If a subscriber does not want to receive any publications that were made earlier than the subscription being made, the subscriber can use the MQSO_NEW_PUBLICATIONS_ONLY subscription option.

Parent topic: ►[Subscription options](#)◀

This build: January 26, 2011 10:56:54

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps22090_

4.8.3. Grouping subscriptions

You can group subscriptions to eliminate receiving duplicate publications from overlapping subscriptions.

Consider grouping subscriptions if you have set up a queue to receive publications and have a number of overlapping subscriptions feeding publications to the same queue. This situation is similar to the example in [Overlapping subscriptions](#).

You can avoid receiving duplicate publications by setting the option MQSO_GROUP_SUB when you subscribe to a topic. The result is that when more than one subscription in the group matches the topic of a publication, only one subscription is responsible for placing the publication on the queue. The other subscriptions that matched the publication topic are ignored.

The subscription responsible for placing the publication on the queue is chosen on the basis that it has the longest matching topic string, before encountering any wildcards. In other words, it can be thought of as the closest matching subscription. Its properties are propagated to the publication, including whether it has the MQSO_NOT_OWN_PUBS property. If it does, no publication is delivered to the queue, even though other matching subscriptions might not have the MQSO_NOT_OWN_PUBS property.

You cannot place *all* your subscriptions in a single group to eliminate duplicate publications. Grouped subscriptions must fulfill these conditions:

1. None of the subscriptions are managed.
2. A group of subscriptions deliver publications to the same queue.
3. Each subscription must be at the same subscription level.
4. The publication message for each subscription in the group has the same correlation identifier.
To ensure each subscription results in a publication message with the same correlation identifier, set MQSO_SET_CORREL_ID to create your own correlation identifier in the publication, and set the same value in the *SubCorrelId* field in each subscription. Do not set *SubCorrelId* to the value MQCI_NONE.

See ►[MQSO_GROUP_SUB](#)◀*Application programming reference* for more information.

Parent topic: ►[Subscription options](#)◀

This build: January 26, 2011 10:56:57

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

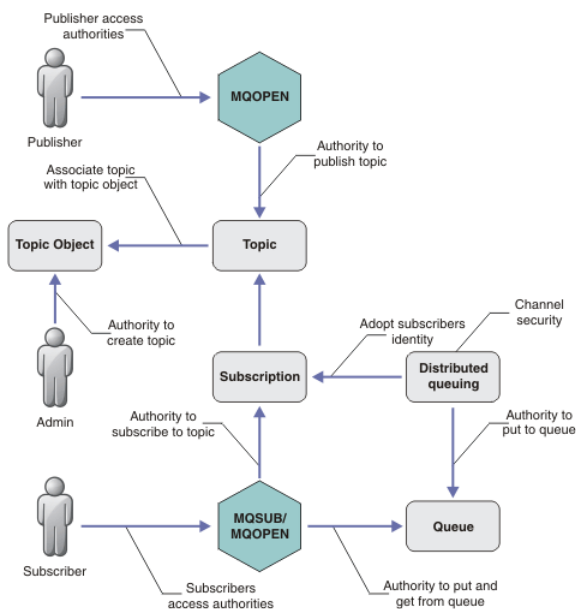
© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps22830_

5. Publish/subscribe security

The components and interactions that are involved in publish/subscribe are described as an introduction to the more detailed explanations and examples that follow.

There are a number of components involved in publishing and subscribing to a topic. Some of the security relationships between them are illustrated in [Figure 1](#) and described below.

Figure 1. Publish/subscribe security relationships



Topics

Topics are identified by topic strings, and are typically organized into trees, see [Topic trees](#). You need to associate a topic with a topic object to control access to the topic. [Topic security model](#) explains how you secure topics using topic objects.

Administrative topic objects

You can control who has access to a topic, and for what purpose, by using the command `setmqaut` with a list of administrative topic objects. See the examples, [Grant access to a user to subscribe to a topic](#) and [Grant access to a user to publish to a topic](#). For controlling access to topic objects on z/OS, see [Profiles for topic security](#).

Subscriptions

Subscribe to one or more topics by creating a subscription supplying a topic string, which can include wildcards, to match against the topic strings of publications. For further details, see:

Subscribe using a topic object

[Subscribing using the topic object name](#)

Subscribe using a topic

[Subscribing using a topic string where the topic node does not exist](#)

Subscribe using a topic with wildcards

[Subscribing using a topic string that contains wildcard characters](#)

A subscription contains information about the identity of the subscriber and the identity of the destination queue on to which the publications are to be placed. It also contains information about how the publication is to be placed on the destination queue.

As well as defining which subscribers have the authority to subscribe to certain topics, you can restrict subscriptions to being used by an individual subscriber. You can also control what information about the subscriber is used by the queue manager when publications are placed on to the destination queue. See [Subscription security](#).

Queues

The destination queue is an important queue to secure. It is local to the subscriber, and publications that matched the subscription are placed onto it. You need to consider access to the destination queue from two perspectives:

1. Putting a publication on to the destination queue.
2. Getting the publication off the destination queue.

The queue manager puts a publication onto the destination queue using an identity provided by the subscriber. The subscriber, or a program that has been delegated the task of getting publications, takes messages off the queue. See [Authority to destination queues](#).

There are no topic object aliases, but you can use an alias queue as the alias for a topic object. If you do so, as well as checking authority to use the topic for publish or subscribe, the queue manager checks authority to use the queue.

Distributed publish/subscribe security

Your permission to publish or subscribe to a topic is checked on the local queue manager using local identities and authorizations. Authorization does not depend on whether the topic is defined or not, nor where it is defined. Consequently, you need to perform topic authorization on every queue manager in a cluster when clustered topics are used.

Note: The security model for topics differs from the security model for queues. You can achieve the same result for queues by defining a queue alias locally for every clustered queue.

Queue managers exchange subscriptions in a cluster. In most WebSphere MQ cluster configurations, channels are configured with `PUTAUT=DEF` to place messages onto target queues using the authority of the channel process. You can modify the channel configuration to use `PUTAUT=CTX` to require the subscribing user to have authority to propagate a subscription onto another queue manager in a cluster.

[Distributed publish/subscribe security](#) describes how to change your channel definitions to control who is allowed to propagate subscriptions onto other servers in the cluster.

Authorization

You can apply authorization to topic objects, just like queues and other objects. There are three authorization operations, `pub`, `sub`, and `resume` that you can apply only to topics. The details are described in [Specifying authorities for different object types](#).

Function calls

In publish and subscribe programs, like in queued programs, authorization checks are made when objects are opened, created, changed, or deleted. Checks are not made when `MQPUT` or `MQGET` MQI calls are made to put and get publications.

To publish a topic, perform an `MQOPEN` on the topic, which performs the authorization checks. Publish messages to the topic handle using the `MQPUT` command, which performs no authorization checks.

To subscribe to a topic, typically you perform an `MQSUB` command to create or resume the subscription, and also to open the destination queue to receive publications. Alternatively, perform a separate `MQOPEN` to open the destination queue, and then perform the `MQSUB` to create or resume the subscription.

Whichever calls you use, the queue manager checks that you can subscribe to the topic and get the resulting publications from the destination queue. If the destination queue is unmanaged, authorization checks are also made that the queue manager is able to place publications on the destination queue. It uses the identity it adopted from a matching subscription. It is assumed that the queue manager is always able to place publications onto managed destination queues.

Roles

Users are involved in four roles in running publish/subscribe applications:

1. Publisher
2. Subscriber
3. Topic administrator
4. WebSphere MQ Administrator - member of group `mqm`

Define groups with appropriate authorizations corresponding to the publish, subscribe, and topic administration roles. You can then assign principals to these groups authorizing them to perform specific publish and subscribe tasks.

In addition, you need to extend the administrative operations authorizations to the administrator of the queues and channels responsible for moving publications and subscriptions.

Topic security model

Only defined topic objects can have associated security attributes. For a description of topic objects see [Administrative topic objects](#). The security attributes specify whether a specified user ID, or security group, is permitted to perform a subscribe or a publish operation on each topic object.

The security attributes are associated with the appropriate administration node in the topic tree. When an authority check is made for a particular user ID during a subscribe or publish operation, the authority granted is based on the security attributes of the associated topic tree node.

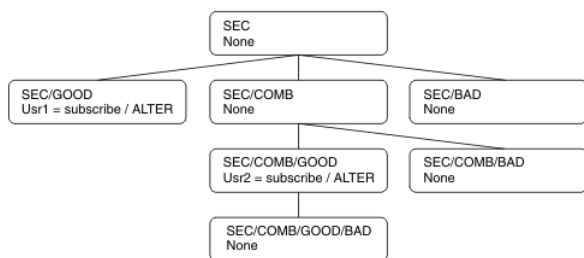
The security attributes are an access control list, indicating what authority a particular operating system user ID or security group has to the topic object.

Consider the following example where the topic objects have been defined with the security attributes, or authorities shown:

Table 1. Example topic object authorities

Topic name	Topic string	Authorities - not z/OS@	z/OS authorities
SECR00T	SEC	none	NONE
SECG00D	SEC/GOOD	usr1+subscribe	ALTER Hlq.SUBSCRIBE.SECGOOD
SECBAD	SEC/BAD	none	NONE Hlq.SUBSCRIBE.SECBAD
SECCOMB	SEC/COMB	none	NONE Hlq.SUBSCRIBE.SECCOMB
SECCOMBB	SEC/COMB/GOOD/BAD	none	NONE Hlq.SUBSCRIBE.SECCOMBB
SECCOMBG	SEC/COMB/GOOD	usr2+subscribe	▶ALTER◀ Hlq.SUBSCRIBE.SECCOMBG
SECCOMBN	SEC/COMB/BAD	none	NONE Hlq.SUBSCRIBE.SECCOMBN

The topic tree with the associated security attributes at each node can be represented as follows:



The examples listed give the following authorizations:

- At the root node of the tree /SEC, no user has authority at that node.
- `usr1` has been granted subscribe authority to the object /SEC/GOOD
- `usr2` has been granted subscribe authority to the object /SEC/COMB/GOOD

Subscribing using the topic object name

When subscribing to a topic object by specifying the MQCHAR48 name, the corresponding node in the topic tree is located. If the security attributes associated with the node indicate that the user has authority to subscribe, then access is granted.

If the user is not granted access, the parent node in the tree determines if the user has authority to subscribe at the parent node level. If so, then access is granted. If not, then the parent of that node is considered. The recursion continues until a node is located that grants subscribe authority to the user. The recursion stops when the root node is considered without authority having been granted. In the latter case, access is denied.

In short, if any node in the path grants authority to subscribe to that user or application, the subscriber is allowed to subscribe at that node, or anywhere below that node in the topic tree.

The root node in the example is SEC.

The user is granted subscribe authority if the access control list indicates that the user ID itself has authority, or that an operating system security group of which the user ID is a member has authority.

So, for example:

- If `usr1` tries to subscribe, using a topic string of `SEC/GOOD`, the subscription would be allowed as the user ID has access to the node associated with that topic. However, if `usr1` tried to subscribe using topic string `SEC/COMB/GOOD` the subscription would not be allowed as the user ID does not have access to the node associated with it.
- If `usr2` tries to subscribe, using a topic string of `SEC/COMB/GOOD` the subscription would be allowed as the user ID has access to the node associated with the topic. However, if `usr2` tried to subscribe to `SEC/GOOD` the subscription would not be allowed as the user ID does not have access to the node associated with it.
- If `usr2` tries to subscribe using a topic string of `SEC/COMB/GOOD/BAD` the subscription would be allowed because the user ID has access to the parent node `SEC/COMB/GOOD`.
- If `usr1` or `usr2` tries to subscribe using a topic string of `/SEC/COMB/BAD`, neither would be allowed as they do not have access to the topic node associated with it, or the parent nodes of that topic.

A subscribe operation specifying the name of a topic object that does not exist results in an `MQRC_UNKNOWN_OBJECT_NAME` error.

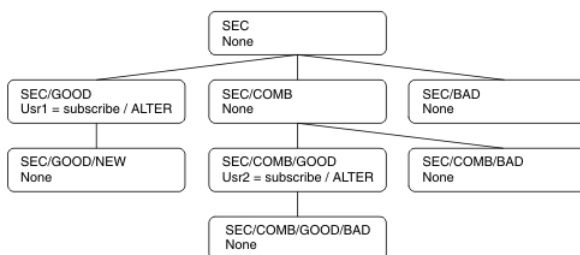
Subscribing using a topic string where the topic node exists

The behavior is the same as when specifying the topic by the `MQCHAR48` object name.

Subscribing using a topic string where the topic node does not exist

Consider the case of an application subscribing, specifying a topic string representing a topic node that does not currently exist in the topic tree. The authority check is performed as outlined in the previous section. The check starts with the parent node of that which is represented by the topic string. If the authority is granted, a new node representing the topic string is created in the topic tree.

For example, `usr1` tries to subscribe to a topic `SEC/GOOD/NEW`. Authority is granted as `usr1` has access to the parent node `SEC/GOOD`. A new topic node is created in the tree as the following diagram shows. The new topic node is not a topic object it does not have any security attributes associated with it directly; the attributes are inherited from its parent.



Subscribing using a topic string that contains wildcard characters

Consider the case of subscribing using a topic string that contains a wildcard character. The authority check is made against the node in the topic tree that matches the fully qualified part of the topic string.

So, if an application subscribes to `SEC/COMB/GOOD/*`, an authority check is carried out as outlined in the previous two sections on the node `SEC/COMB/GOOD` in the topic tree.

Similarly, if an application needs to subscribe to `SEC/COMB/*/GOOD`, an authority check is carried out on the node `SEC/COMB`.

Authority to destination queues

When subscribing to a topic, one of the parameters is the handle `hobj` of a queue that has been opened for output to receive the publications.

If `hobj` is not specified, but is blank, a managed queue is created if the following conditions apply:

- The `MQSO_MANAGED` option has been specified.
- The subscription does not exist.
- Create is specified.

If `hobj` is blank, and you are altering or resuming an existing subscription, the previously provided destination queue could be either managed or unmanaged.

The application or user making the `MQSUB` request must have the authority to put messages to the destination queue it has provided; in effect authority to have published messages put on that queue. The authority check follows the existing rules for queue security checking.

The security checking includes alternate user ID and context security checks where required. To be able to set any of the Identity context fields you must specify the `MQSO_SET_IDENTITY_CONTEXT` option as well as the `MQSO_CREATE` or `MQSO_ALTER` option. You cannot set any of the Identity context fields on an `MQSO_RESUME` request.

If the destination is a managed queue, no security checks are performed against the managed destination. If you are allowed to subscribe to a topic it is assumed that you can use managed destinations.

Publishing using the topic name or topic string where the topic node exists

The security model for publishing is the same as that for subscribing, except that there are no wildcard characters in the topic string case to consider.

The authorities to publish and subscribe are distinct. A user or group can have the authority to do one without necessarily being able to do the other.

When publishing to a topic object by specifying either the `MQCHAR48` name or the topic string, the corresponding node in the topic tree is located. If the security attributes associated with the topic node indicates that the user has authority to publish, then access is granted.

If access is not granted, the parent node in the tree determines if the user has authority to publish at that level. If so, then access is granted. If not, the recursion continues until a node is located which grants publish authority to the user. The recursion stops when the root node is considered without authority having been granted. In the latter case, access is denied.

In short, if any node in the path grants authority to publish to that user or application, the publisher is allowed to publish at that node or anywhere below that node in the topic tree.

Publishing using the topic name or topic string where the topic node does not exist

As with the subscribe operation, when an application publishes, specifying a topic string representing a topic node that does not currently exist in the topic tree, the authority check is performed starting with the parent of the node represented by the topic string. If the authority is granted, a new node representing the topic string is created in the topic tree.

Publishing using an alias queue that resolves to a topic object

If you publish using an alias queue that resolves to a topic object then security checking occurs on both the alias queue and the underlying topic to which it resolves.

The security check on the alias queue verifies that the user has authority to put messages on that alias queue and the security check on the topic verifies that the user can publish to that topic. When an alias queue resolves to another queue, checks are *not* made on the underlying queue. Authority checking is performed differently for topics and queues.

Closing a subscription

There is additional security checking if you close a subscription using the `MQCO_REMOVE_SUB` option if you did not create the subscription under this handle.

A security check is performed to ensure that you have the correct authority to do this as the action results in the removal of the subscription. If the security attributes associated with the topic node indicate that the user has authority, then access is granted. If not, then the parent node in the tree is considered to determine if the user has authority to close the subscription. The recursion continues until either authority is granted or the root node is reached.

Defining, altering, and deleting a subscription

No subscribe security checks are performed when a subscription is created administratively, rather than using an `MQSUB` API request. The administrator has already been given this authority through the command.

Security checks are performed to ensure that publications can be put on the destination queue associated with the subscription. The checks are performed in the same way as for an `MQSUB` request.

The user ID that is used for these security checks depends upon the command being issued. If the `SUBUSER` parameter is specified it affects the way the check is performed, as shown in [Table 2](#):

Table 2. User IDs used for security checks for commands

Command	SUBUSER specified and blank	SUBUSER specified and completed	SUBUSER not specified
DEFINE	Use the administrator ID	Use the user ID specified in <code>SUBUSER</code>	Use the administrator ID
ALTER	Use the administrator ID	Use the user ID specified in <code>SUBUSER</code>	Use the user ID from the existing subscription

The only security check performed when deleting subscriptions using the `DELETE SUB` command is the command security check.

[Example publish/subscribe security setup](#)

This section describes a scenario that has access control setup on topics in a way that allows the security control to be applied as required.

[Subscription security](#)


Parent topic: [Publish/Subscribe User's Guide](#)

Related concepts

[Topic trees](#)

[Distributed publish/subscribe security](#)

[Administrative topic objects](#)

 This build: January 26, 2011 10:56:45

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps19550_

5.1. Example publish/subscribe security setup

This section describes a scenario that has access control setup on topics in a way that allows the security control to be applied as required.

[Grant access to a user to subscribe to a topic](#)

This topic is the first one in a list of tasks that tells you how to grant access to topics by more than one user.

[Grant access to a user to subscribe to a topic deeper within the tree](#)

This topic is the second in a list of tasks that tells you how to grant access to topics by more than one user.

[Grant another user access to subscribe to only the topic deeper within the tree](#)

This topic is the third in a list of tasks that tells you how to grant access to subscribe to topics by more than one user.

[Change access control to avoid additional messages](#)

This topic is the fourth in a list of tasks that tells you how to grant access to subscribe to topics by more than one user and to avoid additional RACF® ICH408I messages on z/OS®.

[Grant access to a user to publish to a topic](#)

This topic is the first one in a list of tasks that tells you how to grant access to publish topics by more than one user.

[Grant access to a user to publish to a topic deeper within the tree](#)

This topic is the second in a list of tasks that tells you how to grant access to publish to topics by more than one user.

[Grant access for publish and subscribe](#)

This topic is the last in a list of tasks that tells you how to grant access to publish and subscribe to topics by more than one user.

Parent topic: [Publish/subscribe security](#)

This build: January 26, 2011 10:56:42

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps19512_

5.1.1. Grant access to a user to subscribe to a topic

This topic is the first one in a list of tasks that tells you how to grant access to topics by more than one user.

About this task

This task assumes that no administrative topic objects exist, nor have any profiles been defined for subscription or publication. The applications are creating new subscriptions, rather than resuming existing ones, and are doing so using the topic string only.

An application can make a subscription by providing a topic object, or a topic string, or a combination of both. Whichever way the application selects, the effect is to make a subscription at a certain point in the topic tree. If this point in the topic tree is represented by an administrative topic object, a security profile is checked based on the name of that topic object.

Figure 1. Topic object access example

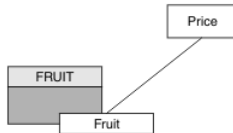


Table 1. Example topic object access

Topic	Subscribe access required	Topic object
Price	No user	None
Price/Fruit	USER1	FRUIT

Define a new topic object as follows:

Procedure

- Issue the MQSC command `DEF TOPIC(FRUIT) TOPICSTR('Price/Fruit')`.
- Grant access as follows:
 - z/OS®. Grant access to USER1 to subscribe to topic "Price/Fruit" by granting the user access to the `hlq.SUBSCRIBE.FRUIT` profile. Do this, using the following RACF® commands:


```
RDEFINE MXTOPIC hlq.SUBSCRIBE.FRUIT UACC(NONE)
PERMIT hlq.SUBSCRIBE.FRUIT CLASS(MXTOPIC) ID(USER1) ACCESS(ALTER)
```
 - Other platforms. Grant access to USER1 to subscribe to topic "Price/Fruit" by granting the user access to the `FRUIT` profile. Do this, using the following `setmqaut` command:


```
setmqaut -t topic -n FRUIT -p USER1 +sub
```

Results

When USER1 attempts to subscribe to topic "Price/Fruit" the result is success.

When USER2 attempts to subscribe to topic "Price/Fruit" the result is failure with an `MQRC_NOT_AUTHORIZED` message, together with:

- On z/OS, the following messages seen on the console that show the full security path through the topic tree that has been attempted:

```

ICH408I USER(USER2 ) ...
hlq.SUBSCRIBE.FRUIT ...

ICH408I USER(USER2 ) ...
hlq.SUBSCRIBE.SYSTEM.BASE.TOPIC ...
  
```

- On other platforms, the following authorization event:

```

MQRC_NOT_AUTHORIZED
ReasonQualifier    MQRC_SUB_NOT_AUTHORIZED
UserIdentifier     USER2
AdminTopicNames   FRUIT, SYSTEM.BASE.TOPIC
TopicString       "Price/Fruit"
  
```

Note that this is an illustration of what you see; not all the fields.

Parent topic: [Example publish/subscribe security setup](#)

Related information

[Profiles for queue security \(z/OS\)](#)

This build: January 26, 2011 10:56:42

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps19515_

5.1.2. Grant access to a user to subscribe to a topic deeper within the tree

This topic is the second in a list of tasks that tells you how to grant access to topics by more than one user.

Before you begin

This topic uses the setup described in [Grant access to a user to subscribe to a topic](#).

About this task

If the point in the topic tree where the application makes the subscription is not represented by an administrative topic object, move up the tree until the closest parent administrative topic object is located. The security profile is checked, based on the name of that topic object.

Figure 1. Example of granting access to a topic within a topic tree

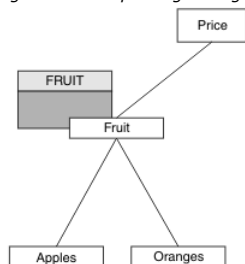


Table 1. Access requirements for example topics and topic objects

Topic	Subscribe access required	Topic object
Price	No user	None
Price/Fruit	USER1	FRUIT
Price/Fruit/Apples	USER1	
Price/Fruit/Oranges	USER1	

In the previous task USER1 was granted access to subscribe to topic "Price/Fruit" by granting it access to the hlq.SUBSCRIBE.FRUIT profile on z/OS® and subscribe access to the FRUIT profile on other platforms. This single profile also grants USER1 access to subscribe to "Price/Fruit/Apples", "Price/Fruit/Oranges" and "Price/Fruit/#".

When USER1 attempts to subscribe to topic "Price/Fruit/Apples" the result is success.

When USER2 attempts to subscribe to topic "Price/Fruit/Apples" the result is failure with an MQRQ_NOT_AUTHORIZED message, together with:

- On z/OS, the following messages seen on the console that show the full security path through the topic tree that has been attempted:

```

ICH408I USER(USER2 ) ...
hlq.SUBSCRIBE.FRUIT ...

ICH408I USER(USER2 ) ...
hlq.SUBSCRIBE.SYSTEM.BASE.TOPIC ...
    
```

- On other platforms, the following authorization event:

```

MQRQ_NOT_AUTHORIZED
ReasonQualifier MQRQ_SUB_NOT_AUTHORIZED
UserIdentifier USER2
AdminTopicNames FRUIT, SYSTEM.BASE.TOPIC
TopicString "Price/Fruit/Apples"
    
```

Note the following:

- The messages you receive on z/OS are identical to those received in the previous task as the same topic objects and profiles are controlling the access.
- The event message you receive on other platforms is similar to the one received in the previous task, but the actual topic string is different.

Parent topic: [Example publish/subscribe security setup](#)

Related information

[Profiles for queue security \(z/OS\)](#)

This build: January 26, 2011 10:56:42

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps19520_

5.1.3. Grant another user access to subscribe to only the topic deeper within the tree

This topic is the third in a list of tasks that tells you how to grant access to subscribe to topics by more than one user.

Before you begin

This topic uses the setup described in [Grant access to a user to subscribe to a topic deeper within the tree](#).

About this task

In the previous task USER2 was refused access to topic "Price/Fruit/Apples". This topic tells you how to grant access to that topic, but not to any other topics.

Figure 1. Granting access to specific topics within a topic tree

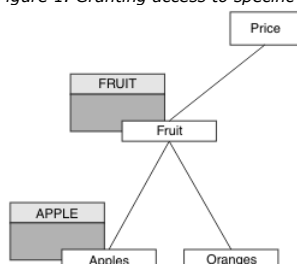


Table 1. Access requirements for example topics and topic objects

Topic	Subscribe access required	Topic object
Price	No user	None
Price/Fruit	USER1	FRUIT
Price/Fruit/Apples	USER1	
Price/Fruit/Oranges	USER1	

Topic	Subscribe access required	Topic object
Price	No user	None
Price/Fruit	USER1	FRUIT
Price/Fruit/Apples	USER1 and USER2	APPLE
Price/Fruit/Oranges	USER1	

Define a new topic object as follows:

Procedure

1. Issue the MQSC command `DEF TOPIC(APPLE) TOPICSTR('Price/Fruit/Apples')`.

2. Grant access as follows:

a. z/OS@.

In the previous task USER1 was granted access to subscribe to topic "Price/Fruit/Apples" by granting the user access to the `hlq.SUBSCRIBE.FRUIT` profile.

This single profile also granted USER1 access to subscribe to "Price/Fruit/Oranges" "Price/Fruit/#" and this access remains even with the addition of the new topic object and the profiles associated with it.

Grant access to USER2 to subscribe to topic "Price/Fruit/Apples" by granting the user access to the `hlq.SUBSCRIBE.APPLE` profile. Do this, using the following RACF® commands:

```
RDEFINE MXTOPIC hlq.SUBSCRIBE.APPLE UACC(NONE)
PERMIT hlq.SUBSCRIBE.FRUIT APPLE(MXTOPIC) ID(USER2) ACCESS(ALTER)
```

b. Other platforms.

In the previous task USER1 was granted access to subscribe to topic "Price/Fruit/Apples" by granting the user subscribe access to the `FRUIT` profile.

This single profile also granted USER1 access to subscribe to "Price/Fruit/Oranges" and "Price/Fruit/#", and this access remains even with the addition of the new topic object and the profiles associated with it.

Grant access to USER2 to subscribe to topic "Price/Fruit/Apples" by granting the user subscribe access to the `APPLE` profile. Do this, using the following `setmqaut` command:

```
setmqaut -t topic -n APPLE -p USER2 +sub
```

Results

On z/OS, when USER1 attempts to subscribe to topic "Price/Fruit/Apples" the first security check on the `hlq.SUBSCRIBE.APPLE` profile fails, but on moving up the tree the `hlq.SUBSCRIBE.FRUIT` profile allows USER1 to subscribe, so the subscription succeeds and no return code is sent to the MQSUB call. However, a RACF ICH message is generated for the first check:

```
ICH408I USER(USER1 ) ...
hlq.SUBSCRIBE.APPLE ...
```

When USER2 attempts to subscribe to topic "Price/Fruit/Apples" the result is success because the security check passes on the first profile.

When USER2 attempts to subscribe to topic "Price/Fruit/Oranges" the result is failure with an `MQRC_NOT_AUTHORIZED` message, together with:

- On z/OS, the following messages seen on the console that show the full security path through the topic tree that has been attempted:

```
ICH408I USER(USER2 ) ...
hlq.SUBSCRIBE.FRUIT ...

ICH408I USER(USER2 ) ...
hlq.SUBSCRIBE.SYSTEM.BASE.TOPIC ...
```

- On other platforms, the following authorization event:


```
MQRC_NOT_AUTHORIZED
ReasonQualifier    MQRC_SUB_NOT_AUTHORIZED
UserIdentifier     USER2
AdminTopicNames   FRUIT, SYSTEM.BASE.TOPIC
TopicString       "Price/Fruit/Oranges"
```

The disadvantage of this setup is that, on z/OS, you receive additional ICH messages on the console. You can avoid this if you secure the topic tree in a different manner.


Parent topic: [Example publish/subscribe security setup](#)

Related information

[Profiles for queue security \(z/OS\)](#)

 This build: January 26, 2011 10:56:43

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps19525_

5.1.4. Change access control to avoid additional messages

This topic is the fourth in a list of tasks that tells you how to grant access to subscribe to topics by more than one user and to avoid additional RACF® ICH408I messages on z/OS@.

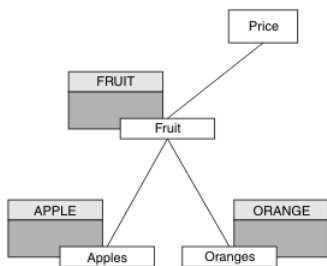
Before you begin

This topic enhances the setup described in [Grant another user access to subscribe to only the topic deeper within the tree](#) so that you avoid additional error messages.

About this task

This topic tells you how to grant access to topics deeper in the tree, and how to remove access to the topic lower down the tree when no user requires it.

Figure 1. Example of granting access control to avoid additional messages.



Define a new topic object as follows:

Procedure

1. Issue the MQSC command `DEF TOPIC(ORANGE) TOPICSTR('Price/Fruit/Oranges')`.

2. Grant access as follows:

a. z/OS.

Define a new profile and add access to that profile, and the existing profiles. Do this, using the following RACF commands:

```
RDEFINE MXTOPIC hlq.SUBSCRIBE.ORANGE UACC(NONE)
PERMIT hlq.SUBSCRIBE.ORANGE CLASS(MXTOPIC) ID(USER1) ACCESS(ALTER)
PERMIT hlq.SUBSCRIBE.APPLE CLASS(MXTOPIC) ID(USER1) ACCESS(ALTER)
```

b. Other platforms.

Setup the equivalent access by using the following **setmqaut** commands:

```
setmqaut -t topic -n ORANGE -p USER1 +sub
setmqaut -t topic -n APPLE -p USER1 +sub
```

Results

On z/OS, when USER1 attempts to subscribe to topic "Price/Fruit/Apples" the first security check on the hlq.SUBSCRIBE.APPLE profile succeeds.

Similarly, when USER2 attempts to subscribe to topic "Price/Fruit/Apples" the result is success because the security check passes on the first profile.

When USER2 attempts to subscribe to topic "Price/Fruit/Oranges" the result is failure with an MQRC_NOT_AUTHORIZED message, together with:

- On z/OS, the following messages seen on the console that show the full security path through the topic tree that has been attempted:

```
ICH408I USER(USER2 ) ...
hlq.SUBSCRIBE.ORANGE ...

ICH408I USER(USER2 ) ...
hlq.SUBSCRIBE.FRUIT ...

ICH408I USER(USER2 ) ...
hlq.SUBSCRIBE.SYSTEM.BASE.TOPIC ...
```

- On other platforms, the following authorization event:

```
MQRC_NOT_AUTHORIZED
ReasonQualifier MQRC_SUB_NOT_AUTHORIZED
UserIdentifier USER2
AdminTopicNames ORANGE, FRUIT, SYSTEM.BASE.TOPIC
TopicString "Price/Fruit/Oranges"
```

Parent topic: [Example publish/subscribe security setup](#)

Related information

[Profiles for queue security \(z/OS\)](#)

This build: January 26, 2011 10:56:43

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps19530_

5.1.5. Grant access to a user to publish to a topic

This topic is the first one in a list of tasks that tells you how to grant access to publish topics by more than one user.

About this task

This task assumes that no administrative topic objects exist on the right hand side of the topic tree, nor have any profiles been defined for publication. The assumption used is that publishers are using the topic string only.

An application can publish to a topic by providing a topic object, or a topic string, or a combination of both. Whichever way the application selects, the effect is to publish at a certain point in the topic tree. If this point in the topic tree is represented by an administrative topic object, a security profile is checked based on the name of that topic object. For example:

Figure 1. Granting publish access to a topic

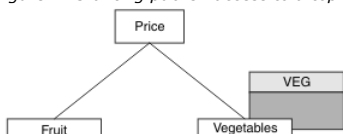


Table 1. Example publish access requirements

Topic	Publish access required	Topic object
Price	No user	None
Price/Vegetables	USER1	VEG

Define a new topic object as follows:

Procedure

1. Issue the MQSC command `DEF TOPIC(VEG) TOPICSTR('Price/Vegetables')`.
2. Grant access as follows:
 - a. z/OS®. Grant access to USER1 to publish to topic "Price/Vegetables" by granting the user access to the hlq.PUBLISH.VEG profile. Do this, using the following RACF® commands:


```
RDEFINE MXTOPIC hlq.PUBLISH.VEG UACC(NONE)
PERMIT hlq.PUBLISH.VEG CLASS(MXTOPIC) ID(USER1) ACCESS(UPDATE)
```
 - b. Other platforms. Grant access to USER1 to publish to topic "Price/Vegetables" by granting the user access to the VEG profile. Do this, using the following `setmqaut` command:


```
setmqaut -t topic -n VEG -p USER1 +pub
```

Results

When USER1 attempts to publish to topic "Price/Vegetables" the result is success; that is, the MQOPEN call succeeds.

When USER2 attempts to publish to topic "Price/Vegetables" the MQOPEN call fails with an `MQRC_NOT_AUTHORIZED` message, together with:

- On z/OS, the following messages seen on the console that show the full security path through the topic tree that has been attempted:

```
ICH408I USER(USER2 ) ...
hlq.PUBLISH.VEG ...

ICH408I USER(USER2 ) ...
hlq.PUBLISH.SYSTEM.BASE.TOPIC ...
```

- On other platforms, the following authorization event:

```
MQRC_NOT_AUTHORIZED
ReasonQualifier    MQRQ_OPEN_NOT_AUTHORIZED
UserIdentifier     USER2
AdminTopicNames   VEG, SYSTEM.BASE.TOPIC
TopicString       "Price/Vegetables"
```

Note that this is an illustration of what you see; not all the fields.

Parent topic: [Example publish/subscribe security setup](#)

Related information
[Profiles for queue security \(z/OS\)](#)

This build: January 26, 2011 10:56:43

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
 This topic's URL:
 ps19535_

5.1.6. Grant access to a user to publish to a topic deeper within the tree

This topic is the second in a list of tasks that tells you how to grant access to publish to topics by more than one user.

Before you begin

This topic uses the setup described in [Grant access to a user to publish to a topic](#).

About this task

If the point in the topic tree where the application publishes is not represented by an administrative topic object, move up the tree until the closest parent administrative topic object is located. The security profile is checked, based on the name of that topic object.

Figure 1. Granting publish access to a topic within a topic tree

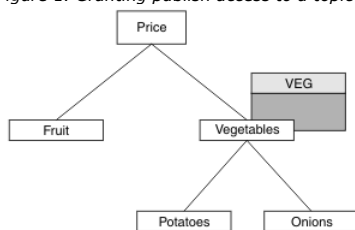


Table 1. Example publish access requirements

Topic	Subscribe access required	Topic object
Price	No user	None
Price/Vegetables	USER1	VEG
Price/Vegetables/Potatoes	USER1	
Price/Vegetables/Onions	USER1	

In the previous task USER1 was granted access to publish topic "Price/Vegetables/Potatoes" by granting it access to the hlq.PUBLISH.VEG profile on z/OS® or publish access to the VEG profile on other platforms. This single profile also grants USER1 access to publish at ▶"Price/Vegetables/Onions".◀

When USER1 attempts to publish at topic "Price/Vegetables/Potatoes" the result is success; that is the MQOPEN call succeeds.

When USER2 attempts to subscribe to topic "Price/Vegetables/Potatoes" the result is failure; that is, the MQOPEN call fails with an `MQRC_NOT_AUTHORIZED` message, together with:

- On z/OS, the following messages seen on the console that show the full security path through the topic tree that has been attempted:

```
ICH408I USER(USER2 ) ...
hlq.PUBLISH.VEG ...

ICH408I USER(USER2 ) ...
```



```
hlq.PUBLISH.SYSTEM.BASE.TOPIC ...
```

- On other platforms, the following authorization event:

```
MQRC_NOT_AUTHORIZED
ReasonQualifier    MQRO_OPEN_NOT_AUTHORIZED
UserIdentifier     USER2
AdminTopicNames   VEG, SYSTEM.BASE.TOPIC
TopicString       "Price/Vegetables/Potatoes"
```

Note the following:

- The messages you receive on z/OS are identical to those received in the previous task as the same topic objects and profiles are controlling the access.
- The event message you receive on other platforms is similar to the one received in the previous task, but the actual topic string is different.

Parent topic: [Example publish/subscribe security setup](#)

Related information

[Profiles for queue security \(z/OS\)](#)

This build: January 26, 2011 10:56:44

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
 This topic's URL:
 ps19540_

5.1.7. Grant access for publish and subscribe

This topic is the last in a list of tasks that tells you how to grant access to publish and subscribe to topics by more than one user.

Before you begin

This topic uses the setup described in [Grant access to a user to publish to a topic deeper within the tree.](#)

About this task

In a previous task USER1 was given access to subscribe to the topic "Price/Fruit". This topic tells you how to grant access to that user to publish to that topic.

Figure 1. Granting access for publishing and subscribing

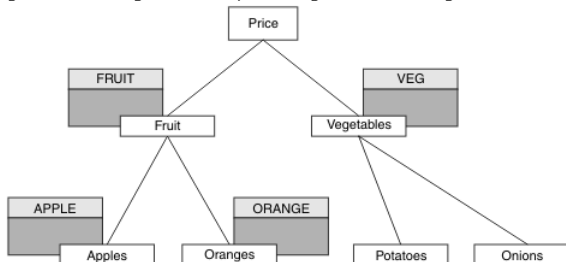


Table 1. Example publishing and subscribing access requirements

Topic	Subscribe access required	Publish access required	Topic object
Price	No user	No user	None
Price/Fruit	USER1	USER1	FRUIT
Price/Fruit/Apples	USER1 and USER2		APPLE
Price/Fruit/Oranges	USER1		ORANGE

Procedure

Grant access as follows:

- z/OS@.

In an earlier task USER1 was granted access to subscribe to topic "Price/Fruit" by granting the user access to the hlq.SUBSCRIBE.FRUIT profile. In order to publish to the "Price/Fruit" topic, grant access to USER1 to the hlq.PUBLISH.FRUIT profile. Do this, using the following RACF@ commands:

```
RDEFINE MXTOPIC hlq.PUBLISH.FRUIT UACC(NONE)
PERMIT hlq.PUBLISH.FRUIT CLASS(MXTOPIC) ID(USER1) ACCESS(ALTER)
```

- Other platforms.

Grant access to USER1 to publish to topic "Price/Fruit" by granting the user publish access to the FRUIT profile. Do this, using the following setmqaut command:

```
setmqaut -t topic -n FRUIT -p USER1 +pub
```

Results

On z/OS, when USER1 attempts to publish to topic "Price/Fruit" the security check on the MQOPEN call passes.

When USER2 attempts to publish at topic "Price/Fruit" the result is failure with an MQRC_NOT_AUTHORIZED message, together with:

- On z/OS, the following messages seen on the console that show the full security path through the topic tree that has been attempted:

```
ICH408I USER(USER2 ) ...
hlq.PUBLISH.FRUIT ...

ICH408I USER(USER2 ) ...
hlq.PUBLISH.SYSTEM.BASE.TOPIC ...
```

- On other platforms, the following authorization event:

```
MQRC_NOT_AUTHORIZED
ReasonQualifier    MQRO_OPEN_NOT_AUTHORIZED
UserIdentifier     USER2
AdminTopicNames   FRUIT, SYSTEM.BASE.TOPIC
```

TopicString

"Price/Fruit"

Following the complete set of these tasks, gives USER1 and USER2 the following access authorities for publish and subscribe to the topics listed:

Table 2. Complete list of access authorities resulting from security examples

Topic	Subscribe access required	Publish access required	Topic object
Price	No user	No user	None
Price/Fruit	USER1	USER1	FRUIT
Price/Fruit/Apples	USER1 and USER2		APPLE
Price/Fruit/Oranges	USER1		ORANGE
Price/Vegetables		USER1	VEG
Price/Vegetables/Potatoes			
Price/Vegetables/Onions			

Where you have different requirements for security access at different levels within the topic tree, careful planning ensures that you do not receive extraneous security warnings on the z/OS console log. Setting up security at the correct level within the tree avoids misleading security messages.

Parent topic: [Example publish/subscribe security setup](#)

Related information

[Profiles for queue security \(z/OS\)](#)

This build: January 26, 2011 10:56:44

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps19545_

5.2. Subscription security

MQSO_ALTERNATE_USER_AUTHORITY

The AlternateUserId field contains a user identifier to use to validate this MQSUB call. The call can succeed only if this AlternateUserId is authorized to subscribe to the topic with the specified access options, regardless of whether the user identifier under which the application is running is authorized to do so.

MQSO_SET_IDENTITY_CONTEXT

The subscription is to use the accounting token and application identity data supplied in the PubAccountingToken and PubAppIdentityData fields.

If this option is specified, the same authorization check is carried out as if the destination queue was accessed using an MQOPEN call with MQOO_SET_IDENTITY_CONTEXT, except in the case where the MQSO_MANAGED option is also used in which case there is no authorization check on the destination queue.

If this option is not specified, the publications sent to this subscriber will have default context information associated with them as follows:

Table 1. Default publication context information

Field in MQMD	Value used
UserIdentifier	The user id associated with the subscription (see SUBUSER field on DISPLAY SBSTATUS) at the time the publication is made.
AccountingToken	Determined from the environment if possible; set to MQACT_NONE otherwise.
AppIdentityData	Set to blanks.

This option is only valid with MQSO_CREATE and MQSO_ALTER. If used with MQSO_RESUME, the PubAccountingToken and PubAppIdentityData fields are ignored, so this option has no effect.

If a subscription is altered without using this option where previously the subscription had supplied identity context information, default context information will be generated for the altered subscription.

If a subscription allowing different user ids to use it with option MQSO_ANY_USERID, is resumed by a different user ID, default identity context will be generated for the new user ID now owning the subscription and any subsequent publications will be delivered containing the new identity context.

AlternateSecurityId

This is a security identifier that is passed with the AlternateUserId to the authorization service to allow appropriate authorization checks to be performed. AlternateSecurityId is used only if MQSO_ALTERNATE_USER_AUTHORITY is specified, and the AlternateUserId field is not entirely blank up to the first null character or the end of the field.

[MQSO_ANY_USERID subscription option](#)

Parent topic: [Publish/subscribe security](#)

This build: January 26, 2011 10:56:57

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps22860_

5.2.1. MQSO_ANY_USERID subscription option

When MQSO_ANY_USERID is specified, the identity of the subscriber is not restricted to a single user ID. This allows any user to alter or resume the subscription when they have suitable authority. Only a single user may have the subscription at any one time. An attempt to resume use of a subscription currently in use by another application will cause the call to fail with MQRC_SUBSCRIPTION_IN_USE.

To add this option to an existing subscription the MQSUB call (using MQSO_ALTER) must come from the same user ID as the original subscription.

If an MQSUB call refers to an existing subscription with MQSO_ANY_USERID set, and the user ID differs from the original subscription, the call succeeds only

if the new user ID has authority to subscribe to the topic. After successful completion, future publications to this subscriber are put to the subscriber's queue with the new user ID set in the publication.

MQSO_FIXED_USERID


When MQSO_FIXED_USERID is specified, the subscription can only be altered or resumed by a single owning user ID. This user ID is the last user ID to alter the subscription that set this option, thereby removing the MQSO_ANY_USERID option, or if no alters have taken place, it is the user ID that created the subscription.

If an MQSUB verb refers to an existing subscription with MQSO_ANY_USERID set and alters the subscription (using MQSO_ALTER) to use option MQSO_FIXED_USERID, the user ID of the subscription is now fixed at this new user ID. The call succeeds only if the new user ID has authority to subscribe to the topic.

If a user ID other than the one recorded as owning a subscription tries to resume or alter an MQSO_FIXED_USERID subscription, the call will fail with MQRCDIDENTITY_MISMATCH. The owning user ID of a subscription can be viewed using the DISPLAY SBSTATUS command.

If neither MQSO_ANY_USERID or MQSO_FIXED_USERID is specified, the default is MQSO_FIXED_USERID.

Parent topic: [Subscription security](#)

 This build: January 26, 2011 10:56:57

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps22840_

6. Queued publish/subscribe compatibility

WebSphere® MQ version 6 publish/subscribe and WebSphere Message and Event Broker version 6 publish/subscribe coexist and interoperate with WebSphere MQ version 7 publish/subscribe, with some restrictions. WebSphere MQ version 6 publish/subscribe is deprecated in WebSphere MQ version 7.

WebSphere® MQ version 7 provides publish/subscribe function that is integrated into the queue manager. Applications use new Message Queue Interface (MQI) verbs to define topics and subscriptions, and to publish and subscribe. There are also new administrative commands to define topics and subscriptions, and to organize publish/subscribe into clusters and hierarchies.

Earlier versions of publish/subscribe were not integrated into the queue manager, but were implemented by separate publish/subscribe brokers. Integrated publish/subscribe is unlike earlier versions of WebSphere MQ. Integrated publish/subscribe extends the MQI verbs to perform publish/subscribe. In earlier versions of WebSphere MQ you placed messages containing MQRPH1, MQRPH2, or PCF commands on special queues to communicate with the publish/subscribe broker. The earlier version of publish/subscribe is known as *queued* publish/subscribe.

Documentation of version 6 publish/subscribe is available with WebSphere MQ version 6, or free to download as an information center or a PDF file from [WebSphere MQ version 6 Publish/Subscribe User's Guide](#).

You can continue to run queued publish/subscribe alongside integrated publish/subscribe.

Interoperability with queued publish/subscribe

You can publish and subscribe on one system to topics defined on a different system. For example, you can write a new WebSphere MQ version 7 publish/subscribe application that uses topics defined using the WebSphere version 6 queued publish/subscribe system. See [Interoperation with queued publish/subscribe](#).

Coexistence with queued publish/subscribe

You can run new WebSphere MQ version 7 publish/subscribe applications alongside existing queued publish/subscribe programs on a version 7 queue manager. See [Coexistence with queued publish/subscribe](#).

Coexistence with WebSphere Message Broker publish/subscribe

If you use the publish/subscribe broker that is part of WebSphere Message Broker version 6.1 to run your queued publish/subscribe applications, or the publish/subscribe broker that is part of earlier members of the WebSphere Message and Event broker family, you can continue to do so if you choose to run the broker on WebSphere MQ version 7. Queued publish/subscribe programs running on WebSphere Message and Event Broker can coexist with integrated publish/subscribe programs running on the same WebSphere MQ version 7 queue manager. See [Coexistence with queued publish/subscribe](#).

Queue manager migration

If you have upgraded your queue manager from version 6 to version 7, you must migrate the version 6 publish/subscribe broker to version 7 to continue to run your queued publish/subscribe applications on the version 7 queue manager. This is described in [Strmqbrk \(Migrate WebSphere MQ version 6.0 broker to version 7.0\)](#).

Changed commands

The commands to manage the integrated publish/subscribe broker in WebSphere version 7 have changed. The new commands are described in [Controlling queued publish/subscribe](#).

JMS

JMS publish/subscribe applications, by default, use integrated publish/subscribe in WebSphere MQ version 7. The default is called normal mode, and is selected by setting the JMS property, PROVIDERVERSION=7. The selection of transport for JMS publish/subscribe is described in [Rules for selecting the WebSphere MQ messaging provider mode](#).

Queued publish/subscribe systems

The queued publish/subscribe implementations that work with version 7 publish/subscribe are listed below. MA08 is no longer available.

SupportPac MA0C

MA0C was a fully supported SupportPac. It added publish/subscribe messaging to WebSphere MQ. It is known as *queued* publish/subscribe, or *command-based* publish/subscribe because it uses MQRPH1 and PCF commands to register publications and subscriptions. Queued publish/subscribe performs these and other publish/subscribe functions by implementing a WebSphere MQ publish/subscribe broker. MA0C was integrated into WebSphere MQ 5.3 in CSD08, and was superseded by WebSphere MQ version 6.0.

SupportPac MA88

MA88 provides JMS support for WebSphere MQ, including its publish/subscribe messaging programming interface. MA0C is one option to implement the publish/subscribe interface. You could also use the earlier versions of WebSphere Message or Event Broker as the publish/subscribe engine. MA88 was superseded by WebSphere MQ 5.3.

WebSphere MQ version 6 (queued) publish/subscribe

WebSphere MQ version 6 superseded MA0C, and supported the JMS and AMI publish/subscribe interfaces in addition to the MQRPH1 and PCF command based programming interfaces.

SupportPac MS0Q

MS0Q extended the IBM® WebSphere MQ version 6 Explorer to provide a topic-based view of the WebSphere MQ V6 publish/subscribe broker. MS0Q has been superseded by the WebSphere MQ version 7 Explorer, which browses topics defined in both version 6 and 7.

WebSphere Message Broker version 6 publish/subscribe

WebSphere Message Broker version 6, and previous and related products, include a variety message transports that provide publish/subscribe support. The message broker enterprise messaging transport includes support for the MA0C and WebSphere version 6 queued publish/subscribe.

Migration

Because queued publish/subscribe coexists and interoperates with the integrated publish/subscribe in WebSphere MQ version 7, you can take a step by step approach to migration. See [Publish/subscribe command messages migration](#) for details. You might take the following steps:

1. Install a version 7 queue manager, enable queued mode publish/subscribe, add the new queue manager to your existing version 6 broker hierarchy, and run your existing queued publish/subscribe programs on the version 7 queue manager.
2. Upgrade your existing version 6 queue managers to version 7, use the **strmqbrk** command to migrate the broker to version 7, and run your existing queued publish/subscribe applications.
3. Write new publish/subscribe programs or migrate existing programs, using the integrated publish/subscribe MQI interface. Guidance on migrating version 6 applications is provided in [Publish/subscribe command messages migration](#). Guidance on writing new publish/subscribe applications is provided in [Writing publish/subscribe applications](#).
4. Migrate broker hierarchies to clusters, to remove dependence on running the queued publish/subscribe broker, and to improve performance by connecting queue managers directly, rather than through an indirect hierarchical network of brokers. Topology migration is described in [WebSphere MQ publish/subscribe topology migration](#).

Streams and topics

WebSphere MQ Version 6 publication streams are mapped to topics by WebSphere MQ Version 7. Default mapping is performed when a version 6 broker is migrated to version 7. You can add and tailor mappings for different configurations.

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 in WebSphere MQ V7.0.1.

Coexistence with queued publish/subscribe

Coexistence is controlled with a new queue manager attribute, PSMODE. Using this attribute you can continue to run existing publish/subscribe applications without modification, and at the same time run new publish/subscribe programs that use the WebSphere MQ Version 7 publish/subscribe interface.

Interoperation with queued publish/subscribe

The WebSphere MQ publish/subscribe brokers exchange publications and subscriptions when they are connected together. There are some differences in the way the queued and integrated publish/subscribe interfaces and brokers work, and what can be connected together. You must take these differences into account when writing applications and administering publish/subscribe brokers that interoperate.

Controlling queued publish/subscribe

Start, stop and display the status of queued publish/subscribe. Add and remove streams, and add and delete queue managers from a broker hierarchy.

MQRFH2 header

In WebSphere Message Broker, the MQRFH2 header was used to pass messages to and from a message broker.

Command messages

This section introduces the RFH2 command messages that an application can send to the queue manager.

Parent topic: [Publish/Subscribe User's Guide](#)

This build: January 26, 2011 10:57:10

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps29000_

6.1. Streams and topics

WebSphere MQ Version 6 publication streams are mapped to topics by WebSphere MQ Version 7. Default mapping is performed when a version 6 broker is migrated to version 7. You can add and tailor mappings for different configurations.

WebSphere MQ Version 6 publish/subscribe has the concept of a publication stream that does not exist in the WebSphere MQ Version 7 publish/subscribe model. In version 6, streams provide a way of separating the flow of information for different topics. 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).

The default stream `SYSTEM.BROKER.DEFAULT.STREAM` is set up automatically for all the brokers 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 brokers, including version 7 brokers running 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.

If you define a topic on WebSphere MQ Version 7, the topic is available to both version 6 and version 7 publishers and subscribers, with no special configuration. For example, suppose a topic is defined, with the topic string `Soccer/Results`. To receive soccer results, a version 6 application subscribes to `Soccer/Results` on `SYSTEM.BROKER.DEFAULT.STREAM`. A version 7 publisher publishes to `Soccer/Results` by opening the topic object and making MQPUT calls to it.

If the version 6 and 7 brokers are on different queue managers, then once 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. So if the topic `Soccer/Results` is registered by a version 6 queued publish/subscribe application, then a version 7 application can subscribe to it using MQSUB.

Named streams

The version 6 solution designer might have decided to place all sports publications into a named stream called `Sport`. In version 6 a stream is often replicated automatically onto other brokers using the model queue, `SYSTEM.BROKER.MODEL.STREAM`. However, for the stream to be available to a version 7 broker running with queued publish/subscribe enabled, the stream needs to be added manually.

If you have just upgraded a queue manager from version 6 to version 7, running the command **strmqbrk** migrates version 6 publish/subscribe broker

resources to version 7, and maps the streams that have been defined to topics. The stream `Sport` is mapped to the topic `Sport`.

Version 6 applications subscribing to `Soccer/Results` on stream `Sport` work without change. New version 7 applications subscribing to the topic `Sport` using `MQSUB`, and supplying the topic string `Soccer/Results` receive the same publications too. When the topic `Sport` is created by `strmqbrk`, it is defined 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 on a version 7 queue manager. The task of adding a stream to a version 7 queue manager 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 version 7 publish/subscribe applications. You need to add new streams to version 7 to flow publications between the new streams and the topics on version 7.
2. You continue to develop your version 6 based publish/subscribe applications that are running on version 7 queue managers, rather than migrate the applications to the version 7 publish/subscribe MQI interface. You need to add new streams to the version 7 queue managers.
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.

Mapping between streams and topics

A version 6 stream is mimicked in version 7 by creating a special queue for publications, giving it the same name as the version 6 stream. Sometimes the queue is called the stream queue, because that is how it appears to version 6 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 in version 7. The purpose of the topic string is to give the topic a unique place in version 7 topic space. Typically the stream name serves that purpose perfectly. Sometimes, a stream name and the version 7 topic space overlap, in which case as long as you keep the topic strings unique, you can choose another topic string for the stream topic.

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.

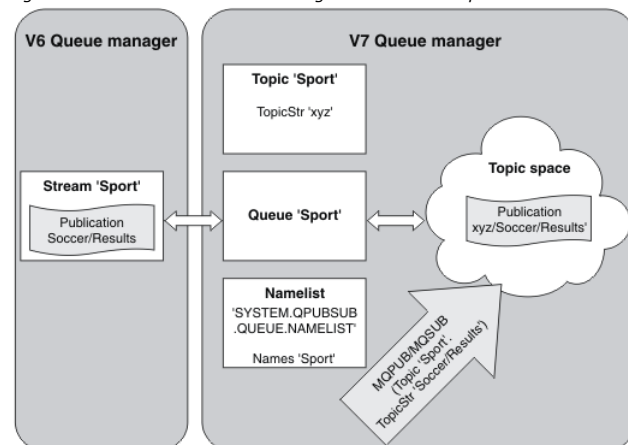
The remapping of different streams onto different topics relies on the prefixes used for the topic strings being unique, to separate one set of topics completely from another. You must define a universal topic naming convention that is rigidly adhered to for the mapping to work. In version 6, if topic strings collided you could use streams to separate the topic spaces. In version 7, 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 is most important if any of the subscriptions originate from other brokers in the broker hierarchy.

Example

In [Figure 1](#), topic `'Sport'` has the topic string `'xyz'` resulting in publications originating 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 1. Version 6 streams coexisting with version 7 topics



Parent topic: [Queued publish/subscribe compatibility](#)

This build: January 26, 2011 10:57:11

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps29050_

6.2. 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 in WebSphere MQ V7.0.1.

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 V7, 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 6 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.

Parent topic: [Queueued publish/subscribe compatibility](#)

Related tasks


[Adding a subscription point](#)
[Migrating publish/subscribe configuration data from WebSphere Event Broker V6](#)

Related reference

[Publish message](#)
[Register Subscriber message](#)

Related information

[Subscription points](#)
[Publication node](#)

 This build: January 26, 2011 10:57:11

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps29052_

6.3. Coexistence with queued publish/subscribe

Coexistence is controlled with a new queue manager attribute, PSMODE. Using this attribute you can continue to run existing publish/subscribe applications without modification, and at the same time run new publish/subscribe programs that use the WebSphere® MQ Version 7 publish/subscribe interface.

Programs written to the WebSphere MQ Version 7 publish/subscribe interface run only on the Version 7 release of WebSphere MQ, and above.

Programs written to the version 6 queued publish/subscribe interfaces run either on version 7 with the queue manager attribute PSMODE set to ENABLED, or on the WebSphere Message or Event Broker Version 6.1 publish/subscribe broker with the queue manager attribute PSMODE set to COMPAT.

Note: If the queue manager is providing enterprise messaging support for WebSphere Message Broker V6 or WebSphere Event Broker V6, then queued publish/subscribe programs must use the publish/subscribe broker provided by those products. New publish/subscribe programs can use version 7 integrated publish/subscribe on the same queue manager. This has implications for interoperability between a version 6 and version 7 publish/subscribe program running on the same queue manager as a message broker. See [Interoperation with queued publish/subscribe](#), for more details.

The PSMODE queue manager attribute controls what kind of publish/subscribe programs run on a version 7 queue manager.

PSMODE

Controls whether the publish/subscribe engine and the queued publish/subscribe interface are running, and therefore controls whether applications can publish or subscribe by using the application programming interface and the queues that are monitored by the queued publish/subscribe interface.

COMPAT

The publish/subscribe engine is running. It is therefore possible to publish or subscribe by using the application programming interface.

The queued publish/subscribe interface is not running. Any publish/subscribe messages put to the queues that are monitored by the queued publish/subscribe interface will not be acted upon.

Use this setting for compatibility with WebSphere Message Broker V6 or earlier versions that use this queue manager, because WebSphere Message Broker needs to read the same queues from which the queued publish/subscribe interface would normally read.

DISABLED


The publish/subscribe engine and the queued publish/subscribe interface are not running. It is therefore not possible to publish or subscribe by using the application programming interface. Any publish/subscribe messages put to the queues that are monitored by the queued publish/subscribe interface will not be acted upon.

ENABLED


The publish/subscribe engine and the queued publish/subscribe interface are running. It is therefore possible to publish or subscribe by using the application programming interface and the queues that are being monitored by the queued publish/subscribe interface. This is the queue manager's initial default value.

Note: Changing the PSMODE attribute can change the PSMODE status. Use [DISPLAY PUBSUB](#), or on i5/OS® DSPMQM, to determine the current state of the publish/subscribe engine and the queued publish/subscribe interface.

Parent topic: [»Queued publish/subscribe compatibility«](#)

 This build: January 26, 2011 10:57:10

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps29010_

6.4. Interoperation with queued publish/subscribe

The WebSphere® MQ publish/subscribe brokers exchange publications and subscriptions when they are connected together. There are some differences in the way the queued and integrated publish/subscribe interfaces and brokers work, and what can be connected together. You must take these differences into account when writing applications and administering publish/subscribe brokers that interoperate.


An example of the interoperation of publish/subscribe is a results service. The subscriber, using MQSUB and running on a WebSphere MQ version 7 queue manager, *subscribes* to the `soccer/scores` topic in the results service that is defined on a WebSphere MQ Version 6 queue manager. The soccer results service publisher is an existing application, written to the WebSphere MQ version 6 publish/subscribe interface. The publisher is running on WebSphere MQ Version 6 and *publishes* to the `soccer/scores` topic. WebSphere MQ Version 6 sends the publication to the subscriber's queue on WebSphere MQ Version 7. Two different kinds of publish/subscribe broker are involved. They share the same topic space, and publications and subscriptions flow between them.

Another example would be to migrate the Version 6 queue manager to version 7. The publisher application continues to run unchanged in queued mode on the migrated queue manager. It interoperates with the version 7 subscriber that runs as a WebSphere MQ Version 7 publish/subscribe application.


»Heterogeneous broker topologies«

Version 6 WebSphere MQ publish/subscribe brokers are organized into *broker hierarchies*. WebSphere Message Brokers are organized into *collectives*. In WebSphere MQ Version 7, you can connect queue managers into *publish/subscribe clusters* or *broker hierarchies*, or a combination of the two.

Parent topic: [»Queued publish/subscribe compatibility«](#)

 This build: January 26, 2011 10:57:11

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps29020_

6.4.1. Heterogeneous broker topologies

Version 6 WebSphere® MQ publish/subscribe brokers are organized into *broker hierarchies*. WebSphere Message Brokers are organized into *collectives*. In WebSphere MQ Version 7, you can connect queue managers into *publish/subscribe clusters* or *broker hierarchies*, or a combination of the two.

The WebSphere MQ Version 6 publish/subscribe broker and the WebSphere MQ Version 7 publish/subscribe interoperate, using broker hierarchies.

WebSphere MQ Version 7 clusters and WebSphere MQ Version 7 hierarchies interoperate. Consult [Combining topic spaces](#).

When you upgrade WebSphere MQ Version 6 to Version 7, you must migrate the Version 6 publish/subscribe broker to use queued publish/subscribe in Version 7. Use the command `strmqbrk` to migrate publish/subscribe from Version 6 to Version 7

Publish/subscribe does not interoperate between WebSphere Message or Event Broker Version 6, and WebSphere MQ Version 7 using publish/subscribe

clusters, collectives, or broker hierarchies. Publish/subscribe connections cannot be established, and hence subscriptions are not propagated, between the brokers and WebSphere MQ Version 7. Applications can exchange messages between WebSphere MQ applications and the brokers using queues.

You can run both WebSphere MQ Version 7 integrated publish/subscribe and WebSphere Message Broker Version 6 publish/subscribe on the same queue manager. You must set the WebSphere MQ Version 7 **PSMODE** queue manager attribute to **COMPAT**. In **COMPAT** mode, queued publish/subscribe applications are serviced by the publish/subscribe broker provided by WebSphere Message or Event Broker Version 6. One implication of running in **COMPAT** mode is that Version 6 publish/subscribe applications do not interoperate with Version 7 applications, whereas in **ENABLED** mode they do.

You can migrate WebSphere Event Broker Version 6.0 to WebSphere MQ Version 7 using the **migmbbrk** command. You must migrate all the Event Brokers in a collective to a WebSphere MQ Version 7 cluster at the same time. Any Event Brokers that remain are not able to connect to the migrated brokers.

Parent topic: [Interoperation with queued publish/subscribe](#)

This build: January 26, 2011 10:57:12

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps29070_

6.5. Controlling queued publish/subscribe

Start, stop and display the status of queued publish/subscribe. Add and remove streams, and add and delete queue managers from a broker hierarchy.

Setting queued publish/subscribe message attributes

The behavior of some publish/subscribe message attributes is controlled by queue manager attributes.

Starting queued publish/subscribe

The task of starting the publish/subscribe broker in WebSphere MQ Version 7 has changed from running the **strmqbrk** to enabling the (deprecated) queued publish/subscribe interface.

Stopping queued publish/subscribe

The task of stopping the publish/subscribe broker in WebSphere MQ Version 7 has changed from running the **endmqbrk** command to disabling the (deprecated) queued publish/subscribe interface.

Adding a stream

You can add streams manually to WebSphere MQ Version 7 queue managers so that they coexist with streams migrated from version 6 queue managers.

Deleting a stream

You can delete a stream from a WebSphere MQ Version 7.0, or later, queue manager.

Adding a subscription point

Add a subscription point that was not migrated from WebSphere MQ Event Broker by **migmbbrk**. Extend an existing queued publish/subscribe application that you have migrated from WebSphere MQ Event Broker with a new subscription point.

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.

Disconnect a queue manager from a broker hierarchy

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

Parent topic: [Queued publish/subscribe compatibility](#)

This build: January 26, 2011 10:56:24

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps11790_

6.5.1. Setting queued publish/subscribe message attributes

The behavior of some publish/subscribe message attributes is controlled by queue manager attributes.

Before you begin

You must use **strmqbrk** to migrate Version 6 publish/subscribe broker state to version 7 if you are working with an upgraded queue manager.

About this task

You can set the following publish/subscribe attributes: for details see, [Queue manager parameters](#).

Table 1. Publish/subscribe attributes

Attribute	Description
PSRTYCNT	Command message retry count
PSNPMSG	Discard undeliverable command input message
PSNPRES	Behavior following undeliverable command response message
PSSYNCPT	Process command messages under syncpoint

Procedure

Use WebSphere MQ Explorer, programmable commands, or the **runmqsc** command to alter the queue manager attributes that control the behavior of publish/subscribe.

Example

```
ALTER QMGR PSNPRES(SAFE)
```

Parent topic: [Controlling queued publish/subscribe](#)

This build: January 26, 2011 10:56:24

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps11801_

6.5.2. Starting queued publish/subscribe

►The task of starting the publish/subscribe broker in WebSphere MQ Version 7 has changed from running the **strmqbrk** to enabling the (deprecated) queued publish/subscribe interface.◀

►

Before you begin

Read the description of [PSMODE](#) to understand the three modes of publish/subscribe:

- COMPAT
- DISABLED
- ENABLED

You must use **strmqbrk** to migrate Version 6 publish/subscribe broker state to version 7 if you are working with an upgraded queue manager.

About this task

Set the QMGR *PSMODE* attribute to start either the queued publish/subscribe interface (also known as the broker), or the publish/subscribe engine (also known as Version 7 publish/subscribe) or both. To start queued publish/subscribe you need to set *PSMODE* to *ENABLED*. The default is *ENABLED*.

Procedure

►Use WebSphere MQ Explorer or the **runmqsc** command to enable the queued publish/subscribe interface if the interface is not already enabled.◀


Example

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

What to do next

WebSphere MQ processes queued publish/subscribe commands and publish/subscribe Message Queue Interface (MQI) calls.

Parent topic: ►[Controlling queued publish/subscribe](#)◀

 This build: January 26, 2011 10:56:24

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps11800_

6.5.3. Stopping queued publish/subscribe

►The task of stopping the publish/subscribe broker in WebSphere MQ Version 7 has changed from running the **endmqbrk** command to disabling the (deprecated) queued publish/subscribe interface.◀

►

Before you begin

Queued publish/subscribe is deprecated.

Read the description of [PSMODE](#) to understand the three modes of publish/subscribe:

- COMPAT
- DISABLED
- ENABLED

◀

About this task

Set the QMGR *PSMODE* attribute to stop either the queued publish/subscribe interface (also known as the broker), or the publish/subscribe engine (also known as Version 7 publish/subscribe) or both. To stop queued publish/subscribe you need to set *PSMODE* to *COMPAT*. To stop the publish/subscribe engine entirely, set *PSMODE* to *DISABLED*.

Procedure

►Use WebSphere MQ Explorer or the **runmqsc** command to disable the queued publish/subscribe interface.◀


Example

```
ALTER QMGR PSMODE(COMPAT)
```

What to do next

WebSphere MQ processes only Version 7 Message Queue Interface (MQI) publish/subscribe calls.

Parent topic: ►[Controlling queued publish/subscribe](#)◀

 This build: January 26, 2011 10:57:11

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps29030_

6.5.4. Adding a stream

You can add streams manually to WebSphere MQ Version 7 queue managers so that they coexist with streams migrated from version 6 queue managers.

Before you begin

Familiarize yourself with the way publish/subscribe streams operate in WebSphere MQ version 7 onwards by reading the topic, [Streams and topics](#).

About this task

Use PCF commands, **runmqsc**, or WebSphere MQ Explorer to do these steps.

Procedure

1. Define a local queue with the same name as the version 6 stream.
2. Define a local topic with the same name as the version 6 stream.
3. Set the `WILDCARD` property of the topic to the value `BLOCK`.
Blocking the subscriptions to `#` or `*` isolates wildcard subscriptions to streams, see [Wildcards and streams](#).
4. Add the name of the queue to the `namelist`, `SYSTEM.QPUBSUB.QUEUE.NAMELIST`
5. Repeat the previous three steps for all queue managers at version 7 or above that are in the Publish/Subscribe hierarchy.

Adding 'Sport'

In the example of sharing the stream 'Sport', version 6 and version 7 queue managers are working in the same publish/subscribe hierarchy. The version 6 queue managers share a stream called 'Sport'. The example shows how to create a queue and a topic on version 7 queue managers called 'Sport', with a topic string 'Sport' that is shared with the version 6 stream 'Sport'.

A version 7 publish application, publishing to topic 'Sport', with topic string 'Soccer/Results', creates the resultant topic string 'Sport/Soccer/Results'. On version 7 queue managers, subscribers to topic 'Sport', with topic string 'Soccer/Results' receive the publication.

On version 6 queue managers, subscribers to stream 'Sport', with topic string 'Soccer/Results' receive the publication.

```
runmqsc QM1
5724-H72 (C) Copyright IBM Corp. 1994, 2008. ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM1.
define qlocal('Sport')
  1 : define qlocal('Sport')
AMQ8006: WebSphere MQ queue created.
define topic('Sport') topicstr('Sport') wildcard(BLOCK)
  2 : define topic('Sport') topicstr('Sport') wildcard(BLOCK)
AMQ8690: WebSphere MQ topic created.
alter namelist(SYSTEM.QPUBSUB.QUEUE.NAMELIST) NAMES('Sport', 'SYSTEM.BROKER.DEFAULT.STREAM', 'SYSTEM.BROKER.ADMIN.STREAM')
  3 : alter namelist(SYSTEM.QPUBSUB.QUEUE.NAMELIST) NAMES('Sport', 'SYSTEM.BROKER.DEFAULT.STREAM', 'SYSTEM.BROKER.ADMIN.STREAM')
AMQ8551: WebSphere MQ namelist changed.
```

Note: You need both to provide the existing names in the `namelist` object, as well as the new names that you are adding, to the **alter namelist** command.

What to do next

Information about the stream is passed to other brokers in the hierarchy.


If a broker is version 6, administer it as a version 6 broker. That is, you have a choice of creating the stream queue manually, or letting the broker create the stream queue dynamically when it is needed. The queue is based on the model queue definition, `SYSTEM.BROKER.MODEL.STREAM`.

If a broker is version 7, you need to configure each version 7 queue manager in the hierarchy manually.

Parent topic: [Controlling queued publish/subscribe](#)

Related concepts

[Wildcard rules](#)

 This build: January 26, 2011 10:57:15

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps29840_



6.5.5. Deleting a stream

You can delete a stream from a WebSphere MQ Version 7.0, or later, queue manager.

Before you begin

The use of queued publish/subscribe is deprecated in WebSphere® MQ Version 7.0.

► Before deleting a stream from a WebSphere MQ Version 7 queue manager you must ensure that there are no remaining subscriptions to the stream and quiesce all applications that use the stream. If publications continue to flow to a deleted stream, it takes a lot of administrative effort to restore the system to a cleanly working state. ◀

About this task

For instructions on deleting the stream from any version 6 queue managers it is connected to, see [Deleting a stream](#).


Procedure

1. Find all the connected brokers that host this stream.
2. Cancel all subscriptions to the stream on all the brokers.
3. Remove the queue (with the same name as the stream) from the `namelist`, `SYSTEM.QPUBSUB.QUEUE.NAMELIST`.
4. Delete or purge all the messages from the queue with the same name as the stream.
5. Delete the queue with the same name as the stream.
6. Delete the associated topic object.


What to do next

1. Repeat steps 3 to 5 on all the other connected version 7, or later, queue managers hosting the stream.
2. Remove the stream from all other connected version 6, or earlier, queue managers.

Parent topic: [▶Controlling queued publish/subscribe◀](#)

 This build: January 26, 2011 10:56:24

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps11870_



6.5.6. Adding a subscription point

Add a subscription point that was not migrated from WebSphere® MQ Event Broker by **migmbbrk**. Extend an existing queued publish/subscribe application that you have migrated from WebSphere MQ Event Broker with a new subscription point.

Before you begin

1. Complete the migration from WebSphere MQ Event Broker V6 to WebSphere MQ V7.0.1.
2. Check that the subscription point is not already defined in `SYSTEM.QPUBSUB.SUBPOINT.NAMELIST`.
3. Check if there is a topic object or a topic string with the same name as the subscription point.

About this task

Existing WebSphere MQ Event Broker applications use subscription points. New WebSphere MQ V7 applications do not use subscription points, but they can interoperate with existing applications that do, using the subscription point migration mechanism.

A subscription point might not have been migrated by **migmbbrk**, if the subscription point was not in use at the time of the migration.

You might want to add a subscription point to existing queued publish/subscribe programs migrated from WebSphere MQ Event Broker.

Subscription points do not work with queued publish/subscribe programs that use `MQRFH1` headers, which have been migrated from WebSphere MQ V6 or earlier.

There is no need to add subscription points to use integrated publish/subscribe applications written for WebSphere MQ V7.

Procedure

1. Add the name of the subscription point to `SYSTEM.QPUBSUB.SUBPOINT.NAMELIST`.
 - On z/OS®, the **NLTYPE** is `NONE`, the default.
 - Repeat the step on every queue manager that is connected in the same publish/subscribe topology.
2. Add a topic object, preferably giving it the name of the subscription point, with a topic string matching the name of the subscription point.
 - If the subscription point is in a cluster, add the topic object as a cluster topic on the cluster topic host.
 - If a topic object already exists with the same topic string as the name of the subscription point, use the existing topic object, as long as you understand the consequences of the subscription point is reusing an existing topic. If the existing topic is part of an existing application, you must resolve the collision between two identically named topics.
 - If a topic object already exists with the same name as the subscription point, but a different topic string, create a new topic with a different name.
3. Set the **Topic** attribute `WILDCARD` to the value `BLOCK`.
Blocking subscriptions to `#` or `*` isolates wildcard subscriptions to subscription points, see [Wildcards and subscription points](#).
4. Set any attributes that you require in the topic object.

Example

The example shows a **runmqsc** command file that adds two subscription points, `USD` and `GBP`.

```
DEFINE TOPIC(USD) TOPICSTR(USD) WILDCARD(BLOCK)
DEFINE TOPIC(GBP) TOPICSTR(GBP) WILDCARD(BLOCK)
ALTER NL(SYSTEM.QPUBSUB.SUBPOINT.NAMELIST) NAMES(SYSTEM.BROKER.DEFAULT.SUBPOINT, USD, GBP)
```

Note:

1. Include the default subscription point in the list of subscription points added using the **ALTER** command. **ALTER** deletes existing names in the namelist.
2. Define the topics before altering the namelist. The queue manager only checks the namelist when the queue manager starts and when the namelist is altered.

Parent topic: [▶Controlling queued publish/subscribe◀](#)


Related concepts

[Wildcard rules](#)


[Subscription points and topics](#)

Related tasks

[Migrating publish/subscribe configuration data from WebSphere Event Broker V6](#)

 This build: January 26, 2011 10:57:12

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps29054_

6.5.7. 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 a WebSphere MQ connection. There are two ways to establish the connection.
 - o Connect the queue managers to a WebSphere MQ cluster.
 - o 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 example, suppose you are connecting to a queue manager called PARENT. Define a queue manager alias for PARENT that resolves to the transmission queue to parent. To place messages for PARENT on the transmission queue `PARENT.XMITQ`, use the following MQSC command to define the queue manager alias.

```
DEFINE QREMOTE (PARENT) RNAME('') RQMNAME(PARENT) XMITQ(PARENT.XMITQ)
```

About this task

In WebSphere® MQ Version 6.0, when the appropriate channels and queues are defined, brokers connect to one another as defined by parameters provided on the **strmqbrk** command.

The **strmqbrk** command works differently in WebSphere MQ Version 7.0 and you can no longer use it to connect children to parents. Instead you use the `ALTER QMGR PARENT (PARENT_NAME) runmqsc` command.

In WebSphere MQ Version 7, distributed publish/subscribe is typically implemented by using queue manager clusters and clustered topic definitions. For interoperability with WebSphere MQ Version 6 and WebSphere Message Broker V6.1 and WebSphere Event Broker V6.1 and earlier, you can also connect version 7 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, 2008. 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 type(All)
  14 : display pubsub type(All)
AMQ8723: Display pub/sub status details.
      QMNAME(QM2) TYPE(LOCAL)
AMQ8723: Display pub/sub status details.
      QMNAME(QM1) TYPE(PARENT)
```


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, 2008. ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM1.
display pubsub type(all)
  1 : display pubsub type(all)
AMQ8723: Display pub/sub status details.
      QMNAME(QM1) TYPE(LOCAL)
AMQ8723: Display pub/sub status details.
      QMNAME(QM2) TYPE(CHILD)
```

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.

Parent topic: [Controlling queued publish/subscribe](#)

 This build: January 26, 2011 10:56:28

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps15810_

6.5.8. 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

In WebSphere® MQ Version 6.0, queue managers were disconnected from one another using the **dltmqbrk** command, and required that all child queue managers were disconnected first. In WebSphere MQ Version 7, the **dltmqbrk** command is used to discard WebSphere MQ Version 6 broker resources after migration to version 7 using the **strmqbrk** command.

You disconnect a version 7 queue manager from a broker hierarchy using the **ALTER QMGR** command. Unlike version 6, you can disconnect version 7 queue managers in any order and 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, 2008. 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.

Parent topic: [Controlling queued publish/subscribe](#)

This build: January 26, 2011 10:56:29

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps15820_

6.6. MQRFH2 header

In WebSphere® Message Broker, the MQRFH2 header was used to pass messages to and from a message broker.

In a message, the MQRFH2 header follows the WebSphere MQ message descriptor (MQMD) and precedes the message body, if present.

Other headers are allowed after the MQRFH2 header, but before the message body.

If you want to use the Message Queuing Interface (MQI) to write application programs that use the same queued interface as Websphere Message Broker for publish/subscribe, you need to understand the structure and content of the MQRFH2 header.

For more details, refer to the following information:

- [MQRFH2 structure](#)
- [Message service folders](#)

►MQRFH2 structure◄

The MQRFH2 header contains information about the structure of a message, and its intended consumers, to enable a queue manager to process the message and deliver or publish the message to those consumers.

►Message service folders◄

A number of folders are defined for use by WebSphere MQ products.

Parent topic: [Queued publish/subscribe compatibility](#)

Multiple MQRFH2 headers

A message can have more than one MQRFH2 header.

For example, if an application forwards a message, including its header, to another application, a second MQRFH2 header precedes the header in the message being forwarded.

- Attributes that describe the body of the message, such as the domain, set, type, and format, or the character set ID and encoding, are taken from the *last* MQRFH2 header, which is immediately in front of the message.
- Anything else, such as the topic for a publish/subscribe message, is taken from the *first* MQRFH2 header.

This build: January 26, 2011 10:57:00

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps22960_

6.6.1. MQRFH2 structure

The MQRFH2 header contains information about the structure of a message, and its intended consumers, to enable a queue manager to process the message and deliver or publish the message to those consumers.

You must put the value 'MQRFH2 ' in the **Format** field of the preceding header (usually the MQMD). The constant `MQFMT_RF_HEADER_2` is defined with this value.

For the C programming language, the constant `MQFMT_RF_HEADER_2_ARRAY` is also defined. This constant has the same value as `MQFMT_RF_HEADER_2`, but it is an array of characters, not a character string.

The character set and encoding of the fields in the MQRFH2 header are as follows:

- Fields other than **NameValueData** are in the character set and encoding defined by the fields **CodedCharSetId** and **Encoding** in the header structure that precedes the MQRFH2 header, or by the same fields in the MQMD structure if the MQRFH2 header is at the start of the application message data. The character set should be one that has single-byte characters for the characters that are valid in queue names.
- **NameValueData** is in the character set defined by the **NameValueCCSID** field. Note that not all Unicode character sets are valid for **NameValueCCSID**; see the description of **NameValueCCSID** for details. Some character sets have a representation that is dependent on the encoding. If **NameValueCCSID** defines one of these character sets, **NameValueData** must be in the same encoding as the other fields in the MQRFH2 header.
- The user data (if any) that follows **NameValueData** can be in any supported character set (single-byte, double-byte, or multi-byte), and in any supported encoding.

The MQRFH2 header contains the following fields:

Field Name	Description	Details
StrucId	Structure identifier	The value must be <code>MQRFH_STRUC_ID</code> , which is the identifier for the rules and formatting header structure. For the C programming language, the constant <code>MQRFH_STRUC_ID_ARRAY</code> is

		also defined; this constant has the same value as <code>MQRFH_STRUCT_ID</code> , but it is an array of characters, not a character string.
Version	Structure version number	The value must be <code>MQRFH_VERSION_2</code> , which is the Version-2 rules and formatting header structure.
Struclength	Total length of MQRFH2 (including NameValueData)	<p>The initial value of this field is <code>MQRFH_STRUCT_LENGTH_FIXED_2</code>, which is the length of the fixed part of the MQRFH2 header structure.</p> <p>This is the length in bytes of the MQRFH2 header structure, including any NameValueLength and NameValueData fields at the end of the structure.</p> <p>There might be more than one pair of these fields at the end of the structure, in the sequence: length1, data1, length2, data2, The length of any user data that follows the last NameValueData field at the end of the structure is not included in Struclength.</p> <p>Note: If Struclength is not a multiple of four, problems might occur with the data conversion of user data in some operating system environments.</p>
Encoding	Numeric encoding of data that follows NameValueData	<p>The initial value of this field is <code>MQENC_NATIVE</code>.</p> <p>This field specifies how numeric values in any data that follows the last NameValueData field are represented. This applies to binary integer data, packed decimal integer data and floating-point data.</p>
CodedCharSetId	Character set identifier of data that follows NameValueData	<p>The initial value of this field is <code>MQCCSI_INHERIT</code>, which means that the character set identifier is the same as that of the current structure.</p> <p>This field identifies the coded character set for any character strings in the data that follows the last NameValueData field.</p>
Format	Format name of data that follows NameValueData	<p>The initial value of this field is <code>MQFMT_NONE</code>.</p> <p>This field specifies the format name of any data that follows the last NameValueData field. The name should be padded with blanks to the length of the field.</p> <p>Note: Do not use a null character to terminate the name before the end of the field; the queue manager does not change to a blank character the null character, or any characters that follow the null character, in the MQRFH2 header.</p> <p>Note: Do not specify a name with leading or embedded blank characters.</p>
Flags	Flags	The initial value of this field is <code>MQRFH_NONE</code> , which means that there are no flags.
NameValueCCSID	Character set identifier of NameValueData	<p>The initial value of this field is 1208, which means that the UTF-8 coded character set is used.</p> <p>This field identifies the coded character set for data in the NameValueData field. This is different from the character set for other character strings in the MQRFH2 header structure, and might be different from the character set for any character data that follows the last NameValueData field.</p> <p>NameValueCCSID must have one of the following values:</p> <p>1200: UCS-2 open-ended</p> <p>1208: UTF-8</p> <p>13488: UCS-2 2.0 subset</p> <p>17584: UCS-2 2.1 subset (includes the euro symbol €)</p> <p>For the UCS-2 character sets, the encoding (byte order) of the NameValueData field must be the same as the encoding of the other fields in the MQRFH2 header structure.</p> <p>Note: Surrogate characters (X'D800' thru X'DFFF') are not supported.</p>
The following two fields are optional, but if present they must occur as a pair. They can be repeated as a pair as many times as required.		
If these fields occur more than once, they must occur in the sequence length1, data1, length2, data2,		
NameValueLength	Length of NameValueData	This field specifies the length, in bytes, of the NameValueData field that follows this field.
NameValueData	This is a variable-length character string containing data that is encoded using an XML-like structure	The length, in bytes, of this string is given by the NameValueLength field that precedes this NameValueData field.

C programming language definition

The following structure is defined in the `cmqc.h` header file that is supplied with WebSphere® MQ. The constants that are used within the **NameValueData** field are defined in the `cmqpsc.h` header file.

```
typedef struct tagMQRFH2 {
    MQCHAR4   StrucId;           /* Structure identifier          */
    MQLONG    Version;          /* Structure version number     */
    MQLONG    Struclength;      /* Total length of MQRFH2 including
                               NameValueData                */
    MQLONG    Encoding;         /* Numeric encoding of data that
                               follows NameValueData        */
    MQLONG    CodedCharSetId;   /* Character set identifier of data
                               that follows NameValueData   */
    MQCHAR8   Format;           /* Format name of data that follows
                               NameValueData                */
};
```

```

                                NameValueData          */
MQLONG  Flags;                /* Flags                */
MQLONG  NameValueCCSID;       /* Character set identifier of NameValueData */
) MQRFH2;

```

Parent topic: [MORFH2 header](#)

This build: January 26, 2011 10:57:01

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps22970_

6.6.2. Message service folders

A number of folders are defined for use by WebSphere® MQ products.

<mcd>

Message content descriptor

<psc>

Publish/subscribe command

<pscr>

Publish/subscribe command response

<usr>

Application (user) defined properties

<jms>

Java Messaging Service

Each folder is contained in a separate **NameValueData** field, each of which is preceded by a **NameValueLength** field.

Independent software vendors can choose other names for their folders. However, you can prefix your chosen folder name with their internet domain name to avoid naming conflicts and problems. For example, a vendor with domain name `ourcompany.com` might name its folders:

```
com.ourcompany.xxx or com.ourcompany.ourData
```

Parent topic: [MORFH2 header](#)

The mcd folder

The <mcd> folder can contain the following elements that describe the structure of the message data in a WebSphere MQ message. They are all character strings, and are case-sensitive.

<Msd>

Message service domain

Valid values are:

mrm

The message is parsed by the MRM domain.

xmlnsc

The message is XML and is parsed by the XMLNSC domain.

xmlns

The message is XML and is parsed by the XMLNS domain.

xml

The message is XML and is parsed by the XML domain.

mime

The message uses the MIME standard and is parsed by the MIME domain.

idoc

The message is a SAP ALE IDoc from the WebSphere MQ Link for R/3, and is parsed by the IDOC domain.

none

The message is treated as an opaque BLOB, and delivered to the recipient without modification.

See the description of the parsers in the WebSphere Message Brokers information center for a description of each domain.

<Set>

The name of the message set containing the definition of the message.

If <Msd> is *mrm* or *idoc*, you must use the <Set> element to supply the name of the message set.

<Type>

The name of the message type, within the specified message set, to which this message corresponds. The format of a simple message type is `{namespace-uri}:name` where `name` is the name of the message.

The format `{namespace-uri}name` (that is, with no colon) is also valid to maintain compatibility with previous versions of the WebSphere Message Brokers product.

If <Msd> is *mrm* or *idoc*, you must use the <Type> element to supply the name of the message definition.

<Fmt>

The name of a physical format within the specified message set.

If <Msd> is *mrm* or *idoc*, you must use the <Fmt> element to supply the name of the physical format in the message set.

If an MQRFH2 header is present in an output message tree, an <mcd> folder is always added and populated. This is so that the MQRFH2 header accurately reflects the message template of the message. The <mcd> folder must not be removed from the MQRFH2 header. If it is deleted - for example, by a Compute node - it is automatically added and populated again on exit from the node.

The psc folder

The <psc> folder is used to convey publish/subscribe command messages to the queue manager.

Only one psc folder is allowed in the **NameValueData** field.

See [Command messages](#) for full details.

The pscr folder

The <pscr> folder is used to contain information from the queue manager, in response to publish/subscribe command messages.

Only one pscr folder is present in a response message.

See [Queue Manager Response message](#) for full details.

The queue manager ignores this folder in messages that it receives from publish/subscribe applications.

The usr folder

The content model of the <usr> folder has the following characteristics.


- Any valid XML name that does not contain a colon can be used as an element name.
- Only simple elements, not groups, are allowed.
- All elements take the default type of string.
- All elements are optional, but must not occur more than once in a folder.
- An MQRFH2 instance can contain, at most, one <usr> folder.

The jms folder

The content model of the <jms> folder contains the following MQRFH2 JMS fields:

- `Dst` - represents the JMSDestination header field.
- `Div` - represents the JMSDeliveryMode header field.
- `Exp` - represents the JMSExpiration header field.
- `Pri` - represents the JMSPriority header field.
- `Tms` - represents the JMSTimestamp header field.
- `Cid` - represents the JMSCorrelationID header field.
- `Rto` - represents the JMSReplyTo header field.

See [JMS messages](#) for more information.

 This build: January 26, 2011 10:57:01

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps22980_

6.7. Command messages

This section introduces the RFH2 command messages that an application can send to the queue manager.

An application that is using RFH2 for publish/subscribe can send the following command messages to the queue manager:

- [Delete Publication message](#)
- [Deregister Subscriber message](#)
- [Publish message](#)
- [Register Subscriber message](#)
- [Request Update message](#)

If you are using the Message Queue Interface (MQI) to write applications that use the publish/subscribe model, you need to understand these messages, the Queue Manager Response message, and the message descriptor (MQMD). Refer to the following information:

- [Queue Manager Response message](#)
- [MQMD settings for publications forwarded by a queue manager](#)
- [MQMD settings in queue manager response messages](#)
- [Reason codes](#)

The commands are contained in a <psc> folder in the **NameValueData** field of the MQRFH2 header.

The message that can be sent by a broker in response to a command message is contained in a <pscr> folder.

Refer to [Message service folders](#) for details about the message service folders.

The descriptions of each command list the properties that can be contained in a folder. Unless otherwise specified, the properties are optional and can occur no more than once.

Names of properties are shown as <Command>.

Values must be in string format, for example: `Publish`.

A string constant representing the value of a property is shown in parentheses, for example: `(MQPSC_PUBLISH)`.

String constants are defined in the header file `cmqpsc.h` which is supplied with the queue manager.

►Delete Publication message◄

The **Delete Publication** command message is sent to a queue manager from a publisher, or from another queue manager, to tell the queue manager to delete any retained publications for the specified topics.

►Deregister Subscriber message◄

The **Deregister Subscriber** command message is sent to a queue manager by a subscriber, or by another application on behalf of a subscriber, to indicate that it no longer wants to receive messages matching the given parameters.

►Publish message◄

The **Publish** command message is sent from a publisher to a queue manager, or from a queue manager to a subscriber, to publish information on a specified topic or topics.

►Register Subscriber message◄

The **Register Subscriber** command message is sent to a queue manager by a subscriber, or by another application on behalf of a subscriber, to indicate that it wants to subscribe to one or more topics at a subscription point. A message content filter can also be specified.

►Request Update message◄

The **Request Update** command message is sent from a subscriber to a queue manager, to request the current retained publications for the specified topic and subscription point that match the given (optional) filter.

►Queue Manager Response message◄

A **Queue Manager Response** message is sent from a queue manager to the `ReplyToQ` of a publisher or a subscriber, to indicate the success or failure of a command message received by the queue manager if the command message descriptor specified that a response is required.

►Reason codes◄

These reason codes might be returned in the `Reason` field of a publish/subscribe response `<pscr>` folder. Constants that can be used to represent these codes in the C or C++ programming languages are also listed.

►MQMD settings in command messages to the queue manager◄

Applications that send command messages to the queue manager use the following settings of fields in the message descriptor (MQMD). Fields that are left as the default value, or can be set to any valid value in the usual way, are not listed here.


►MQMD settings for publications forwarded by a queue manager◄

A queue manager uses these settings of fields in the message descriptor (MQMD) when it sends a publication to a subscriber. All other fields in the MQMD are set to their default values.

►MQMD settings in queue manager response messages◄

A queue manager uses these settings of fields in the message descriptor (MQMD) when sending a reply to a publication message. All other fields in the MQMD are set to their default values.

Parent topic: ►Queued publish/subscribe compatibility◄

 This build: January 26, 2011 10:57:02

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps22990_



6.7.1. Delete Publication message

The **Delete Publication** command message is sent to a queue manager from a publisher, or from another queue manager, to tell the queue manager to delete any retained publications for the specified topics.

This message is sent to a queue monitored by the queue manager's queued publish/subscribe interface.

The input queue should be the queue that the original publication was sent to.

If you have the authority for some, but not all, of the topics that are specified in the **Delete Publication** command message, only those topics are deleted. A **Broker Response** message indicates which topics are not deleted.

Similarly, if a **Publish** command contains more than one topic, a **Delete Publication** command matching some, but not all, of those topics deletes only the publications for the topics that are specified in the **Delete Publication** command.

See [MQMD settings for publications forwarded by a queue manager](#) for details of the message descriptor (MQMD) parameters that are needed when sending a command message to the queue manager.

Parent topic: ►Command messages◄

Properties

<Command> (MQPSC_COMMAND)

The value is `DeletePub(MQPSC_DELETE_PUBLICATION)`.

This property must be specified.

<Topic> (MQPSC_TOPIC)

The value is a string that contains a topic for which retained publications are to be deleted. Wildcard characters can be included in the string to delete publications on more than one topic.

This property must be specified; it can be repeated for as many topics as needed.

<DelOpt> (MQPSC_DELETE_OPTION)

The delete options property can take one of the following values:

Local (MQPSC_LOCAL)

All retained publications for the specified topics are deleted at the local queue manager (that is, the queue manager to which this message is sent),

whether they were published with the `Local` option or not.

Publications at other queue managers are not affected.

None (**MQPSC_NONE**)


All options take their default values. This has the same effect as omitting the `DelOpt` property. If other options are specified at the same time, `None` is ignored.

The default if this property is omitted is that all retained publications for the specified topics are deleted at all queue managers in the network, regardless of whether they were published with the `Local` option.

Example

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

```
<psc>
  <Command>DeletePub</Command>
  <Topic>Sport/Soccer/State/LatestScore/Team1 Team2</Topic>
  <DelOpt>Local</DelOpt>
</psc>
```

 This build: January 26, 2011 10:57:02

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps23000_



6.7.2. Deregister Subscriber message

The **Deregister Subscriber** command message is sent to a queue manager by a subscriber, or by another application on behalf of a subscriber, to indicate that it no longer wants to receive messages matching the given parameters.

This message is sent to `SYSTEM.BROKER.CONTROL.QUEUE`, the queue manager's control queue. The user must have the necessary authority to put a message onto this queue.

See [MQMD settings for publications forwarded by a queue manager](#) for details of the message descriptor (MQMD) parameters that are needed when sending a command message to the queue manager.

An individual subscription can be deregistered by specifying the corresponding topic, subscription point and filter values of the original subscription. If any of the values were not specified (that is, they took the default values) in the original subscription, they should be omitted when the subscription is deregistered.

All subscriptions for a subscriber, or a group of subscribers, can be deregistered by using the `DeregAll` option. For example, if `DeregAll` is specified, together with a subscription point (but no topic or filter), then all subscriptions for the subscriber on the specified subscription point are deregistered, regardless of the topic and filter. Any combination of topic, filter and subscription point is allowed; if all three are specified only one subscription can match, and the `DeregAll` option is ignored.

The message must be sent by the subscriber that registered the subscription; this is confirmed by checking the subscriber's user ID.

Subscriptions can also be deregistered by a system administrator using MQSC or PCF commands. However, the subscriptions registered with a temporary dynamic queue are associated with the queue, not just the queue name. If the queue is deleted, either explicitly, or by the application disconnecting from the queue manager, it is no longer possible to use the **Deregister Subscriber** command to deregister the subscriptions for that queue. The subscriptions can be deregistered using the developer workbench, and they are removed automatically by the queue manager the next time that it matches a publication to the subscription, or the next time the queue manager restarts. Under normal circumstances, applications should deregister their subscriptions before deleting the queue, or disconnecting from the queue manager.

If a subscriber sends a message to deregister a subscription, and receives a response message to say that this was processed successfully, some publications might still reach the subscriber queue if they were being processed by the queue manager at the same time as the subscription was being deregistered. If the messages are not removed from the queue, there might be a buildup of unprocessed messages on the subscriber queue. If the application executes a loop that includes an MQGET call with the appropriate `CorrelId` after sleeping for a while, these messages are cleared off the queue.

Similarly, if the subscriber uses a permanent dynamic queue, and deregisters and closes the queue with the `MQCO_DELETE_PURGE` option on an MQCLOSE call, the queue might not be empty. If any publications from the queue manager are not yet committed when the queue is deleted, an MQRC_Q_NOT_EMPTY return code is issued by the MQCLOSE call. The application can avoid this problem by sleeping and reissuing the MQCLOSE call from time to time.

Parent topic: [Command messages](#)

Properties

<Command> (**MQPSC_COMMAND**)

The value is `DeregSub` (`MQPSC_DEREGISTER_SUBSCRIBER`).

This property must be specified.

<Topic> (**MQPSC_TOPIC**)

The value is a string that contains the topic to be deregistered.

This property can, optionally, be repeated if multiple topics are to be deregistered. It can be omitted if `DeregAll` is specified in `<RegOpt>`.

The topics that are specified can be a subset of those that are registered if the subscriber wants to retain subscriptions for other topics. Wildcard characters are allowed, but a topic string that contains wildcard characters must exactly match the corresponding string that was specified in the **Deregister Subscriber** command message.

<SubPoint> (**MQPSC_SUBSCRIPTION_POINT**)

The value is a string that specifies the subscription point from which the subscription is to be detached.

This property must not be repeated. It can be omitted if a `<Topic>` is specified, or if `DeregAll` is specified in `<RegOpt>`. If you omit this property, the following happens:

- If you do **not** specify `DeregAll`, subscriptions matching the `<Topic>` property (and the `<Filter>` property, if present) are deregistered from the default subscription point.
- If you specify `DeregAll`, all subscriptions (matching the `<Topic>` and `<Filter>` properties if present) are deregistered from all subscription points.

Note that you cannot specify the default subscription point explicitly. Therefore, there is no way of deregistering all subscriptions from this subscription point only; you must specify the topics.

<SubIdentity> (MQPSC_SUBSCRIPTION_IDENTITY)

This is a variable-length string with a maximum length of 64 characters. It is used to represent an application with an interest in a subscription. The queue manager maintains a set of subscriber identities for each subscription. Each subscription can allow its identity set to hold only a single identity, or an unlimited number of identities.

If the `SubIdentity` is in the identity set for the subscription then it is removed from the set. If the identity set becomes empty as a result of this, the subscription is removed from the queue manager, unless `LeaveOnly` is specified as a value of the `RegOpt` property. If the identity set still contains other identities then the subscription is not removed from the queue manager, and publication flow is not interrupted.

If `SubIdentity` is specified, but the `SubIdentity` is not in the identity set for the subscription, then the **Deregister Subscriber** command fails with the return code `MQRCCF_SUB_IDENTITY_ERROR`.

<Filter> (MQPSC_FILTER)

The value is a string specifying the filter to be deregistered. It must match exactly, including case and any spaces, a subscription filter that has been previously registered.

This property can, optionally, be repeated if more than one filter is to be deregistered. It can be omitted if a `<Topic>` is specified, or if `DeregAll` is specified in `<RegOpt>`.

The filters specified can be a subset of those registered if the subscriber wants to retain subscriptions for other filters.

<RegOpt> (MQPSC_REGISTRATION_OPTION)

The registration options property can take the following values:

DeregAll

(MQPSC_DEREGISTER_ALL)

All matching subscriptions registered for this subscriber are to be deregistered.

If you specify `DeregAll`:

- `<Topic>`, `<SubPoint>`, and `<Filter>` can be omitted.
- `<Topic>` and `<Filter>` can be repeated, if required.
- `<SubPoint>` must not be repeated.

If you do **not** specify `DeregAll`:

- `<Topic>` must be specified, and can be repeated if required.
- `<SubPoint>` and `<Filter>` can be omitted.
- `<SubPoint>` must not be repeated.
- `<Filter>` can be repeated, if required.

CorrelAsId

(MQPSC_CORREL_ID_AS_IDENTITY)

The `CorrelId` in the message descriptor (MQMD), which must not be zero, is used to identify the subscriber. It must match the `CorrelId` used in the original subscription.

FullResp

()

When `FullResp` is specified all attributes of the subscription are returned in the response message, if the command does not fail.

When `FullResp` is specified `DeregAll` is not permitted in the **Deregister Subscriber** command. It is also not possible to specify multiple topics. The command fails with return code `MQRCCF_REG_OPTIONS_ERROR`, in both cases.

LeaveOnly

(MQPSC_LEAVE_ONLY)

When you specify this with a `SubIdentity` which is in the identity set for the subscription the `SubIdentity` is removed from the identity set for the subscription. The subscription is not removed from the queue manager, even if the resulting identity set is empty. If the `SubIdentity` value is not in the identity set the command fails with return code `MQRCCF_SUB_IDENTITY_ERROR`.

If `LeaveOnly` is specified with no `SubIdentity`, the command fails with return code `MQRCCF_REG_OPTIONS_ERROR`.

If neither `LeaveOnly` nor a `SubIdentity` are specified, then the subscription is removed regardless of the contents of the identity set for the subscription.

None

(MQPSC_NONE)

All options take their default values. This has the same effect as omitting the registration options property. If other options are specified at the same time, `None` is ignored.

VariableUserId

(MQPSC_VARIABLE_USER_ID)

When specified the identity of the subscriber (queue, queue manager and correlid) is not restricted to a single userid. This differs from the existing behavior of the queue manager that associates the userid of the original registration message with the subscriber's identity and from then on prevents any other user using that identity. If a new subscriber tries to use the same identity, the return code `MQRCCF_DUPLICATE_SUBSCRIPTION` is returned.

Any user can modify or deregister the subscription when they have suitable authority, avoiding the existing check that the userid must match that of the original subscriber.

To add this option to an existing subscription the command must come from the same userid as the original subscription itself.

If the subscription to be deregistered has `VariableUserId` set this must be set at deregister time to indicate which subscription is being deregistered. Otherwise, the `userid` of the **Deregister Subscriber** command is used to identify the subscription. This is overridden, along with the other subscriber identifiers, if a subscription name is supplied.

The default, if this property is omitted, is that no registration options are set.

<QMgrName> (MQPSC_Q_MGR_NAME)

The value is the queue manager name for the subscriber queue. It must match the `QMgrName` used in the original subscription.

If this property is omitted, the default is the `ReplyToQMgr` name in the message descriptor (MQMD). If the resulting name is blank, it defaults to the name of the queue manager.

<QName> (MQPSC_Q_NAME)

The value is the name of the subscriber queue. It must match the `QName` used in the original subscription.

If this property is omitted, the default is the `ReplyToQ` name in the message descriptor (MQMD), which must not be blank.

<SubName> (MQPSC_SUBSCRIPTION_NAME)

If you specify `SubName` on a **Deregister Subscriber** command the `SubName` value takes precedence over all other identifier fields except the `userid`, unless `VariableUserId` is set on the subscription itself. If `VariableUserId` is not set, the **Deregister Subscriber** command succeeds only if the `userid` of the command message matches that of the subscription, if not the command fails with return code `MQRCCF_DUPLICATE_IDENTITY`.

If a subscription exists that matches the traditional identity of this command but has no `SubName` the **Deregister Subscriber** command fails with return code `MQRCCF_SUB_NAME_ERROR`. If an attempt is made to deregister a subscription that has a `SubName` using a command message that matches the traditional identity but with no `SubName` specified the command succeeds.

<SubUserData> (MQPSC_SUBSCRIPTION_USER_DATA)


This is a variable-length text string. The value is stored by the queue manager with the subscription but has no influence on the delivery of the publication to the subscriber. The value can be altered by re-registering to the same subscription with a new value. This attribute is for the use of the application.

`SubUserData` is returned in the Metatopic information (`MQCACF_REG_SUB_USER_DATA`) for a subscription, if `SubUserData` is present.

Example

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

```
<psc>
<Command>DeregSub</Command>
<RegOpt>CorrelAsId</RegOpt>
<Topic>Sport/Soccer/State/LatestScore/#</Topic>
</psc>
```

 This build: January 26, 2011 10:57:03

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps23010_

6.7.3. Publish message

The **Publish** command message is sent from a publisher to a queue manager, or from a queue manager to a subscriber, to publish information on a specified topic or topics.

This message is sent to the input queue of a message flow that contains a publication node. Authority to put a message onto this queue, and to publish on the specified topic or topics, is necessary.

If the user has authority on some, but not all, topics, only those topics are published; a warning response indicates which topics are not published.

If a subscriber has any matching subscriptions, the queue manager forwards the **Publish** message to the subscriber queues defined in the corresponding **Register Subscriber** command messages.

See [Queue Manager Response message](#) for details of the message descriptor (MQMD) parameters needed when sending a command message to the queue manager, and used when a queue manager forwards a publication to a subscriber.

The queue manager forwards the **Publish** message to other queue managers in the network that have matching subscriptions, unless it is a local publication.

Publication data, if any, is included in the body of the message. The data can be described in an `<mcd>` folder in the `NameValueData` field of the `MQRFH2` header.

Properties

<Command> (MQPSC_COMMAND)

The value is `Publish(MQPSC_PUBLISH)`.

This property must be specified.

<Topic> (MQPSC_TOPIC)

The value is a string that contains a topic that categorizes this publication. No wildcard characters are allowed.

This property must be specified, and can optionally be repeated for as many topics as needed.

<SubPoint> (MQPSC_SUBSCRIPTION_POINT)

The subscription point on which the publication is published.

In WebSphere Event Broker V6, the value of the `<SubPoint>` property is the value of the `Subscription Point` attribute of the `Publication` node that is handling the publishing.

In WebSphere MQ V7.0.1, the value of the `<SubPoint>` property must match the name of a subscription point. See [Adding a subscription point](#).

`<PubOpt>` (**MQPSC_PUBLICATION_OPTION**)

The publication options property can take the following values:

RetainPub

(MQPSC_RETAIN_PUB)

The queue manager is to retain a copy of the publication. If this option is not set, the publication is deleted as soon as the queue manager has sent the publication to all its current subscribers.

IsRetainedPub

(MQPSC_IS_RETAINED_PUB)

(Can only be set by a queue manager.) This publication has been retained by the queue manager. The queue manager sets this option to notify a subscriber that this publication was published earlier and has been retained, provided that the subscription has been registered with the `InformIfRetained` option. It is set only in response to a **Register Subscriber** or **Request Update** command message. Retained publications that are sent directly to subscribers do not have this option set.

Local

(MQPSC_LOCAL)

This option tells the queue manager that this publication must not be sent to other queue managers. All subscribers that registered at this queue manager receive this publication if they have matching subscriptions.

OtherSubsOnly

(MQPSC_OTHER_SUBS_ONLY)

This option allows simpler processing of conference-type applications, where a publisher is also a subscriber to the same topic. It tells the queue manager not to send the publication to the publisher's subscriber queue even if it has a matching subscription. The publisher's subscriber queue consists of its `QMgrName`, `QName`, and optional `CorrelId`, as described below.

CorrelAsId

(MQPSC_CORREL_ID_AS_IDENTITY)

The `CorrelId` in the MQMD (which must not be zero) is part of the publisher's subscriber queue, in applications where the publisher is also a subscriber.

None

(MQPSC_NONE)

All options take their default values. This has the same effect as omitting the publication options property. If other options are specified at the same time, `None` is ignored.

► You can have more than one publication option by introducing additional `<PubOpt>` elements.◀

The default, if this property is omitted, is that no publication options are set.

`<PubTime>` (**MQPSC_PUBLISH_TIMESTAMP**)

The value is an optional publication timestamp set by the publisher. It is 16 characters long with format:

```
YYYYMMDDHHMSSSTH
```

using Universal Time. This information is not checked by the queue manager before being sent to the subscribers.

`<SeqNum>` (**MQPSC_SEQUENCE_NUMBER**)

The value is an optional sequence number set by the publisher.

It should be incremented by 1 with each publication. However, this is not checked by the queue manager, which merely transmits this information to subscribers.

If publications on the same topic are published to different interconnected queue managers, it is the responsibility of the publishers to ensure that sequence numbers, if used, are meaningful.

`<QMgrName>` (**MQPSC_Q_MGR_NAME**)

The value is a string containing the name of the queue manager for the publisher's subscriber queue, in applications where the publisher is also a subscriber (see `OtherSubsOnly`).

If this property is omitted, the default is the `ReplyToQMgr` name in the message descriptor (MQMD). If the resulting name is blank, it defaults to the name of the queue manager.

`<QName>` (**MQPSC_Q_NAME**)

The value is a string containing the name of the publisher's subscriber queue, in applications where the publisher is also a subscriber (see `OtherSubsOnly`).

If this property is omitted, the default is the `ReplyToQ` name in the message descriptor (MQMD), which must not be blank if `OtherSubsOnly` is set.

Example

Here are some examples of `NameValueData` for a **Publish** command message.

The first example is for a publication sent by the match simulator in the sample application to indicate that a match has started.

```
<psc>
  <Command>Publish</Command>
  <Topic>Sport/Soccer/Event/MatchStarted</Topic>
</psc>
```

The second example is for a retained publication. The latest score in the match between `Team1` and `Team2` is published.

```
<psc>
  <Command>Publish</Command>
  <PubOpt>RetainPub</PubOpt>
  <Topic>Sport/Soccer/State/LatestScore/Team1 Team2</Topic>
</psc>
```

Parent topic: ► [Command messages](#)◀

Related concepts[Subscription points and topics](#)**Related tasks**[Adding a subscription point](#)**Related information**[ALTER NAMELIST](#) This build: January 26, 2011 10:57:03[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps23020_

6.7.4. Register Subscriber message

The **Register Subscriber** command message is sent to a queue manager by a subscriber, or by another application on behalf of a subscriber, to indicate that it wants to subscribe to one or more topics at a subscription point. A message content filter can also be specified.

The **Register Subscriber** command message is sent to a queue manager by a subscriber, or by another application on behalf of a subscriber, to indicate that it wants to subscribe to one or more topics at a subscription point. A message content filter can also be specified.

In publish/subscribe filter expressions, nesting parentheses causes performance to decrease exponentially. Avoid nesting parentheses to a depth greater than about 6.

The message is sent to `SYSTEM.BROKER.CONTROL.QUEUE`, which is the queue manager's control queue. Authority to put a message to this queue is required, in addition to access authority (set by the queue manager's system administrator) for the topic, or topics, in the subscription.

If the user has authority on some, but not all, topics, only those with authority are registered; a warning response indicates those that are not registered.

See [Queue Manager Response message](#) for details of the message descriptor (MQMD) parameters that are needed when sending a command message to the queue manager.

If the queue is a temporary dynamic queue, the subscription is deregistered automatically by the queue manager when the queue is closed.

Properties

<Command> (MQPSC_COMMAND)

The value is `RegSub` (MQPSC_REGISTER_SUBSCRIBER). This property must be specified.

<Topic> (MQPSC_TOPIC)

The topic for which the subscriber wants to receive publications. Wildcard characters can be specified as part of the topic.

This property is required, and can optionally be repeated for as many topics as needed.

<SubPoint> (MQPSC_SUBSCRIPTION_POINT)

The value is the subscription point to which the subscription is attached.

If this property is omitted, the default subscription point is used.

In WebSphere Event Broker V6, the value of the `<SubPoint>` property must match the value of the `Subscription Point` attribute of the `Publication` nodes that are subscribed to.

In WebSphere MQ V7.0.1, the value of the `<SubPoint>` property must match the name of a subscription point. See [Adding a subscription point](#).

<Filter> (MQPSC_FILTER)

The value is an SQL expression that is used as a filter on the contents of publication messages. If a publication on the specified topic matches the filter, it is sent to the subscriber.

If this property is omitted, no content filtering takes place.

<RegOpt> (MQPSC_REGISTRATION_OPTION)

This Registration Options property can take the following values:

AddName

(MQPSC_ADD_NAME)

When specified for an existing subscription that matches the traditional identity of this Register Subscription command, but with no current `SubName` value, the `SubName` specified in this command is added to the subscription.

If `AddName` is specified the `SubName` field is mandatory, otherwise `MQRCCF_REG_OPTIONS_ERROR` is returned.

CorrelAsId

(MQPSC_CORREL_ID_AS_IDENTITY)

The `CorrelId` in the message descriptor (MQMD) is used when sending matching publications to the subscriber queue. The `CorrelId` must not be zero,

FullResp

(MQPSC_FULL_RESPONSE)

When specified all attributes of the subscription are returned in the response message, if the command does not fail.

`FullResp` is valid only when the command message refers to a single subscription. Therefore, only one topic is permitted in the command; otherwise the command fails with return code `MQRCCF_REG_OPTIONS_ERROR`.

InformIfRet

(MQPSC_INFORM_IF_RETAINED)

The queue manager informs the subscriber if a publication is retained when it sends a `Publish` message in response to a **Register Subscriber** or

Request Update command message. The queue manager does this by including the `IsRetainedPub` publication option in the message.

JoinExcl

(MQPSC_JOIN_EXCLUSIVE)

This option indicates that the specified `SubIdentity` must be added as the exclusive member of the identity set for the subscription, and that no other identities can be added to the set.

If the identity has already joined 'shared' and is the sole entry in the set, the set is changed to an exclusive lock held by this identity. Otherwise, if the subscription currently has other identities in the identity set (with shared access) the command fails with return code `MQRCCF_SUBSCRIPTION_IN_USE`.

JoinShared

(MQPSC_JOIN_SHARED)

This option indicates that the specified `SubIdentity` must be added to the identity set for the subscription.

If the subscription is currently locked exclusively (using the `JoinExcl` option), the command fails with return code `MQRCCF_SUBSCRIPTION_LOCKED`, unless the identity that has the subscription locked is the same identity as that in this command message. In this case the lock is automatically modified to a shared lock.

Local

(MQPSC_LOCAL)

The subscription is local and is not distributed to other queue managers in the network. Publications made at other queue managers are not delivered to this subscriber, unless it also has a corresponding global subscription.

NewPubsOnly

(MQPSC_NEW_PUBS_ONLY)

Retained publications that exist at the time the subscription is registered are not sent to the subscriber; only new publications are sent.

If a subscriber re-registers and changes this option so that it is no longer set, a publication that has already been sent to it might be sent again.

NoAlter

(MQPSC_NO_ALTER)

The attributes of an existing matching subscription is not changed.

When a subscription is being created, this option is ignored. All other options specified apply to the new subscription.

If a `SubIdentity` also has one of the join options (`JoinExcl` or `JoinShared`) specified, the identity is added to the identity set regardless of whether `NoAlter` is specified.

None

(MQPSC_NONE)

All registration options take their default values.

If the subscriber is already registered, its options are reset to their default values (note that this does not have the same affect as omitting the registration options property), and the subscription expiry is updated from the MQMD of the **Register Subscriber** message.

If other registration options are specified at the same time, `None` is ignored.

NonPers

(MQPSC_NON_PERSISTENT)

Publications matching this subscription are delivered to the subscriber as non-persistent messages.

Pers

(MQPSC_PERSISTENT)

Publications matching this subscription are delivered to the subscriber as persistent messages.

PersAsPub

(MQPSC_PERSISTENT_AS_PUBLISH)

Publications matching this subscription are delivered to the subscriber with the persistence specified by the publisher. This is the default behavior.

PersAsQueue

(MQPSC_PERSISTENT_AS_Q)

Publications matching this subscription are delivered to the subscriber with the persistence specified on the subscriber queue.

PubOnReqOnly

(MQPSC_PUB_ON_REQUEST_ONLY)

The queue manager does not send publications to the subscriber, except in response to a **Request Update** command message.

VariableUserId

(MQPSC_VARIABLE_USER_ID)

When specified the identity of the subscriber (queue, queue manager and correlid) is not restricted to a single userid. This differs from the existing behavior of the queue manager that associates the userid of the original registration message with the subscriber's identity and from then on prevents any other user using that identity. If a new subscriber tries to use the same identity `MQRCCF_DUPLICATE_SUBSCRIPTION` is returned.

This allows any user to modify or deregister the subscription if the user has suitable authority. There is therefore no need to check that the userid matches that of the original subscriber.

To add this option to an existing subscription the command must come from the same userid as the original subscription itself.

If the subscription of the **Request Update** command has `VariableUserId` set, this must be set at request update time to indicate which subscription is referred to. Otherwise, the userid of the **Request Update** command is used to identify the subscription. This is overridden, along with the other subscriber identifiers, if a subscription name is supplied.

If a **Register Subscriber** command message without this option set refers to an existing subscription which has this option set, the option is removed from this subscription and the userid of the subscription is now fixed. If there already exists a subscriber which has the same identity (queue, queue manager and correlation identifier) but with a different user ID associated to it, the command fails with return code `MQRCCF_DUPLICATE_IDENTITY` because there can only be one userid associated with a subscriber identity.

If the registration options property is omitted and the subscriber is already registered, its registration options are not changed and the subscription expiry is updated from the MQMD of the **Register Subscriber** message.

If the subscriber is not already registered, a new subscription is created with all registration options taking their default values.

The default values are `PersAsPub` and no other options set.

<QMgrName> (MQPSC_Q_MGR_NAME)

The value is the name of the queue manager for the subscriber queue, to which matching publications are sent by the queue manager.

If this property is omitted, the default is the `ReplyToQMgr` name in the message descriptor (MQMD). If the resulting name is blank, it defaults to the queue manager's `QMgrName`.

<QName> (MQPSC_Q_NAME)

The value is the name of the subscriber queue, to which matching publications are sent by the queue manager.

If this property is omitted, the default is the `ReplyToQ` name in the message descriptor (MQMD), which must not be blank in this case.

If the queue is a temporary dynamic queue, nonpersistent delivery of publications (`NonPers`) must be specified in the <RegOpt> property.

If the queue is a temporary dynamic queue, the subscription is deregistered automatically by the queue manager when the queue is closed.

<SubName> (MQPSC_SUBSCRIPTION_NAME)

This is a name given to a particular subscription. You can use it instead of the queue manager, queue and optional `CorrelId` to refer to a subscription.

If a subscription already exists with this `SubName`, any other attributes of the subscription (`Topic`, `QMgrName`, `QName`, `CorrelId`, `UserId`, `RegOpts`, `UserSubData`, and `Expiry`) are overridden with the attributes, if specified, that are passed in the new **Register Subscriber** command message. However, if `SubName` is used with no `QName` field specified, and a `ReplyToQ` is specified in the MQMD header, the subscriber queue is changed to be the `ReplyToQ`.

If a subscription that matches the traditional identity of this command already exists, but has no `SubName`, the Registration command fails with return code `MQRCCF_DUPLICATE_SUBSCRIPTION`, unless the `AddName` option is specified.

If you try to alter an existing named subscription by using another **Register Subscriber** command that specifies the same `SubName`, and the values of `Topic`, `QMgrName`, `QName`, and `CorrelId` in the new command match a different existing subscription, with or without a `SubName` defined, the command fails with return code `MQRCCF_DUPLICATE_SUBSCRIPTION`. This prevents two subscription names referring to the same subscription.

<SubIdentity> (MQPSC_SUBSCRIPTION_IDENTITY)

This string is used to represent an application with an interest in a subscription. It is a variable-length character string whose maximum length is 64 characters, and is optional. The queue manager maintains a set of subscriber identities for each subscription. Each subscription can allow its identity set to contain only one identity, or an unlimited number of identities (see the `JoinShared` and `JoinExcl` options).

A subscribe command that specifies the `JoinShared` or `JoinExcl` option adds the `SubIdentity` to the subscription's identity set, if it is not already there and if the existing set of identities allows such an action; that is, no other subscriber has joined exclusively or the identity set is empty.

Any alteration of the subscription's attributes as the result of a **Register Subscriber** command in which a `SubIdentity` is specified, only succeeds if it would be the only member of the set of identities for this subscription. Otherwise the command fails with return code `MQRCCF_SUBSCRIPTION_IN_USE`. This prevents a subscription's attributes from changing without other interested subscribers being aware.

If you specify a character string that is longer than 64 characters, the command fails with return code `MQRCCF_SUB_IDENTITY_ERROR`.

<SubUserData> (MQPSC_SUBSCRIPTION_USER_DATA)

This is a variable-length text string. The value is stored by the queue manager with the subscription, but has no influence on publication delivery to the subscriber. The value can be altered by re-registering to the same subscription with a new value. This attribute is there for the use of the application.

The `SubUserData` is returned in the Metatopic information (`MQCACF_REG_SUB_USER_DATA`) for a subscription if present.

If you specify more than one of the registration option values `NonPers`, `PersAsPub`, `PersAsQueue`, and `Pers`, then only the last one is used. You cannot combine these options in an individual subscription.

Example

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

```
<psc>
<Command>RegSub</Command>
<RegOpt>PubOnReqOnly</RegOpt>
<RegOpt>CorrelAsId</RegOpt>
<Topic>Sport/Soccer/State/LatestScore/#</Topic>
</psc>
```

Parent topic: [Command messages](#)

Related concepts

[Subscription points and topics](#)

Related tasks

[Adding a subscription point](#)

 This build: January 26, 2011 10:57:07

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps23030_

6.7.5. Request Update message

The **Request Update** command message is sent from a subscriber to a queue manager, to request the current retained publications for the specified topic and subscription point that match the given (optional) filter.

This message is sent to `SYSTEM.BROKER.CONTROL.QUEUE`, the queue manager's control queue. Authority to put a message to this queue is required, in addition to access authority for the topic in the request update; this is set by the queue manager's system administrator.

This command is normally used if the subscriber specified the option `PubOnReqOnly` when it registered. If the queue manager has any matching retained publications, they are sent to the subscriber. If the queue manager has no matching retained publications, the request fails with return code `MQRCCF_NO_RETAINED_MSG`. The requester must have previously registered a subscription with the same Topic, SubPoint, and Filter values.

Parent topic: [Command messages](#)

Properties

<Command> (MQPSC_COMMAND)

The value is `ReqUpdate` (`MQPSC_REQUEST_UPDATE`). This property must be specified.

<Topic> (MQPSC_TOPIC)

The value is the topic that the subscriber is requesting; wildcard characters are allowed.

This property must be specified, but only one occurrence is allowed in this message.

<SubPoint> (MQPSC_SUBSCRIPTION_POINT)

The value is the subscription point to which the subscription is attached.

If this property is omitted, the default subscription point is used.

<Filter> (MQPSC_FILTER)

The value is an ESQL expression that is used as a filter on the contents of publication messages. If a publication on the specified topic matches the filter, it is sent to the subscriber.

The `<Filter>` property should have the same value as that specified on the original subscription for which you are now requesting an update.

If this property is omitted, no content filtering takes place.

<RegOpt> (MQPSC_REGISTRATION_OPTION)

The registration options property can take the following value:

CorrelAsId

(`MQPSC_CORREL_ID_AS_IDENTITY`)

The `CorrelId` in the message descriptor (`MQMD`), which must not be zero, is used when sending matching publications to the subscriber queue.

None

(`MQPSC_NONE`)

All options take their default values. This has the same effect as omitting the `<RegOpt>` property. If other options are specified at the same time, `None` is ignored.

VariableUserId

(`MQPSC_VARIABLE_USER_ID`)

When specified the identity of the subscriber (queue, queue manager, and `correlid`) is not restricted to a single `userid`. This differs from the existing behavior of the queue manager that associates the `userid` of the original registration message with the subscriber's identity and from then on prevents any other user using that identity. If a new subscriber tries to use the same identity, the command fails with return code `MQRCCF_DUPLICATE_SUBSCRIPTION`.

This allows any user to modify or deregister the subscription when they have suitable authority. Therefore, there is no need to check that the `userid` matches that of the original subscriber.

To add this option to an existing subscription, the command must come from the same `userid` as the original subscription.

If the subscription of the **Request Update** command has `VariableUserId` set, this must be set at request update time to indicate which subscription is referred to. Otherwise, the `userid` of the **Request Update** command is used to identify the subscription. This is overridden, along with the other subscriber identifiers, if a subscription name is supplied.

The default, if this property is omitted, is that no registration options are set.

<QMgrName> (MQPSC_Q_MGR_NAME)

The value is the name of the queue manager for the subscriber queue, to which the matching retained publication is sent by the queue manager.

If this property is omitted, the default is the `ReplyToQMgr` name in the message descriptor (`MQMD`). If the resulting name is blank, it defaults to the queue manager's `QMgrName`.

<QName> (MQPSC_Q_NAME)

The value is the name of the subscriber queue, to which the matching retained publication is sent by the queue manager.

If this property is omitted, the default is the `ReplyToQ` name in the message descriptor (`MQMD`), which must not be blank in this case.

<SubName> (MQPSC_SUBSCRIPTION_NAME)

This is a name given to a particular subscription. If specified on a **Request Update** command the `SubName` value takes precedence over all other identifier fields except the `userid`, unless `VariableUserId` is set on the subscription itself. If `VariableUserId` is not set, the **Request Update** command succeeds only if the `userid` of the command message matches that of the subscription. If the `userid` of the command message does not match that of the subscription, the command fails with return code `MQRCCF_DUPLICATE_IDENTITY`.

If `VariableUserId` is set, and the `userid` differs from that of the subscription, the command succeeds if the `userid` of the new command message has authority to browse the stream queue and put to the subscriber queue of the subscription. Otherwise, the command fails with return code `MQRCCF_NOT_AUTHORIZED`.

If a subscription exists that matches the traditional identity of this command, but has no `SubName`, the **Request Update** command fails with return code `MQRCCF_SUB_NAME_ERROR`.

If an attempt is made to request an update for a subscription that has a `SubName` using a command message that matches the traditional identity, but with no `SubName` specified, the command succeeds.

Example

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

```
<psc>
  <Command>ReqUpdate</Command>
  <RegOpt>CorrelAsId</RegOpt>
  <Topic>Sport/Soccer/State/LatestScore/#</Topic>
</psc>
```

 This build: January 26, 2011 10:57:07

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps23040_

6.7.6. Queue Manager Response message

A **Queue Manager Response** message is sent from a queue manager to the `ReplyToQ` of a publisher or a subscriber, to indicate the success or failure of a command message received by the queue manager if the command message descriptor specified that a response is required.

The response message is contained within the `NameValueData` field of the MQRFH2 header, in a `<pscr>` folder.

In the case of a warning or error, the response message contains the `<psc>` folder from the command message as well as the `<pscr>` folder. The message data, if any, is not contained in the queue manager response message. In the case of an error, none of the message that caused an error has been processed; in the case of a warning, some of the message might have been processed successfully.

If there is a failure sending a response:

- For publication messages, the queue manager tries to send the response to the WebSphere® MQ dead-letter queue if the MQPUT fails. This allows the publication to be sent to subscribers even if the response cannot be sent back to the publisher.
- For other messages, or if the publication response cannot be sent to the dead-letter queue, an error is logged and the command message is normally rolled back. Whether this happens depends on how the MQInput node has been configured.

Parent topic: [Command messages](#)

Properties

<Completion> (MQPSCR_COMPLETION)

The completion code, which can take one of three values:

ok

Command completed successfully

warning

Command completed but with warning

error

Command failed

<Response> (MQPSCR_RESPONSE)

The response to a command message, if that command produced a completion code of `warning` or `error`. It contains a `<Reason>` property, and might contain other properties that indicate the cause of the warning or error.

In the case of one or more errors, there is only one response folder, indicating the cause of the first error only. In the case of one or more warnings, there is a response folder for each warning.

<Reason> (MQPSCR_REASON)

The reason code qualifying the completion code, if the completion code is a `warning` or `error`. It is set to one of the error codes listed below. The `<Reason>` property is contained within a `<Response>` folder. The reason code can be followed by any valid property from the `<psc>` folder (for example, a topic name), indicating the cause of the error or warning.

Examples

Here are some examples of `NameValueData` in a **Queue Manager Response** message. A successful response might be the following:

```
<pscr>
  <Completion>ok</Completion>
</pscr>
```

Here is an example of a failure response; the failure is a filter error. The first `NameValueData` string contains the response; the second contains the original command.

```
<pscr>
  <Completion>error</Completion>
  <Response>
    <Reason>3150</Reason>
  </Response>
</pscr>

<psc>
  ...
  command message (to which
  the queue manager is responding)
  ...
</psc>
```


Here is an example of a warning response (due to unauthorized topics). The first `NameValueData` string contains the response; the second `NameValueData` string contains the original command.

```

<pscr>
  <Completion>warning</Completion>
  <Response>
    <Reason>3081</Reason>
    <Topic>topic1</Topic>
  </Response>
  <Response>
    <Reason>3081</Reason>
    <Topic>topic2</Topic>
  </Response>
</pscr>

<psc>
  ...
  command message (to which
  the queue manager is responding)
  ...
</psc>

```

 This build: January 26, 2011 10:57:08

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps23050_

6.7.7. Reason codes

These reason codes might be returned in the *Reason* field of a publish/subscribe response `<pscr>` folder. Constants that can be used to represent these codes in the C or C++ programming languages are also listed.

The *MQRC_* constants require the WebSphere MQ *cmqc.h* header file. The *MQRCCF_* constants require the WebSphere MQ *cmqcf.h* header file (apart from *MQRCCF_FILTER_ERROR* and *MQRCCF_WRONG_USER*, which require the *cmqpsc.h* header file).

Reason code and text	Explanation	Issued by
2336 MQRC_RFH_COMMAND_ERROR	Valid values for the <code><Command></code> field of a <code><psc></code> folder are: RegSub, DeregSub, Publish, DeletePub, and ReqUpdate. Any other values result in this error code being issued.	Any command
2337 MQRC_RFH_PARM_ERROR	The <code><psc></code> and <code><cmd></code> folders both have a set of valid parameters that can be specified within them. Check the descriptions of these folders and ensure that you have not specified incorrect parameters.	Any command
2338 MQRC_RFH_DUPLICATE_PARM	Some parameters (for example, Topic) within a <code><psc></code> folder can be repeated, but others (for example, Command) cannot be repeated. Check that you have not duplicated a non-repeatable parameter.	Any command
2339 MQRC_RFH_PARM_MISSING	Some parameters within <code><psc></code> or <code><cmd></code> folders are optional and can be omitted; some are mandatory and must not be omitted. Check that you have included all mandatory parameters within your <code><psc></code> and <code><cmd></code> folders.	Any command
3008 MQRCCF_COMMAND_FAILED	An internal error occurred which prevented the command from executing correctly. The error might occur if the command is reissued. The system event log for the queue manager contains information which should be used when reporting the problem to IBM.	Any command
3072 MQRCCF_TOPIC_ERROR	One or more of the values you supplied for the Topic parameter are incorrect. Check that your values for Topic conform to the specified restrictions.	Any command
3073 MQRCCF_NOT_REGISTERED	The combination of SubPoint, Topic, and Filter that you specified on your DeregSub or ReqUpdate command was either not a combination with which you had previously registered or, for the DeregSub command if the DeregAll option was specified, one of the SubPoint, Topic, or Filter properties was not used to deregister any subscription.	Deregister Subscriber and Request Update commands
3074 MQRCCF_Q_MGR_NAME_ERROR	The specified queue manager was not valid, or the queue manager was not available or did not exist.	Deregister Subscriber, Publish, Register Subscriber, and Request Update commands
3076 MQRCCF_Q_NAME_ERROR	The specified queue name was not valid, or the queue did not exist on the specified queue manager.	Deregister Subscriber, Publish, Register Subscriber, and Request Update commands
3077 MQRCCF_NO_RETAINED_MSG	There were no retained messages for the topic you specified. This might or might not be an error, depending on the design of your application program.	Request Update command
3079 MQRCCF_INCORRECT_Q	RegSub, DeregSub, and ReqUpdate commands are always sent to the SYSTEM.BROKER.CONTROL.QUEUE queue of the queue manager for which they are intended. Publish and Delete Publication commands are sent to the input queue for the particular publish/subscribe message flow for which they are intended; this is determined when the message flow is designed. This error code is returned if a command is sent to the wrong queue.	Any command

3080 MQRCCF_CORREL_ID_ERROR	You have specified CorrelAsId as one of your RegOpt parameters. However, the CorrelId field of the MQMD does not contain a valid correlation identifier (that is, it is set to MQCI_NONE).	Deregister Subscriber and Register Subscriber commands
3081 MQRCCF_NOT_AUTHORIZED	You are not authorized to perform the requested action. Authorization settings for the queue manager are handled by the system administrator using the Topics Hierarchy editor.	Publish and Register Subscriber commands
3083 MQRCCF_REG_OPTIONS_ERROR	You have specified an unrecognized RegOpt parameter in the <psc> folder that contains your RegSub or DeregSub command.	Deregister Subscriber and Register Subscriber commands
3084 MQRCCF_PUB_OPTIONS_ERROR	You have specified an unrecognized PubOpt parameter in the <psc> folder that contains your Publish command.	Publish command
3087 MQRCCF_DEL_OPTIONS_ERROR	You have specified an unrecognized DelOpt parameter in the <psc> folder that contains your DeletePub command.	Delete Publication command
3150 MQRCCF_FILTER_ERROR	The value specified for the Filter parameter is not valid. Check the section that describes the valid syntax for filter expressions and ensure that your expression conforms.	Deregister Subscriber, Register Subscriber, and Request Update commands
3151 MQRCCF_WRONG_USER	A subscription that matches the one specified already exists; however, it was registered by a different user. A subscription can only be changed or deregistered by the user who originally registered it.	Deregister Subscriber, Register Subscriber, and Request Update commands
3152 MQRCCF_DUPLICATE_SUBSCRIPTION	A matching subscription already exists with a different subscription name.	
3153 MQRCCF_SUB_NAME_ERROR	Either the format of the subscription name is not valid, or a matching subscription already exists with no subscription name.	
3154 MQRCCF_SUB_IDENTITY_ERROR	The subscription identity parameter is in error. Either the supplied value exceeds the maximum length allowed, or the subscription identity is not currently a member of the subscription's identity set and a Join registration option was not specified.	
3155 MQRCCF_SUBSCRIPTION_IN_USE	An attempt to modify or deregister a subscription was attempted by a member of the identity set when it was not the only member of this set.	
3156 MQRCCF_SUBSCRIPTION_LOCKED	The subscription is currently exclusively locked by another identity.	
3157 MQRCCF_ALREADY_JOINED	A Join registration option was specified but the subscriber identity was already a member of the subscription's identity set.	

Parent topic: [Command messages](#)

 This build: January 26, 2011 10:57:09

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps23090_

6.7.8. MQMD settings in command messages to the queue manager

Applications that send command messages to the queue manager use the following settings of fields in the message descriptor (MQMD). Fields that are left as the default value, or can be set to any valid value in the usual way, are not listed here.

Report

See *MsgType* and *CorrelId* (below).

MsgType

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

If *MsgType* is set to *MQMT_DATAGRAM*, responses depend on the setting of the *MQRO_PAN* and *MQRO_NAN* flags in the *Report* field:

- *MQRO_PAN* alone means that the queue manager sends a response only if the command succeeds.
- *MQRO_NAN* alone means that the queue manager sends a response only if the command fails.
- If a command completes with a warning, a response is sent if either *MQRO_PAN* or *MQRO_NAN* is set.
- *MQRO_PAN* + *MQRO_NAN* means that the queue manager sends a response whether the command succeeds or fails. This has the same effect from the queue manager's perspective as setting *MsgType* to *MQMT_REQUEST*.
- If neither *MQRO_PAN* nor *MQRO_NAN* is set, no response is ever sent.

Format

Set to *MQFMT_RF_HEADER_2*

MsgId

This field is normally set to *MQMI_NONE*, so that the queue manager generates a unique value.

CorrelId

This field can be set to any value. If the sender's identity includes a *CorrelId*, specify this value, together with *MQRO_PASS_CORREL_ID* in the *Report* field, to ensure that it is set in all response messages sent by the queue manager to the sender.

ReplyToQ

This field defines the queue to which responses, if any, are to be sent. This might be the sender's queue; this has the advantage that the `QName` parameter can be omitted from the message. If, however, responses are to be sent to a different queue, the `QName` parameter is needed.

ReplyToQMGr

This field defines the queue manager for responses. If you leave this field blank (the default value), the local queue manager puts its own name in this field.

Parent topic: [Command messages](#)

 This build: January 26, 2011 10:57:08

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps23060_

6.7.9. MQMD settings for publications forwarded by a queue manager

A queue manager uses these settings of fields in the message descriptor (MQMD) when it sends a publication to a subscriber. All other fields in the MQMD are set to their default values.

Report

Report is set to `MQRO_NONE`.

MsgType

MsgType is set to `MQMT_DATAGRAM`.

Expiry

Expiry is set to the value in the **Publish** message received from the publisher. In the case of a retained message, the time outstanding is reduced by the approximate time that the message has been at the queue manager.

Format

Format is set to `MQFMT_RF_HEADER_2`

MsgId

MsgId is set to a unique value.

CorrelId

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

Priority

Priority takes the value set by the publisher, or as resolved if the publisher specified `MQPRI_PRIORITY_AS_Q_DEF`.

Persistence

Persistence takes the value set by the publisher, or as resolved if the publisher specified `MQPER_PERSISTENCE_AS_Q_DEF`, unless specified otherwise in the **Register Subscriber** message for the subscriber to which this publication is being sent.

ReplyToQ

ReplyToQ is set to blanks.

ReplyToQMGr

ReplyToQMGr is set to the name of the queue manager.

UserIdentifier

UserIdentifier is the subscriber's user identifier, as set when the subscriber registered.

AccountingToken

AccountingToken is the subscriber's accounting token, as set when the subscriber first registered.

AppIdentityData

AppIdentityData is the subscriber's application identity data, as set when the subscriber first registered.

PutAppType

PutAppType is set to `MQAT_BROKER`.

PutAppName

PutAppName is set to the first 28 characters of the name of the queue manager.

PutDate

PutDate is the date when the message was put.

PutTime

PutTime is the time when the message was put.

AppOriginData

AppOriginData is set to blanks.

Parent topic: [Command messages](#)

 This build: January 26, 2011 10:57:08

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps23070_

6.7.10. MQMD settings in queue manager response messages

A queue manager uses these settings of fields in the message descriptor (MQMD) when sending a reply to a publication message. All other fields in the MQMD are set to their default values.

Report

Report is set to all zeroes.

MsgType

MsgType is set to MQMT_REPLY.

Format

Format is set to MQFMT_RF_HEADER_2

MsgId

The setting of *MsgId* depends on the *Report* options in the original command message. By default, it is set to MQMI_NONE, so that the queue manager generates a unique value.

CorrelId

The setting of *CorrelId* depends on the *Report* options in the original command message. By default, this means that the *CorrelId* is set to the same value as the *MsgId* of the command message. This can be used to correlate commands with their responses.

Priority

Priority is set to the same value as in the original command message.

Persistence

Persistence is set to the value set in the original command message.

Expiry

Expiry is set to the same value as in the original command message received by the queue manager.

PutApplType


PutApplType is set to MQAT_BROKER.

PutApplName

PutApplName is set to the first 28 characters of name of the queue manager.

Other context fields are set as if generated with MQPMO_PASS_IDENTITY_CONTEXT.

Parent topic: [Command messages](#)

 This build: January 26, 2011 10:57:08

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps23080_

7. WebSphere MQ version 6 publish/subscribe migration

Publish/subscribe function in WebSphere MQ Version 7.0 is performed by the queue manager, rather than by a separate *publish/subscribe* broker. When you migrate your WebSphere MQ Version 6 systems to WebSphere MQ Version 7.0, *publish/subscribe* function is not automatically migrated. You must upgrade *publish/subscribe* information to WebSphere MQ Version 7.0 separately.



In WebSphere MQ V6, applications perform *publish* and *subscribe* operations by placing special request messages on certain queues. The WebSphere MQ V6 *Publish/Subscribe* Broker then reads and acts on these messages (for example by publishing messages to subscribing applications). State information such as who is subscribing to which publications is owned and maintained by the *publish/subscribe* broker. This broker is started and stopped independently from the queue manager.

In WebSphere MQ V7, newly written *publish/subscribe* applications do not communicate with the broker in order to *publish* or *subscribe*; they use the new API directly. The verb **MQPUT** is used to publish messages to a topic and **MQSUB** is used to subscribe. The queue manager itself performs the *publish/subscribe* function, so no separate *publish/subscribe* broker is required.

When you upgrade a queue manager from WebSphere MQ V6 to WebSphere MQ V7, the *publish/subscribe* broker is not upgraded. State information must be migrated from the WebSphere MQ *publish/subscribe* broker into the queue manager. Data that is migrated includes subscriptions, retained publications, hierarchy relations, and authorities. You migrate a queue manager by using the **strmqbrk** command, which previously started the *publish/subscribe* broker.

WebSphere MQ V6 *publish/subscribe* brokers could be connected into hierarchies so that publications and subscriptions could flow between them. After migrating (using **strmqbrk**) these hierarchies continue to function in WebSphere MQ V7. WebSphere MQ V7 also contains a new method of allowing publications and subscriptions flow between queue managers; *publish/subscribe* clusters. An advantage of using *publish/subscribe* clusters is that no queue manager provides a single point of failure to the flow of publications or subscriptions. To migrate to a *publish/subscribe* cluster, first migrate to a WebSphere MQ V7 hierarchy using **strmqbrk** and then convert it to a cluster by creating cluster topics and altering parent/child relations.



[strmqbrk \(Migrate WebSphere MQ Version 6.0 broker to Version 7.0\)](#)

Migrate the persistent state of a Websphere MQ *publish/subscribe* broker.

[Publish/subscribe command messages migration](#)

The WebSphere® MQ *publish/subscribe* command message interface is deprecated in version 7.0. If you have applications that use this interface directly, you should migrate those applications to use the new Version 7.0 *publish/subscribe* functions.

[New queue manager attributes for publish/subscribe](#)

Five attributes, formerly held in the queue manager configuration file, *qm.ini*, are now replaced by attributes of the queue manager.

[WebSphere MQ publish/subscribe topology migration](#)

Migrate a version 6 *publish/subscribe* hierarchy to a version 7 hierarchy, or convert it to a *publish/subscribe* cluster, before running existing version 6 *publish/subscribe* applications on version 7.

[Differences from WebSphere MQ Version 6 publish/subscribe](#)

Queued *publish/subscribe* programs and queued broker administration in version 7 differ from that in version 6. The differences in program behavior are slight; the administration differences are more extensive. Many version 6 programs coexist and interoperate with version 7, without change.

[Using publish/subscribe with WebSphere MQ classes for JMS](#)

Existing WebSphere MQ classes for JMS applications run unchanged after you upgrade your queue manager to WebSphere MQ V7.0. In some circumstances, you must specify whether WebSphere MQ classes for JMS uses WebSphere MQ Version 6.0 or Version 7.0 *publish/subscribe* function.

[Migration implications of mapping an alias queue to a topic object](#)

WebSphere MQ Version 7.0 introduces an extension to the *alias* queue object that allows an *alias* queue to be mapped to a topic object.

Parent topic: [Publish/Subscribe User's Guide](#)

This build: January 26, 2011 10:56:48

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps21080_

7.1. strmqbrk (Migrate WebSphere MQ Version 6.0 broker to Version 7.0)

Migrate the persistent state of a Websphere MQ publish/subscribe broker.

Purpose

► Use the **strmqbrk** command to migrate WebSphere® MQ Version 6.0 publish/subscribe broker's state to WebSphere MQ Version 7.0 publish/subscribe. If the queue manager has already been migrated, no action is taken. ◀

In WebSphere MQ Version 6.0, **strmqbrk** started a broker. The WebSphere MQ Version 7.0 publish/subscribe engine cannot be started in this manner. To enable publish/subscribe for a queue manager, use the ALTER QMGR command. For details, see [ALTER QMGR](#).

► You can also use the **runmqbrk** command. This has the same parameters as **strmqbrk** and exactly the same effect. ◀

►

Syntax

AIX®, HP-UX, Linux, Solaris, and Windows

[Syntax diagram format](#) [Railroad diagram](#) [Dotted decimal](#)
QMGrName\n' + '4? -f\n' + '5? -l LogFileName\n' + '\n' + '

);break; default:document.write('

```
\n'
+ '>>-strmqbrk--#-----#-----#-----#----->\n'
+ ' \'- -p --ParentQMGrName-\ ' \'- -m --QMGrName-\ ' \n'
+ '\n'
+ '>--#-----#-----#-----#-----><\n'
+ ' \'- -f -\ ' \'- -l --LogFileName-\ ' \n'
+ '\n'
+ '
```

);} //]]>

◀

►

Optional parameters

AIX, HP-UX, Linux, Solaris, and Windows

-p ParentQMGrName

Note: This option is deprecated. **strmqbrk** migrates the parent connection automatically.

If you specify the current parent queue manager, a warning message is issued and migration continues. If you specify a different queue manager, an error is issued and migration is not performed.

-m QMGrName

The name of the queue manager to be migrated. If you do not specify this parameter, the command is routed to the default queue manager.

-f

Force migration. This option specifies that objects created during the migration replace existing objects with the same name. If this option is not specified, if migration would create a duplicate object, a warning is issued, the object is not created, and migration continues.

-l LogFileName

Log migration activity to the file specified in LogFileName.

◀

►

Syntax

i5/OS®

[Syntax diagram format](#) [Railroad diagram](#) [Dotted decimal](#)
QMGrName\n' + '\n' + '

);break; default:document.write('

```
\n'
+ '>>-STRMQBRK--#-----#-----#-----#----->\n'
+ ' \'- -PARENTMQM--(ParentQMGrName)-\ ' \n'
+ '\n'
+ '>--#-----#-----#-----#-----><\n'
+ ' \'- -MQMNAME --QMGrName-\ ' \n'
+ '\n'
+ '
```

);} //]]> ◀

►

Optional parameters

AIX, HP-UX, Linux, Solaris, and Windows

-PARENTMQM(ParentQMGrName)

Note: This option is deprecated.

If you specify the current parent queue manager, a warning message is issued and migration continues. If you specify a different queue manager, a warning is issued and migration is not performed.

-MQMNAME QMGrName

The name of the queue manager to be migrated. If you do not specify this parameter, the command is routed to the default queue manager.



Parent topic: [WebSphere MQ version 6 publish/subscribe migration](#)

This build: January 26, 2011 10:56:25

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps12010_

7.2. Publish/subscribe command messages migration

The WebSphere® MQ publish/subscribe command message interface is deprecated in version 7.0. If you have applications that use this interface directly, you should migrate those applications to use the new Version 7.0 publish/subscribe functions.

The following sections explain how to replace existing command messages.

Identity

In WebSphere MQ Version 6 there were two ways of identifying a subscriber. These were referred to as the traditional identity and the subscription name.

The traditional identity was also used to identify a publisher. The traditional identity was a combination of queue name, queue manager name, and optional correlation identifier.

A publisher no longer has an explicit publisher identity, but can be identified in the same way as any other WebSphere MQ application, by means of its connection to the queue manager. As there is no explicit registration of a publisher, or his identity over and above what can be obtained by displaying the connections to the queue manager, there is no longer a need for the anonymous option on Register Publisher. Your application must now use the SubName field in the MQSD to identify a subscriber.

The correlation identifier also had a secondary use which was to allow subscribers to MQGET by CorrelId to only get publications for a particular subscription, if there were multiple subscriptions all using the same queue. This is provided by using the SubCorrelId field returned in the MQSD from the MQSUB call.

Stream Name

MQPS_STREAM_NAME is deprecated because stream names are part of the full topic name. Stream names can be mapped to administrative topic objects, and then the topic name used along with the stream name can be mapped to a topic string to be concatenated with the topic string from the topic object. For example, if the application was previously using a stream queue name of SYSTEM.BROKER.RESULTS.STREAM and a topic of Sport/Soccer/State/LatestScore/*, then a topic object can be created whose name is SYSTEM.BROKER.RESULTS.STREAM which is defined to have a topic string of / and the new application will provide a two part topic name in the MQOD or MQSD using an ObjectName of SYSTEM.BROKER.RESULTS.STREAM and an ObjectString of Sport/Soccer/State/LatestScore/*.

If an administrative topic object that does not exist is used in place of a stream name, the error (effectively mapping to MQRCCF_STREAM_ERROR) which is given is MQRC_UNKNOWN_OBJECT_NAME.

Application migration details

When migrating to use the MQ API to do publish/subscribe, the code within any one application program must be consistent.

The application program must not contain a mixture of these deprecated APIs and the new MQ API options. An entire application suite, such as the combination of a subscribing application program and a publishing application program, does not all need to be migrated at the same time. Interaction between a publishing application program using the deprecated APIs and a subscribing application using the new MQ API is supported.

[Delete Publication - Version 7 replacement](#)

Replace the **Delete Publication** command by using the PCF **ClearTopic** command.

[Deregister publisher - Version 7 replacement](#)

Replace the **Deregister publisher** command with the MQCLOSE Message Queue Interface (MQI) call.

[Deregister subscriber - Version 7 replacement](#)

Replace the **Deregister subscriber** command with the MQCLOSE Message Queue Interface (MQI) call.

[Publish - Version 7 replacement](#)

Replace the Publish command with the MQPUT/MQPUT1 Message Queue Interface (MQI) calls.

[Register publisher - Version 7 replacement](#)

[Register subscriber - Version 7 replacement](#)

[Request Update - Version 7 replacement](#)

Parent topic: [WebSphere MQ version 6 publish/subscribe migration](#)

This build: January 26, 2011 10:56:41

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps19080_

7.2.1. Delete Publication - Version 7 replacement

Replace the **Delete Publication** command by using the PCF **ClearTopic** command.

The Delete Publication command message contains a number of parameters. This should be replaced by using the PCF ClearTopic command. This section details the equivalent options or fields in the PCF command message to show how you can migrate an application from using the Delete Publication command message to using the PCF ClearTopic command message.

Required parameters

MQPS_COMMAND with value MQPS_DELETE_PUBLICATION is implied when you use the ClearTopic command.

MQPS_TOPIC is provided in a field in the ClearTopic command message. If your application provided more than one MQPS_TOPIC in a single Delete Publication command message, the migrated application must now issue a separate ClearTopic call for each separate topic string.

Optional parameters

MQPS_DELETE_OPTIONS is replaced with an attribute of the ClearTopic command message.

For MQPS_STREAM_NAME see [Streams and topics](#).

Error codes

If your application checked for any of the following error codes, the equivalent MQRC error codes are shown in the following table:

Reason codes in NameValueString of the broker response message.	MQRC equivalent
MQRCCF_STREAM_ERROR	MQRC_UNKNOWN_OBJECT_NAME
MQRCCF_TOPIC_ERROR	MQRC_OBJECT_STRING_ERROR
MQRCCF_INCORRECT_STREAM	See Note 1
Notes:	
1. No equivalent because there is no need to provide the stream name twice, once in the command and once by putting it to the stream queue, so you cannot have a mismatch.	

Parent topic: [Publish/subscribe command messages migration](#)

 This build: January 26, 2011 10:56:39

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps19010_

7.2.2. Deregister publisher - Version 7 replacement

Replace the **Deregister publisher** command with the MQCLOSE Message Queue Interface (MQI) call.

The Deregister Publisher command message contains a number of parameters. You should replace it with the MQCLOSE verb. This section details the equivalent options or fields in the WebSphere® MQ API to show how to migrate an application from the Deregister Publisher command message to MQCLOSE.

A difference in behavior will be seen because a Register Publisher command could leave an application registered even when it was not connected, whereas the equivalent MQOPEN will only show a publisher's intent when the application is connected and keeps the handle from MQOPEN available. Even without issuing MQCLOSE, an application will be deregistered when the queue manager detects that the application's connection is lost.

Required parameters

MQPS_COMMAND with value MQPS_REGISTER_PUBLISHER is implied when closing a handle to a topic previously opened using MQOPEN with the MQOO_OUTPUT option.

Optional parameters

If your application provided a queue and queue manager name (either by using MQPS_Q_MGR_NAME and MQPS_Q_NAME in the command message, or from the ReplyToQ and ReplyToQMgr fields in MQMD of the command message), for the migrated application, these attributes are specified implicitly by providing the handle obtained when opening the topic.

MQPS_REGISTRATION_OPTIONS is replaced with options on the MQCLOSE call. See [MQCLOSE](#) for more details. Note that there are two ways you could have specified each of these options in your application, a string constant, MQPS_* or an integer constant, MQREGO_*. Both are replaced by the use of a single numeric constant.

String constant	Integer constant	MQCLOSE Options field constant
MQPS_CORREL_ID_AS_IDENTITY	MQREGO_CORREL_ID_AS_IDENTITY	See Queued publish/subscribe compatibility
MQPS_DEREGISTER_ALL	MQREGO_DEREGISTER_ALL	See Note 1
Notes:		
1. ➤ Because only one topic can be opened by the MQOPEN call, closing the handle closes that one topic. There is no need for an equivalent option. If many topics are opened, issuing MQDISC will close them all, saving the need to MQCLOSE each handle. ◀		

For MQPS_STREAM_NAME see [Streams and topics](#), although in this case, the stream name is specified implicitly by providing the handle obtained when opening the topic. MQPS_TOPIC is implied by the provision of the handle obtained when opening the topic.

Error codes

If your application checked for any of the following error codes, the equivalent MQRC error codes are shown in the following table:

--	--	--

Reason codes in NameValueString of the broker response message.	MQRC equivalent
MQRCCF_STREAM_ERROR	MQRC_HOBJ_ERROR (See note 1)
MQRCCF_TOPIC_ERROR	MQRC_HOBJ_ERROR (See note 1)
MQRCCF_NOT_REGISTERED	MQRC_HOBJ_ERROR (See note 1)
MQRCCF_Q_MGR_NAME_ERROR	MQRC_HOBJ_ERROR (See note 1)
MQRCCF_Q_NAME_ERROR	MQRC_HOBJ_ERROR (See note 1)
MQRCCF_DUPLICATE_IDENTITY	MQRC_HOBJ_ERROR (See note 1)
MQRCCF_UNKNOWN_STREAM	MQRC_HOBJ_ERROR (See note 1)
MQRCCF_REG_OPTIONS_ERROR	MQRC_OPTIONS_ERROR

Notes:

- This error code implies the same type of problem, but because, for the migrated application, all of these fields are specified implicitly by providing the handle obtained when opening the topic, this is the only equivalent error.

Parent topic: [Publish/subscribe command messages migration](#)

 This build: January 26, 2011 10:56:39

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps19020_

7.2.3. Deregister subscriber - Version 7 replacement

Replace the **Deregister subscriber** command with the `MQCLOSE` Message Queue Interface (MQI) call.

The Deregister Subscriber command message contains a number of parameters. This should be replaced by using the `MQCLOSE` verb. This section details the equivalent options or fields in the MQ API to show how an application would migrate from using the Deregister Subscriber command message to using `MQCLOSE`. If the Deregister Subscriber command message was used in a different program from that of the Register Subscriber command message, the application must now first use the `MQSUB` call with the `MQSO_RESUME` option to get a handle to the subscription, in order to deregister it.

Required parameters

`MQPS_COMMAND` with value `MQPS_DEREGISTER_SUBSCRIBER` is replaced by the use of the `MQCLOSE` verb with the option `MQCO_REMOVE_SUB`.

Optional parameters

If your application provided a queue and queue manager name (either by using `MQPS_Q_MGR_NAME` and `MQPS_Q_NAME` in the command message, or from the `ReplyToQ` and `ReplyToQMgr` fields in `MQMD` of the command message), after migrating the application, these attributes are specified implicitly by providing the handle obtained when subscribing to the topic.

`MQPS_REGISTRATION_OPTIONS` is replaced with Options on the `MQCLOSE` call. See [MQCLOSE](#) for more details. Note that there are two ways you could have specified each of these options in your application, a string constant, `MQPS_*` or an integer constant, `MQREGO_*`. Both are replaced by the use of a single numeric constant.

String constant	Integer constant	MQCLOSE Options field constant
<code>MQPS_CORREL_ID_AS_IDENTITY</code>	<code>MQREGO_CORREL_ID_AS_IDENTITY</code>	See Note 1
<code>MQPS_DEREGISTER_ALL</code>	<code>MQREGO_DEREGISTER_ALL</code>	See Note 2
<code>MQPS_FULL_RESPONSE</code>	<code>MQREGO_FULL_RESPONSE</code>	See Note 3
<code>MQPS_LEAVE_ONLY</code>	<code>MQREGO_LEAVE_ONLY</code>	See Note 4
<code>MQPS_VARIABLE_USER_ID</code>	<code>MQREGO_VARIABLE_USER_ID</code>	See Note 1

Notes:

- This option is implied by the provision of the handle obtained when subscribing to the topic.
- Because only one topic can be subscribed to by the `MQSUB` call, closing the handle closes that one topic. There is no need for an equivalent option. If many topics are opened, issuing `MQDISC` will close them all, saving the need to close each handle.
- Use of this option is implied in the use of the `MQSUB` verb. The fields returned in the response message are now populated in the `MQSD` structure. See [MQSUB](#) for more details. Because an `MQSUB` call must be made in order to obtain the handle to pass to the `MQCLOSE` call, this option is deprecated.
- Use of these options are deprecated.

For `MQPS_STREAM_NAME` see [Streams and topics](#), although in this case, the stream name is specified implicitly by providing the handle obtained when subscribing to the topic. `MQPS_SUBSCRIPTION_NAME` is replaced by the field in the `MQSD` called `SubName` and is therefore implied by the provision of the handle obtained when subscribing to the topic. `MQPS_TOPIC` is provided in a field in the `MQSD` called `ObjectString`, and is therefore implied by the provision of the handle obtained when subscribing to the topic.

See [MQSD](#) for more details

Error codes

If your application checked for any of the following error codes, the equivalent `MQRC` error codes are shown in the following table:

Reason codes in NameValueString of the broker response message.	MQRC equivalent
MQRCCF_STREAM_ERROR	MQRC_HOBJ_ERROR (See note 1)
MQRCCF_TOPIC_ERROR	MQRC_HOBJ_ERROR (See note 1)
MQRCCF_NOT_REGISTERED	MQRC_HOBJ_ERROR (See note 1)
MQRCCF_Q_MGR_NAME_ERROR	MQRC_HOBJ_ERROR (See note 1)
MQRCCF_Q_NAME_ERROR	MQRC_HOBJ_ERROR (See note 1)
MQRCCF_DUPLICATE_IDENTITY	MQRC_HOBJ_ERROR (See note 1)
MQRCCF_UNKNOWN_STREAM	MQRC_HOBJ_ERROR (See note 1)


MQRCCF_REG_OPTIONS_ERROR

MQRC_OPTIONS_ERROR

Notes:

1. This error code implies the same type of problem, but because all of these fields are now implied by the provision of the handle obtained when opening the topic, this is the only equivalent error.

Parent topic: [Publish/subscribe command messages migration](#)

 This build: January 26, 2011 10:56:40

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps19030_

7.2.4. Publish - Version 7 replacement

Replace the *Publish* command with the MQPUT/MQPUT1 Message Queue Interface (MQI) calls.

The *Publish* command message contains a number of parameters. The command message is replaced by using the MQPUT/MQPUT1 calls. This information details the equivalent options or fields in the MQI to show how an application would migrate from using the *Publish* command message to using MQPUT/MQPUT1.

Required parameters

MQPS_COMMAND with value MQPS_PUBLISH is implied when putting a message to an object handle that was obtained by opening a topic for MQOO_OUTPUT.

There is no equivalent to registering a publisher for new applications and registering it as an anonymous publisher. To achieve the same result of not passing a reply-to destination to subscribers, set the MQPMO option MQPMO_SUPPRESS_REPLYTO.

MQPS_TOPIC is provided in a field in the MQOD called ObjectString. See [MQOD - Object Descriptor](#) for more details. If your application provided more than one MQPS_TOPIC in a single Register Publisher command message, it must now issue a separate MQOPEN call for each separate topic string.

Optional parameters

MQPS_INTEGER_DATA can be replaced with the message property mqps.Sid or its synonym MQPubStrIntData.

MQPS_PUBLICATION_OPTIONS is replaced with the Options field in the MQPMO structure. See [MQPMO](#) for more details. There are two ways you could have specified each of these options in your application; a string constant, MQPS_* or an integer constant, MQPUBO_*. Both are replaced by the use of a single numeric constant.

String constant	Integer constant	Version 7 replacement
MQPS_CORREL_ID_ AS_IDENTITY	MQPUBO_CORREL_ID_ AS_IDENTITY	See Queued publish/subscribe compatibility for more information
MQPS_IS_RETAINED_ PUBLICATION	MQPUBO_IS_RETAINED_ PUBLICATION	This option is replaced with the message property mqps.Ret or its synonym MQIsRetained
MQPS_NO_ REGISTRATION	MQPUBO_NO_ REGISTRATION	This option is deprecated because publishers are no longer registered
MQPS_OTHER_ SUBSCRIBERS_ONLY	MQPUBO_OTHER_ SUBSCRIBERS_ONLY	This option is deprecated. If an application does not want to receive its own publications it can subscribe using the option MQPMO_NOT_OWN_SUBS on the MQPUT/MQPUT1 calls.
MQPS_RETAIN_ PUBLICATION	MQPUBO_RETAIN_ PUBLICATION	MQPMO_RETAIN

►MQPS_Q_MGR_NAME is replaced by the ReplyToQMGr in the MQMD of the publication. If the publisher specifies MQPMO_SUPPRESS_REPLYTO the ReplyToQMGr field is not set to the publishers queue manager name by the queue manager, otherwise it is.◀

MQPS_Q_NAME is replaced by the ReplyToQ in the MQMD of the publication. If the publisher does not set the ReplyToQ, it is not available.

MQPS_REGISTRATION_OPTIONS is replaced with Options in the MQPMO structure. See [MQPMO](#) for more details. These options are the same as for the Register Publisher command.

MQPS_SEQUENCE_NUMBER is replaced with the message property mqps.Seq or its synonym MQPubSeqNum.

For MQPS_STREAM_NAME, see [Streams and topics](#).

MQPS_STRING_DATA is replaced with the message property mqps.Sid or its synonym MQPubStrIntData.

Parent topic: [Publish/subscribe command messages migration](#)

 This build: January 26, 2011 10:56:40

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps19040_

7.2.5. Register publisher - Version 7 replacement

The Register Publisher command message contains a number of parameters. This should be replaced by using the MQOPEN verb. This section details the equivalent options or fields in the MQI to show how an application would migrate from using the Register Publisher command message to using MQOPEN. A difference in behaviour will be seen because a Register Publisher command could leave an application registered even when it was not connected, whereas MQOPEN will only show a publishers intent when the application is connected and keeps the handle from MQOPEN available.

Required parameters

MQPS_COMMAND with value MQPS_REGISTER_PUBLISHER is implied when opening a topic for MQOO_OUTPUT. If your application did not use Register Publisher, see the description of the Publish command message for details of remaining unregistered.

`MQPS_TOPIC` is provided in a field in the `MQOD` called `ObjectString`. See [MQOD - Object Descriptor](#) for more details. If your application provided more than one `MQPS_TOPIC` in a single Register Publisher command message, it must now issue a separate `MQOPEN` call for each separate topic string.

Optional parameters

If your application provided a queue and queue manager name (either by using `MQPS_Q_MGR_NAME` and `MQPS_Q_NAME` in the command message, or from the `ReplyToQ` and `ReplyToQMgr` fields in `MQMD` of the command message) in order for subscribing applications to be able to directly contact the publisher, then your application must now provide these details on each published message.

`MQPS_REGISTRATION_OPTIONS` is replaced with `Options` in the `MQPMO`. See [MQPMO](#) for more details. Note that there are two ways you could have specified each of these options in your application, a string constant, `MQPS_*` or an integer constant, `MQREGO_*`. Both are replaced by the use of a single numeric constant.


String constant	Integer constant	MQCLOSE Options field constant
<code>MQPS_ANONYMOUS</code>	<code>MQREGO_ANONYMOUS</code>	See Queued publish/subscribe compatibility
<code>MQPS_CORREL_ID_AS_IDENTITY</code>	<code>MQREGO_CORREL_ID_AS_IDENTITY</code>	See Queued publish/subscribe compatibility
<code>MQPS_DIRECT_REQUEST</code>	<code>MQREGO_DIRECT_REQUEST</code>	See note 1
<code>MQPS_LOCAL</code>	<code>MQREGO_LOCAL</code>	<code>MQPMO_SCOPE_QMGR</code>

Note:

- Use of this option (that provides a queue for reply messages from subscribers) is implied if either the `ReplyToQ` or `ReplyToQMgr` field is set in the `MQMD` of a publication message. If these fields are not provided, the queue manager sets the `ReplyToQMgr` to the queue manager local to the publisher, but does not set the `ReplyToQ`. To make the publication anonymous, the publishing application should use `MQPMO_SUPPRESS_REPLYTO`.

For `MQPS_STREAM_NAME` see [Streams and topics](#).

Parent topic: [Publish/subscribe command messages migration](#)

 This build: January 26, 2011 10:56:40

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
[ps19050_](#)

7.2.6. Register subscriber - Version 7 replacement

The Register Subscriber command message contains a number of parameters. This should be replaced by using the `MQSUB` verb. This section details the equivalent options or fields in the `MQ API` to show how an application would migrate from using the Register Subscriber command message to using `MQSUB`.

Required parameters

`MQPS_COMMAND` with value `MQPS_REGISTER_SUBSCRIBER` is replaced by the use of the `MQSUB` verb. If your application did not use Register Subscriber then the use of the `MQSUB` verb is not required for equivalent behaviour.

`MQPS_TOPIC` is provided in a field in the `MQSD` called `ObjectString`. See [MQSD](#) for more details. If your application provided more than one `MQPS_TOPIC` in a single Register Subscriber command message, it must now issue a separate `MQSUB` call for each separate topic string.

Optional parameters

If your application provided a non-local queue name and/or a queue manager name other than the one connected to (either by using `MQPS_Q_MGR_NAME` and `MQPS_Q_NAME` in the command message, or from the `ReplyToQ` and `ReplyToQMgr` fields in `MQMD` of the command message) then your application must now provide an object handle, which has been returned by a `MQOPEN` call for that queue, in the `Hobj` parameter of the `MQSUB` verb.

If your application provided the name of a queue local to the queue manager it connected to, it now has the option to request that the queue manager manage where the publications are sent. This can be done by using the `MQSO_MANAGED` option in the field in the `MQSD` called `Options`.

`MQPS_REGISTRATION_OPTIONS` is replaced with a field in the `MQSD` called `Options`. See [MQSD](#) for more details. Note that there are two ways you could have specified each of these options in your application, a string constant, `MQPS_*` or an integer constant, `MQREGO_*`. Both are replaced by the use of a single numeric constant.

String constant	Integer constant	MQCLOSE Options field constant
<code>MQPS_ADD_NAME</code>	<code>MQREGO_ADD_NAME</code>	See Note 1
<code>MQPS_ANONYMOUS</code>	<code>MQREGO_ANONYMOUS</code>	See Identity
<code>MQPS_CORREL_ID_AS_IDENTITY</code>	<code>MQREGO_CORREL_ID_AS_IDENTITY</code>	See Identity (also see Note 7)
<code>MQPS_DUPLICATES_OK</code>	<code>MQREGO_DUPLICATES_OK</code>	See Note 2
<code>MQPS_FULL_RESPONSE</code>	<code>MQREGO_FULL_RESPONSE</code>	See Note 3
<code>MQPS_INCLUDE_STREAM_NAME</code>	<code>MQREGO_INCLUDE_STREAM_NAME</code>	See Note 4
<code>MQPS_INFORM_IF_RETAINED</code>	<code>MQREGO_INFORM_IF_RETAINED</code>	See Note 5
<code>MQPS_JOIN_EXCLUSIVE</code>	<code>MQREGO_JOIN_EXCLUSIVE</code>	See Note 6
<code>MQPS_JOIN_SHARED</code>	<code>MQREGO_JOIN_SHARED</code>	See Note 6
<code>MQPS_LOCAL</code>	<code>MQREGO_LOCAL</code>	<code>MQSO_SCOPE_QMGR</code>
<code>MQPS_LOCKED</code>	<code>MQREGO_LOCKED</code>	See Note 6
<code>MQPS_NEW_</code>	<code>MQREGO_NEW_PUB</code>	<code>MQSO_NEW_</code>

PUBLICATIONS_ONLY	LICATIONS_ONLY	PUBLICATIONS_ONLY
MQPS_NO_ALTERATION	MQREGO_NO_ALTERATION	MQSO_RESUME
MQPS_NON_PERSISTENT	MQREGO_NON_PERSISTENT	MQSO_NON_PERSISTENT
MQPS_PERSISTENT	MQREGO_PERSISTENT	MQSO_PERSISTENT
MQPS_PERSISTENT_AS_PUBLISH	MQREGO_PERSISTENT_AS_PUBLISH	MQSO_PERSISTENT_AS_PUBLISH
MQPS_PERSISTENT_AS_Q	MQREFO_PERSISTENT_AS_Q	MQSO_PERSISTENT_AS_QUEUE_DEF
MQPS_PUBLISH_ON_REQUEST_ONLY	MQREGO_PUBLISH_ON_REQUEST_ONLY	MQSO_PUBLICATIONS_ON_REQUEST
MQPS_VARIABLE_USER_ID	MQREGO_VARIABLE_USER_ID	MQSO_ANY_USERID, (also see Note 7)

Notes:

1. Use of this option is deprecated because the only identity of a subscription is the SubName. See [Queued publish/subscribe compatibility](#).
2. Use of this option is deprecated because the queued interface has been removed.
3. Use of this option is implied in the use of the MQSUB verb. The fields returned in the response message are now populated in the MQSD structure. See [MQSD](#) for more details.
4. Use of this option is deprecated because stream names are part of the full topic name.
5. Use of this option is deprecated because the information about whether a publication is a retained publication or not is a message property that is always present.
6. Use of these options are deprecated.
7. This option is also relevant for Request Update.

For `MQPS_STREAM_NAME` see [Streams and topics](#), although in this case, the stream name is implied by the provision of the handle obtained when subscribing to the topic.

`MQPS_SUBSCRIPTION_NAME` is replaced by the field in the MQSD called SubName. See [MQSD](#) for more details.

`MQPS_SUBSCRIPTION_USER_DATA` is replaced by the field in the MQSD called SubUserData. See [MQSD](#) for more details.

Error codes

If your application checked for any of the following error codes, the equivalent MQRC error codes are shown in the following table:

Reason codes in NameValueString of the broker response message.	MQRC equivalent
MQRCCF_STREAM_ERROR	
MQRCCF_TOPIC_ERROR	
MQRCCF_Q_MGR_NAME_ERROR	
MQRCCF_Q_NAME_ERROR	
MQRCCF_DUPLICATE_IDENTITY	MQRC_IDENTITY_MISMATCH
MQRCCF_CORREL_ID_ERROR	
MQRCCF_NOT_AUTHORIZED	
MQRCCF_UNKNOWN_STREAM	
MQRCCF_REG_OPTIONS_ERROR	
MQRCCF_DUPLICATE_SUBSCRIPTION	►MQRC_SUB_ALREADY_EXISTS◀
MQRCCF_SUB_NAME_ERROR	
MQRCCF_SUB_IDENTITY_ERROR	See note 1
MQRCCF_SUBSCRIPTION_IN_USE	►MQRC_SUBSCRIPTION_IN_USE◀
MQRCCF_SUBSCRIPTION_LOCKED	See note 1
MQRCCF_ALREADY_JOINED	See note 1

Notes:

1. There is no equivalent because the use of SubIdentity is deprecated, because the only identity of a subscription is the SubName. See [Queued publish/subscribe compatibility](#).

Parent topic: [Publish/subscribe command messages migration](#)

 This build: January 26, 2011 10:56:41

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps19060_

7.2.7. Request Update - Version 7 replacement

The Request Update command message contains a number of parameters. This should be replaced by using the MQSUBRQ verb. This section details the equivalent options or fields in the MQ API to show how an application would migrate from using the Request Update command message to using MQSUBRQ.

Required parameters

`MQPS_COMMAND` with value `MQPS_REQUEST_UPDATE` is replaced by the use of the MQSUBRQ verb.

`MQPS_TOPIC` is implied by the use of the Hsub handle returned from the MQSUB call which is used as a parameter on the MQSUBRQ call.

Optional parameters

QMgrName, QName and StreamName are used in exactly the same way in Request Update command messages as they are in Register Subscriber command messages.


See [Register subscriber - Version 7 replacement](#) for details of how to migrate the use of these fields.

See [Register subscriber - Version 7 replacement](#) for details of how to migrate your application's use of MQPS_REGISTRATION_OPTIONS in this command message.

For MQPS_STREAM_NAME see [Streams and topics](#).

MQPS_SUBSCRIPTION_NAME is implied by the use of the Hsub handle returned from the MQSUB call which is used as a parameter on the MQSUBRQ call.

Parent topic: [Publish/subscribe command messages migration](#)

 This build: January 26, 2011 10:56:41

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps19070_



7.3. New queue manager attributes for publish/subscribe

Five attributes, formerly held in the queue manager configuration file, `qm.ini`, are now replaced by attributes of the queue manager.

In WebSphere MQ Version 6.0, the attributes listed in the following table were held in the Brokers stanza of the `qm.ini` file (or the registry in Windows). In WebSphere MQ Version 7.0, they are replaced by the queue manager attributes listed, which can be set by the MQSC command **ALTER QMGR** or the PCF command **Change Queue Manager**.

Table 1.

Attribute in qm.ini	Queue manager attribute (PCF parameter name)	MQSC parameter name
MaxMsgRetryCount	PubSubMaxMsgRetryCount	PSRTCNT
DiscardNonPersistentInputMsg	PubSubNPInputMsg	PSNPMMSG
DLQNonPersistentResponse	PubSubNPRResponse	PSNPRES
DiscardNonPersistentResponse		
SyncPointIfPersistent	PubSubSyncPoint	PSSYNCP

Parent topic: [WebSphere MQ version 6 publish/subscribe migration](#)

 This build: January 26, 2011 10:57:09

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps23110_



7.4. WebSphere MQ publish/subscribe topology migration

Migrate a version 6 publish/subscribe hierarchy to a version 7 hierarchy, or convert it to a publish/subscribe cluster, before running existing version 6 publish/subscribe applications on version 7.

Publish/subscribe clusters and hierarchies offer different ways to connect queue managers together in a publish/subscribe topology. Their characteristics are different. Weigh up the merits of simply migrating a version 6 publish/subscribe hierarchy to a version 7 hierarchy or whether to convert the hierarchy to a cluster. You might consider breaking a hierarchy up into multiple clusters, perhaps joined in a hierarchy. You might also consider creating a publish/subscribe cluster using new queue managers, and then running the version 6 publish/subscribe applications on the new cluster, rather than converting the hierarchy to a cluster.

If your principle objective is to upgrade to version 7, with the least change to operational behavior, use **strmqbrk** to migrate from a version 6 publish/subscribe hierarchy to a version 7 hierarchy. See [Migrate a WebSphere® MQ version 6 publish/subscribe hierarchy to a version 7 hierarchy](#).

After running **strmqbrk**, you can continue to use version 6 publish/subscribe applications, write new version 7 publish/subscribe programs, and later decide to convert from using the hierarchy to using clusters.

New version 7 publish/subscribe applications are more likely to use clusters than hierarchies. What advantages do clusters have over hierarchies that might prompt you to convert from a hierarchy to a cluster?

Robustness

The failure of a queue manager can affect the transmission of publications and subscriptions between other queue managers in the hierarchy. In a cluster every queue manager is connected to every other queue manager. A failure in one queue manager does not affect the connection between two other queue managers.

Simpler channel administration

A hierarchy uses manually configured channel connections between queue managers in the hierarchy. You need to maintain these connections, adding and removing channels as queue managers are added and removed from the hierarchy. In a publish/subscribe cluster, queue managers are connected by automatically maintained cluster connections.

Ease of use

Cluster publication topics, subscriptions, and their attributes are replicated to every member of a cluster. You can display and modify cluster topics and subscriptions attributes using the WebSphere MQ Explorer. Your changes are replicated to other members of the cluster.

Consistency

In general, you must not connect the same queue managers together with hierarchies and clusters. If you have decided that new publish/subscribe applications are to use publish/subscribe clusters, avoid queue managers being part of both a hierarchy and a cluster. You need to convert existing hierarchies to clusters.

The principle reasons for continuing to use a hierarchy, despite the advantages of publish/subscribe clusters, are twofold:

1. **strmqbrk** performs the migration to a version 7 hierarchy automatically. To convert the hierarchy to a cluster, you must do several manual tasks in addition to running **strmqbrk**.

2. You do not need to complete the migration of the whole hierarchy to version 7, before resuming your publish/subscribe applications using a hierarchy. In contrast, the whole hierarchy must be converted to a cluster before resuming your publish/subscribe applications using a cluster.

Treat the conversion of a version 6 publish/subscribe hierarchy to using clusters as a two-stage process. First, migrate all the queue managers in the hierarchy to use a version 7 publish/subscribe hierarchy, and then convert the version 7 hierarchy to a cluster.

[Migrate a WebSphere MQ version 6 publish/subscribe hierarchy to a version 7 hierarchy](#)

Migrate queue managers in a version 6 publish/subscribe hierarchy to version 7, a queue manager, or server at a time.

[Convert a WebSphere MQ version 6 publish/subscribe hierarchy to a version 7 publish/subscribe cluster](#)

Convert a version 6 publish/subscribe hierarchy to a cluster by migrating it to a version 7 hierarchy, and then converting to a cluster. Conversion to a cluster requires manual steps and the whole hierarchy needs to be converted at the same time.


Parent topic: [WebSphere MQ version 6 publish/subscribe migration](#)

Related concepts

[Publish/subscribe topologies](#)

Related reference

[strmqbrk \(Migrate WebSphere MQ Version 6.0 broker to Version 7.0\)](#)

 This build: January 26, 2011 10:56:39

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps17000_



7.4.1. Migrate a WebSphere MQ version 6 publish/subscribe hierarchy to a version 7 hierarchy

Migrate queue managers in a version 6 publish/subscribe hierarchy to version 7, a queue manager, or server at a time.

Before you begin

You can migrate a version 6 publish/subscribe hierarchy to version 7 one server at a time. You do not need to migrate the whole hierarchy at once. The hierarchy continues to work with some queue managers at version 6, and some at version 7.

Consider how you are going to upgrade multiple queue managers on one server to version 7. Do you need to upgrade the queue managers one at a time, or can you follow the simpler course and upgrade them altogether? All the queue managers on a server share the same WebSphere® MQ libraries and therefore are at the same release level. You cannot upgrade some queue managers on the same server to version 7, leaving others on version 6.

Consider these two alternatives, if you are running multiple queue managers on the same server.

1. The simpler approach to migrating publish/subscribe to version 7 is to upgrade WebSphere MQ on the server to version 7.
Migrate the publish/subscribe broker for each queue manager on the server to version 7 publish/subscribe, one queue manager at a time.
2. If you want to upgrade the queue managers to version 7 one at a time, you must perform a "side-by-side" upgrade rather than an "in-place" upgrade.
In a side-by-side upgrade:
 - a. Install WebSphere MQ version 7 on a new server.
 - b. Transfer a queue manager to the new server.
 - c. Migrate its version 6 broker to version 7.
 - d. Repeat, one queue manager at a time, until all the queue managers are running on the new server, and none are left on the old server.

About this task

In this task, we assume that you have already upgraded a queue manager to version 7. You now want to migrate the publish/subscribe broker to version 7.

You might be migrating the first publish/subscribe broker in the hierarchy, or the last. Follow the same procedure. At any point in the conversion, of all the publish/subscribe brokers in the hierarchy, the hierarchy, and your applications continue to work.

Procedure

1. Start the queue manager.

```
STRMQM -m QMgrName
```
2. Run the **strmqbrk** command to migrate all publish/subscribe configuration data for the queue manager to WebSphere MQ Version 7

```
strmqbrk -m QMgrName
```
3. Check the migration log to verify that the migration was successful.
The migration log is in the queue manager data directory; the default locations are `/var/mqm/qmgrs/QMgrName/psmigr.log` on UNIX or `C:\Program Files\IBM\WebSphereMQ\qmgrs\QMgrName\psmigr.log` on Windows.
4. **strmqbrk** starts queued publish/subscribe when it completes, by switching the queue manager attribute **PSMODE** from its default value of `COMPAT` to `ENABLED`.

Parent topic: [WebSphere MQ publish/subscribe topology migration](#)

Related reference

[strmqbrk \(Migrate WebSphere MQ Version 6.0 broker to Version 7.0\)](#)

 This build: January 26, 2011 10:56:48

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps21090_



7.4.2. Convert a WebSphere MQ version 6 publish/subscribe hierarchy to a version 7 publish/subscribe cluster

Convert a version 6 publish/subscribe hierarchy to a cluster by migrating it to a version 7 hierarchy, and then converting to a cluster. Conversion to a cluster requires manual steps and the whole hierarchy needs to be converted at the same time.

Before you begin

Advise users that the publish/subscribe system is not available while conversion takes place. The conversion takes time. All the queue managers in the hierarchy have to be converted together. Publications that are being processed are not lost, but all running publish/subscribe applications that use the hierarchy must be stopped. There are manual configuration tasks to be performed, some of which are not automatic transformations, before applications can be restarted.

To minimize disruption, plan, prepare, and test scripts to do the manual steps, and review which topic objects are to be clustered with the publish/subscribe application owners.

About this task

The task is organized to minimize disruption. In step, [1](#) only one queue manager is impacted at a time. Step [2](#) can be done in advance, without disrupting applications. Stop applications between steps [3](#) to [6](#) inclusive.

In step [6](#), you need to define cluster topic objects on the cluster topic host. There are administrative advantages of using cluster topic objects, but that is not the reason for defining cluster topic objects during the conversion task. The principle reason for defining cluster topic objects as part of the conversion task to restore existing publish/subscribe applications to a fully working state.

Existing publish/subscribe applications that use a hierarchy do not work in a cluster, unless you identify and define the correct cluster topics. In hierarchical publish/subscribe, subscriptions are propagated to every queue manager in the hierarchy, as long as the subscription is registered with **SUBSCOPE**(ALL). In clustered publish/subscribe no subscriptions are propagated to the cluster, unless the subscription resolves to a topic with the attribute **CLUSTER** (clusterName) set, and **SUBSCOPE**(ALL).

Part of the migration procedure is to identify or create topic objects that are to be given the cluster attribute. Application knowledge is required to choose the correct topics to associate with cluster topic objects. Here is some guidance to help you in choosing which topics to cluster.

- If your version 6 publish/subscribe applications used streams, part of the migration to version 7 is to create topic objects corresponding to the streams. These topic objects are prime candidates to be made into cluster topic objects.
The migration process creates one topic object for each stream that is defined on a version 6 queue manager. Thus, depending on what streams are defined on what queue managers, you might find different topic objects created on different queue managers.
Unlike any other clustered topic object, you must define the topic objects that correspond to streams on every queue manager in the cluster. Do not follow the typical procedure, and define the topic object only on the cluster topic host. The queued publish/subscribe daemon requires a locally defined topic object corresponding to the name of the stream in `SYSTEM.QPUBSUB.QUEUE.NAMELIST`. You must set the **CLUSTER** attribute on each of the automatically defined local topic objects. The cluster attribute causes subscriptions to topics that resolve to a stream topic object to be propagated to other members of the cluster. The attributes of the topic object are resolved to the cluster topic object that is locally defined, and not to the latest cluster topic object to be defined.
- Identify or create as few topic objects to be clustered as the application design requires. Allow other topics to inherit from these topic objects. To this end, look for topics near the root of the topic trees of your publish/subscribe applications, and make them clustered.
- Identify any version 6 publish/subscribe applications that are going to be hard to migrate from a hierarchy to a cluster.
Applications that use the default stream and have a flat topic space, or do not have topics you can clearly identify as the root of their topic trees, are hard to migrate. It might be difficult to know how to define cluster topic objects associated with the topics an application uses. Do not, except as a last resort, set the cluster attribute on `SYSTEM.BASE.TOPIC` as a way of causing all topics to inherit the cluster attribute. It might be worthwhile to convert your existing version 6 publish/subscribe applications that use the default stream, to use named streams. Then each stream converts to a defined topic object that you can cluster.
Note: If you set the cluster attribute on `SYSTEM.BASE.TOPIC`, you do not need to set it on any other topics that inherit from `SYSTEM.BASE.TOPIC`.

Procedure

1. Migrate all the queue managers from the version 6 hierarchy to version 7.
 - a. Carry out the procedure described in [Migrate a WebSphere® MQ version 6 publish/subscribe hierarchy to a version 7 hierarchy](#), to convert all the queue managers in the hierarchy from version 6 to version 7.
2. Create a cluster and add all the queue managers in the hierarchy to the cluster.
 - a. Create a cluster or nominate an existing cluster, which does not need to be an existing publish/subscribe cluster.
Use WebSphere MQ Script commands (MQSC), or any other type of administration command or utility that is available on your platform, such as the WebSphere MQ Explorer. These methods are described in [Using WebSphere MQ commands with clusters](#)
 - b. Ensure that each queue manager is in the cluster by using the MQSC command `DISPLAY CLUSQMGR(*)`, described in [WebSphere MQ Script \(MQSC\) Command Reference](#). If a queue manager is not in the cluster, add it. For more information, refer to [Using WebSphere MQ commands with clusters](#)
3. Stop publish/subscribe applications, allowing current work to complete.
 - a. Stop all publish/subscribe publishers from putting new work into the system.
Do not stop new work by disabling the input stream queues - the input streams are needed to process publications that remain in the hierarchy. You must stop the applications themselves.
 - b. Switch off queued publish/subscribe on all the queue managers in the hierarchy. Leave version 7 integrated publish/subscribe running. Run the following MQSC command on all the queue managers in the hierarchy.

```
ALTER QMGR PSMODE (COMPAT)
```
 - c. Wait for the transmission queues and channels used to connect queue managers in the hierarchy to finish processing publications that are already in the system.
When no more publications are left in transmission queues and channels, all the publications have reached their destination queue manager. The queue manager delivers the publications to subscribers when queued publish/subscribe is re-enabled.
4. Delete all the channels and transmission queues used to connect queue managers in the hierarchy.
 - a. Stop all the channels used to connect the queue managers in the hierarchy.
Run the following MQSC command on all the queue managers in the hierarchy.

```
STOP CHANNEL (SenderChanName) MODE (QUIESCE)
```
 - b. Delete all the channels used to connect the queue managers in the hierarchy.
Run the following MQSC command on all the queue managers in the hierarchy.

```
DELETE CHANNEL (SenderChanName)
DELETE CHANNEL (ReceiverChanName)
```
 - c. Delete the transmission queues associated with the channels that were deleted.
Run the following MQSC command on all the queue managers in the hierarchy.

```
DELETE QLOCAL (xmitQName)
```
5. Delete the queue manager hierarchy.

- a. Enable queued publish/subscribe on each queue manager in the hierarchy using the MQSC command:

```
ALTER QMGR PSMODE(ENABLE)
```

- b. Run the following MQSC command on each queue manager in the hierarchy, except the uppermost parent in the hierarchy which has no parent.

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

Alternatively, on I5/OS run the following commands to remove queue managers from the hierarchy.

- i. Run **WRKMQMPS PUBSUBNAME**(parentQmgrName) to display the hierarchy.
 - ii. Use **option 4=Remove** to remove the parent from the hierarchy.
 - iii. Use **option 34=Work with Pub/Sub** to move down the subhierarchy
 - iv. Repeat options **4** and **34** until there are no child queue managers displayed.
- c. Before proceeding to the next step confirm that all the hierarchical relationships have been canceled. Run the following MQSC command on each queue manager.

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

6. Set the **CLUSTER** attribute on the topic objects that you have decided to make clustered.

- a. If you need to create cluster topic objects, define them on the cluster topic host. Define cluster topic objects on only one queue manager.
- b. If you are setting the cluster attribute on existing topic objects, created by **strmqbrk**, set the cluster attribute on the topic object defined on the cluster topic host. Delete the topic object from the other queue managers. Multiple definitions of a cluster topic object on multiple queue managers in a cluster can lead to problems.
- c. Review whether to delete any topic objects that were created by **strmqbrk** that are not clustered. Subscriptions that inherit from these topic objects do not get propagated to other queue managers in the cluster.

7. Restart publish/subscribe applications.

Parent topic: [WebSphere MQ publish/subscribe topology migration](#)

Related information

[Creating a new cluster](#)

 This build: January 26, 2011 10:56:49

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps21110_



7.5. Differences from WebSphere MQ Version 6 publish/subscribe

Queued publish/subscribe programs and queued broker administration in version 7 differ from that in version 6. The differences in program behavior are slight; the administration differences are more extensive. Many version 6 programs coexist and interoperate with version 7, without change.

There are many changes in the implementation of version 7 queued publish/subscribe from the implementation in version 6. Queued publish/subscribe in version 7 uses integrated publish/subscribe. Only the changes that might affect your applications or administration procedures are described here. The use of queued publish/subscribe is deprecated in version 7.

Administration

The version 6 WebSphere® MQ publish/subscribe broker is integrated into version 7 publish/subscribe. The commands you used to control a version 6 publish/subscribe broker are obsolete, and replace by commands to control version 7 publish/subscribe. The new commands, as they relate to controlling queued publish/subscribe, are described in [Controlling queued publish/subscribe](#). [Table 1. Broker command differences](#) relates the old and new commands.

Table 1. WebSphere Message Broker command differences

Operation	WebSphere MQ Version 6	WebSphere MQ Version 7
Remove broker from hierarchy	clrmqbrk	See Disconnect a queue manager from a broker hierarchy for instructions how to disconnect a version 7 queue manager from a hierarchy.
Delete broker	dlmqbrk	There is no publish/subscribe broker in version 7. The dlmqbrk command removes version 6 broker resources after running the command strmqbrk to migrate the version 6 broker to version 7 publish/subscribe.
Display broker	dspmqbrk	Use the runmqsc command DISPLAY PUBSUB to display publish/subscribe status.
Stop broker	endmqbrk	See Stopping queued publish/subscribe for instructions how to stop queued publish/subscribe in version 7.
Migrate broker to WebSphere Message Broker	miqmqrk	Run the migmqbrk command on a version 6 queue manager. Once you have upgraded to Version 7 there is <i>no migration</i> from version 7 publish/subscribe to the version 6.1 WebSphere Message Broker publish/subscribe broker.
Start broker	strmqbrk	In version 7, the strmqbrk command migrates the version 6 broker to version 7. See Starting queued publish/subscribe for instructions how to start queued publish/subscribe in version 7.

Configuration data

The broker stanza parameters in WebSphere MQ Version 6 publish/subscribe are described in [Broker configuration stanza](#). They are replaced with queue manager attributes, which are described in [New queue manager attributes for publish/subscribe](#).

Local and global publications and subscriptions

The local or global publication scope is set by defining the scope of a subscription or a publication to `QMGR` or `ALL`. With this flag you can control, on a publication, whether it is propagated locally or globally. On a subscription, you can control whether it has an interest in publications made locally or globally. The flow of publications you observe is a result of the interplay of both these flags.

There has been a subtle change in the behavior between version 6 and version 7 if the publication and subscription flags are set to different options.

- In version 7: if you specify that a subscription is only for local publications, the subscription receives any publications that are created on the local queue manager.
- In version 6: if you set the same subscription option, you would receive only publications that have been created on this queue manager and are flagged for local publication only. Thus no publications are received if the publication scope is global, even if the publisher is local.
- In version 7: if you specify that a publication is for global distribution, a subscription to the publication on any queue manager is eligible to receive the publication. Successful subscriptions include a subscription for local publications made on the same queue manager as the publisher.

- In version 6: setting the same option, only subscribers that are eligible to receive global publications receive the publication. Thus no subscribers on the same queue manager as the publisher receive the publication, if their subscription has local scope.

The operation of publication and subscription scope in version 6 and version 7 is described by [Table 2](#).

Notice that if you specify opposite subscription scopes for the publisher and subscriber, it makes no difference which scope the publication or subscription is given. Either way round, in either version, the set of publications received is the same. In version 6 no publications are received, and in version 7 local publications are received.

Table 2. Version 6 and version 7 publication and subscription scope truth table

Publisher Subscriber	QMGR		ALL	
	Version 6	Version 7	Version 6	Version 7
QMGR	No change: publications go to local subscribers		No publications are delivered	Only subscribers local to the publisher receive publications
ALL	No publications are delivered	Publications go to subscribers local to the publisher	No change: publications go to all subscribers	

Interoperation with WebSphere Message Broker

You cannot connect the WebSphere Message Broker V6.1 publish/subscribe broker to WebSphere MQ Version 7 using clusters or broker hierarchies. No messages flow between the broker and WebSphere MQ, because no cluster or hierarchy connection can be established. Subscriptions and publications are not passed directly between the broker publication node and the WebSphere MQ queue manager.

Metatopics

Metatopics are a special set of topics recognized by the WebSphere MQ Version 6 broker. See [Metatopics](#).

Metatopics are not provided by WebSphere MQ Version 7. Instead you can inquire on the list of topic names, and on individual topics and subscriptions.

If you send a subscription to a metatopic, the subscription is ignored.

Register Publisher and Deregister Publisher commands

The [Register Publisher](#) and [Deregister Publisher](#) commands do nothing in version 7, except return a successful response message to a request. Your publisher program is not affected by the change.

Publish/subscribe exit

The WebSphere MQ Version 6 publish/subscribe has an exit for customizing and routing publications, which is described in [WebSphere Message Broker exit](#). The exit is renamed the Publish exit and is available in WebSphere MQ Version 7.0.1 onwards; see [Publish exit](#). It was not available in WebSphere MQ V7.0.0.x.

The publish/subscribe exit capability is largely replaced by using subscription levels. Using the `MQSD sublevel` field, an intermediate subscriber can intercept publications to customize or block them, before they arrive at the ultimate subscribers, see [Intercepting publications](#).

The function provided by intercepting publications using the `MQSD sublevel` field, and using the Publish exit is similar. The main differences to be aware of are twofold:

1. If you use the subscription level mechanism, the intercepting application is a regular publish\subscribe application. You subscribe to the publications you intend to intercept. Publications are delivered to the intercepting application, which then must publish them again if they are to reach their intended destination.
The Publish exit is written as an exit program and is called by the queue manager.
2. Intercept all publications from a publisher by using the subscription level mechanism. Run the intercepting application on the queue manager that the publisher is connected to.
Intercept a publication just before it is delivered to a subscriber by using a Publish exit. Configure the exit on the queue manager the subscriber is connected to.

Streams

There are significant changes in how streams are implemented in WebSphere MQ version 7.

Streams are not supported by the integrated publish/subscribe MQI interface. However, version 6 queued publish/subscribe applications using streams interoperate without change with version 7 integrated publish/subscribe applications. Streams are mapped to the topic space in version 7, see [Streams and topics](#)

Altering fields in a subscription

In version 7 it is not possible to alter the topic, destination queue, and subscription name of a subscription once the fields are defined. In version 6, these fields are only alterable for a subscription with a subscription name.

Multi-topic publications and subscriptions

A single command message can publish a publication to multiple topics. Suppose a subscriber subscribes to both topics. A version 6 subscriber receives one publication. A version 7 subscriber receives two copies of the publication.

Variable user ID and traditional identity

Subscriptions are tied to a particular user unless you specify that other users are able to alter the subscription. Other users are able to alter the subscription, subject to access control checks, by setting the variable user ID field.

In version 6, subscriptions are tied to a traditional identity. The traditional identity is a combination of queue name, queue manager name, and optional correlation identifier. Only the user who created a subscription to a traditional identity is permitted to modify it, unless you specify that other users can modify the subscription. Subscriptions created by another user, which result in publications to the same combination of queue name, queue manager name, and optional correlation identifier, might not succeed. They only succeed if the original subscription has allowed other users to modify their original subscription to this traditional identity.

In version 7, subscriptions are not tied to a traditional identity. Any subscription can result in publications being sent to destinations with any combination of queue, queue manager, and correlation identifier. Modifications to a subscription are still restricted to the user that created the subscription unless the `MQSUB MQSO_ANY_USERID` subscription option is set. Setting the option to `MQSO_FIXED_USERID` does not prevent other subscriptions delivering publications to the same destination.

Wildcards


The wildcard schemes used by version 6 and version 7 publish subscribe are different.

The earlier WebSphere MQ Version 6 publish/subscribe scheme, uses the characters described in [Character-based wildcard scheme](#).


To use the version 6 wildcard scheme when subscribing with a version 7 subscriber, set the `MQSO_WILDCARD_CHAR` option.

For more information about how WebSphere Message Broker handles WebSphere MQ Version 6 wildcards, see [Wildcard characters](#).

Parent topic: [WebSphere MQ version 6 publish/subscribe migration](#)

 This build: January 26, 2011 10:57:13

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps29080_

7.6. Using publish/subscribe with WebSphere MQ classes for JMS

► Existing WebSphere® MQ classes for JMS applications run unchanged after you upgrade your queue manager to WebSphere MQ V7.0. In some circumstances, you must specify whether WebSphere MQ classes for JMS uses WebSphere MQ Version 6.0 or Version 7.0 publish/subscribe function. ◀

► The circumstances in which you must specify whether WebSphere MQ classes for JMS uses WebSphere MQ Version 6.0 or Version 7.0 publish/subscribe function are described later in this topic. The advantages of using WebSphere MQ Version 7.0 publish/subscribe function, compared with WebSphere MQ Version 6.0 Publish/Subscribe, WebSphere Event Broker, or WebSphere Message Broker, are introduced in [What is new in WebSphere MQ Version 7.0?](#) ◀

WebSphere MQ messaging provider

The WebSphere MQ messaging provider has two modes of operation:

- WebSphere MQ messaging provider normal mode
- WebSphere MQ messaging provider migration mode

The WebSphere MQ messaging provider normal mode uses all the features of the WebSphere MQ Version 7.0 queue managers to implement JMS. This mode is used only to connect to a WebSphere MQ queue manager and can connect to WebSphere MQ Version 7.0 queue managers in either client or bindings mode. The WebSphere MQ messaging provider normal mode is optimized to use the new WebSphere MQ Version 7.0 function.

The WebSphere MQ messaging provider migration mode is based on WebSphere MQ Version 6.0 function and uses only features that were available in the WebSphere MQ Version 6.0 queue manager to implement JMS. You can connect to a WebSphere MQ Version 7.0 queue manager using WebSphere MQ messaging provider migration mode but you cannot use any of the Version 7.0 optimizations. This mode allows connections to either of the following queue manager versions:

- WebSphere MQ Version 7.0 queue manager in bindings or client mode, but this mode uses only those features that were available to a WebSphere MQ Version 6.0 queue manager
- WebSphere MQ Version 6.0 or earlier queue manager in client mode

If you want to connect to WebSphere Event Broker or WebSphere Message Broker using either WebSphere MQ Enterprise Transport or WebSphere MQ Real-Time Transport, use the WebSphere MQ messaging provider migration mode. If you use WebSphere MQ Real-Time Transport, the WebSphere MQ messaging provider migration mode is automatically selected, because you have explicitly selected properties in the connection factory object. Connection to WebSphere Event Broker or WebSphere Message Broker using the WebSphere MQ Enterprise Transport follows the general rules for mode selection described in [Rules for selecting the WebSphere MQ messaging provider mode](#).

Rules for selecting the WebSphere MQ messaging provider mode

If you are not using WebSphere MQ Real-Time Transport, the mode of operation used is determined primarily by the `PROVIDERVERSION` property of the connection factory. If you cannot change the connection factory you are using, you can use a client configuration property called `com.ibm.msg.client.wmq.overrideProviderVersion`, which overrides any setting on the connection factory. This override applies to all connection factories in the JVM but the actual connection factory objects are not modified. You can set `PROVIDERVERSION` to three possible values: 7, 6, or `unspecified`:

PROVIDERVERSION=7

Uses the WebSphere MQ messaging provider normal mode.

If you set `PROVIDERVERSION` to 7 only the WebSphere MQ messaging provider normal mode of operation is available. If the queue manager that is connected to as a result of the other settings in the connection factory is not a Version 7.0 queue manager, the `createConnection()` method fails with an exception.

The WebSphere MQ messaging provider normal mode uses the sharing conversations feature, and the number of conversations that can be shared is controlled by the `SHARECNV()` property on the server connection channel. If this property is set to 0, you cannot use WebSphere MQ messaging provider normal mode and the `createConnection()` method fails with an exception.

PROVIDERVERSION=6

Uses the WebSphere MQ messaging provider migration mode.

The WebSphere MQ classes for JMS use the features and algorithms supplied with WebSphere MQ Version 6.0. If you want to connect to WebSphere Event Broker or WebSphere Message Broker using WebSphere MQ Enterprise Transport, you must use this mode. You can connect to a WebSphere MQ Version 7.0 queue manager using this mode, but none of the new features of a Version 7.0 queue manager are used, for example, read ahead or streaming.

PROVIDERVERSION=unspecified

This is the default value and the actual text is "unspecified".

A connection factory that was created with a previous version of WebSphere MQ classes for JMS in JNDI takes this value when the connection factory is used with V7.0 of WebSphere MQ classes for JMS. The following algorithm is used to determine which mode of operation is used. This algorithm is used when the `createConnection()` method is called and uses other aspects of the connection factory to determine if WebSphere MQ messaging provider normal mode or WebSphere MQ messaging provider migration mode is required.

- Firstly, an attempt to use WebSphere MQ messaging provider normal mode is made.
- If the queue manager connected is not WebSphere MQ Version 7.0, the connection is closed and WebSphere MQ messaging provider migration mode is used instead.
- If the `SHARECNV()` property on the server connection channel is set to 0, the connection is closed and WebSphere MQ messaging provider migration mode is used instead.
- If `BROKERVER` is set to `V1` or `unspecified`, WebSphere MQ messaging provider normal mode continues to be used, and therefore any publish/subscribe operations use the new WebSphere MQ V7.0 features.
If WebSphere Event Broker or WebSphere Message Broker are used in compatibility mode (and you want to use Version 6.0 publish/subscribe function rather than the WebSphere MQ Version 7 publish/subscribe function), set `PROVIDERVERSION` to 6 to ensure WebSphere MQ messaging provider migration mode is used.
- If `BROKERVER` is set to `V2` and `BROKERQMR` is nonblank, this means `BROKERQMR` has been explicitly changed from the default, so the

assumption is the connection factory is intended for use with WebSphere Event Broker or WebSphere Message Broker and WebSphere MQ Enterprise Transport. Therefore WebSphere MQ messaging provider migration mode is used.

- If `BROKERVER` is set to `V2`, `BROKERQMGR` is blank, the specified `BROKERCONQ` command queue exists and can be opened for output (that is, `MQOPEN` for output succeeds), and `PSMODE` on the queue manager is set to `COMPAT` or `DISABLED`, WebSphere MQ messaging provider migration mode is used.

You can find further guidance about using `PROVIDERVERSION` in [When to use PROVIDERVERSION](#).

When to use PROVIDERVERSION

There are two scenarios where you cannot use the algorithm described in [Rules for selecting the WebSphere MQ messaging provider mode](#); consider using `PROVIDERVERSION` in these scenarios.

1. If WebSphere Event Broker or WebSphere Message Broker is in compatibility mode, you must specify `PROVIDERVERSION` for them to work correctly.
2. If you are using WebSphere Application Server Version 6.0.1, Version 6.0.2, or Version 6.1, you define connection factories using the WebSphere Application Server administrative console.

In WebSphere Application Server the default value of the `BROKERVER` property on a connection factory is `V2`. The default `BROKERVER` property for connection factories created by using `JMSAdmin` or WebSphere MQ Explorer is `V1`. This property is now "unspecified" in WebSphere MQ Version 7.0.

If `BROKERVER` is set to `V2` (either because it was created by WebSphere Application Server or the connection factory has been used for publish/subscribe before) and the existing queue manager has a `BROKERCONQ` defined (because it has been used for publish/subscribe messaging before), the WebSphere MQ messaging provider migration mode is used.

However, if you want the application to use peer-to-peer communication and the application is using an existing queue manager that has previously been used for publish/subscribe, and has a connection factory with `BROKERVER` set to 2 (which is the default if the connection factory was created in WebSphere Application Server), the WebSphere MQ messaging provider migration mode is used. Using WebSphere MQ messaging provider migration mode in this case is unnecessary; use WebSphere MQ messaging provider normal mode instead. You can use one of the following methods to work around this:

- Set `BROKERVER` to `V1` or *unspecified*. This is dependent on your application.
- Set `PROVIDERVERSION` to 7; this is a custom property in WebSphere Application Server Version 6.1. The option to set custom properties in WebSphere Application Server Version 6.1 and later is not currently documented in the WebSphere Application Server Information Center. Alternatively, use the client configuration property (see [Rules for selecting the WebSphere MQ messaging provider mode](#) for details about how you can specify this system property for all environments), or modify the queue manager connected so it does not have the `BROKERCONQ`, or make the queue unusable.


Subscription name migration on the JMS client

On the JMS client, if the `ConnectionFactory` property `brokerPubQ` is not the default, WebSphere MQ adds the stream name to the subscription name.

Parent topic: [WebSphere MQ version 6 publish/subscribe migration](#)

 This build: January 26, 2011 10:56:15

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
[mi11010_](#)


7.6.1. Subscription name migration on the JMS client

On the JMS client, if the `ConnectionFactory` property `brokerPubQ` is not the default, WebSphere® MQ adds the stream name to the subscription name.


In WebSphere MQ Version 6.0, a subscription name needed to be unique only within the stream and not across the queue manager. In WebSphere MQ Version 7.0, a subscription name must be unique across the queue manager. Therefore to migrate WebSphere MQ Version 6.0 durable subscriptions to WebSphere MQ Version 7.0, the subscription names must be unique. WebSphere MQ does this when it migrates the queue manager, by appending the stream name to the existing subscription name. For any existing durable subscription that uses a stream other than the default of "SYSTEM.BROKER.DEFAULT.STREAM" the migration process appends the stream name to the subscription name.

On the JMS client, if the `ConnectionFactory` property `brokerPubQ` is not the default, it is assumed that a WebSphere MQ Version 6.0 durable subscription is being resumed, and WebSphere MQ Version 7.0 appends the stream name to match the action of the migration process. Subscription names that use the default stream are migrated across with the subscription name unchanged.

Parent topic: [Using publish/subscribe with WebSphere MQ classes for JMS](#)

 This build: January 26, 2011 10:56:15

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
[mi12170_](#)

7.7. Migration implications of mapping an alias queue to a topic object

WebSphere MQ Version 7.0 introduces an extension to the alias queue object that allows an alias queue to be mapped to a topic object.

The new `TARGETTYPE` attribute allows you to specify that a queue alias resolves to a queue or a topic. The `TARGQ` attribute, defined in WebSphere MQ Version 6.0 as the name of the queue to which the alias queue resolves, is renamed to `TARGET` in WebSphere MQ Version 7.0 and generalized to allow you to specify the name of either a queue or a topic. The attribute name `TARGQ` is retained for compatibility with your existing programs.

This feature is useful for migrating your existing applications to a publish/subscribe message model.

A useful example of this feature is the queue to which statistics messages are written. Before WebSphere MQ Version 7.0 there could be only a single consumer of a statistic message because a single statistics message only was written to a queue and got from a queue.

By defining a queue alias that points to a topic object, it is possible for each person interested in processing statistics messages to subscribe to the topic, rather than getting from the queue, allowing multiple consumers of the statistics information.

Within a queue sharing group it is possible to define a queue alias as a group object - this means that each queue manager in the queue sharing group will create a queue alias definition with the same name and the same properties as the `QSGDISP(GROUP)` object.

The new `TARGETTYPE` attribute may be set or altered in a `QSGDISP(GROUP)` object by a new Version 7.0 queue manager, so that the queue alias refers to a topic object. However, any Version 6 queue managers in the queue sharing group do not understand and will ignore the new `TARGETTYPE` attribute. A V6 queue manager will interpret the queue alias as referring to a queue object, regardless of the setting of `TARGETTYPE`.

Defining a queue alias is described in [Working with alias queues](#).

Parent topic: [WebSphere MQ version 6 publish/subscribe migration](#)

 This build: January 26, 2011 10:56:16

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
mi12180_

8. WebSphere Event Broker Version 6.0 or WebSphere Message Broker Version 6.0 or 6.1 publish/subscribe migration to WebSphere MQ Version 7.0.1 and later versions

You can migrate publish/subscribe configuration data from WebSphere® Event Broker Version 6.0 or WebSphere Message Broker Version 6.0 or 6.1 to a WebSphere MQ Version 7.0.1 or later queue manager.

You can migrate the following publish/subscribe configuration data:

- Subscriptions
- Subscription points. Subscription points are supported only when RFH2 messages are used.
- Streams
- Retained publications

Note: WebSphere MQ Version 7.0.1 does not support the content-based filtering provided in WebSphere Event Broker Version 6.0. Do not migrate if you use this function and intend to continue to do so.

There are three phases to the migration:

The rehearsal phase

This phase creates a migration log, reporting any errors that might be found, but does not change the current configurations. You can use this phase to observe what the result of a real migration would be. Rehearsing the migration also produces a file containing suggested security commands to set up a security environment in the queue manager that is equivalent to the security environment that existed in the broker. You can review the security command file and determine the actions that you need.

The initial phase

This phase creates topic objects that might be needed in the queue manager, based on the Access Control List entries that are defined in the broker. You must run this phase before you run the completion phase. The initial phase also produces a file containing suggested security commands to set up a security environment in the queue manager that is equivalent to the security environment that existed in the broker. The topic objects are created in anticipation of you executing the security commands to create ACLs for the topic objects. Before you run the completion phase you must review and modify the security command file as required and execute the commands that you need. The initial phase also creates a migration log.

Note: When the migration process attempts to create a topic object, it first checks whether a suitable topic object already exists in the queue manager; if it does, it uses that existing topic object. This ensures that if the migration process is run multiple times, it does not attempt to create multiple topic objects for the same purpose.

However, if you modify the properties of one of these topic objects (for example the wildcard property) the migration process does not take account of that and still uses the topic object, even though it no longer has the same properties that the migration process originally created for it. For this reason, with the exception of setting access permissions between the initial and completion phases, you must leave unchanged the topic objects that are created by the migration process until it has completed.

The completion phase

This phase retrieves the current publish/subscribe definitions from the broker and uses those definitions to create equivalent definitions in the publish/subscribe component of the WebSphere MQ Version 7.0.1 and later queue manager that is associated with the named broker. The details of the migration are documented in the log file. When the migration is complete, the queue manager publish/subscribe configuration is equivalent to the broker publish/subscribe configuration. The completion phase also creates a migration log.

The WebSphere Event Broker Version 6.0 or WebSphere Message Broker Version 6.0 or 6.1 publish/subscribe information, which is stored in the subscription database tables, is not deleted by the migration process. This information is therefore available to use until you explicitly delete it.

[Access Control List \(ACL\) migration](#)

The function that migrates publish/subscribe configuration data from WebSphere Event Broker Version 6.0 to WebSphere MQ produces a file containing suggested security commands and creates topic objects as required.

[Migrating publish/subscribe configuration data from WebSphere Event Broker V6](#)

Complete these tasks to migrate publish/subscribe configuration data from WebSphere Event Broker Version 6.0 to WebSphere MQ Version 7.0.1 and later versions.

[Migrating publish/subscribe configuration data from WebSphere Message Broker Version 6 to WebSphere MQ Version 7.0.1 or later versions and upgrading to WebSphere Message Broker Version 7](#)

Complete these tasks to migrate publish/subscribe configuration data from WebSphere Message Broker Version 6.0 or 6.1 to WebSphere MQ Version 7.0.1 or later versions and upgrade from WebSphere Message Broker Version 6.0 or 6.1 to WebSphere Message Broker Version 7.0.

[Publish/subscribe migration log file](#)

An example publish/subscribe migration log file is provided to show you the format and some example contents.

[Differences from WebSphere Message Broker Versions 6.0 and 6.1 publish/subscribe](#)

There are a number of minor differences between the publish/subscribe support in WebSphere MQ Version 7.0.1, and that in WebSphere Message Broker Version 6.0 and 6.1.

[Migrating a WebSphere Event Broker publish/subscribe collective to a WebSphere MQ publish/subscribe cluster](#)

Complete these tasks to migrate a publish/subscribe collective from WebSphere Event Broker Version 6.0 or WebSphere Message Broker Version 6.0 or 6.1 to a WebSphere MQ Version 7.0.1 or later version publish/subscribe cluster.

[Retained publications with headers in MQRFH format](#)

Retained publications in MQRFH format might lose data when migrated to WebSphere MQ Version 7.0.1 or later versions.

Parent topic: [Publish/Subscribe User's Guide](#)

Related concepts

[Access Control List \(ACL\) migration](#)

Related information

[migmbbrk \(migrate publish/subscribe information\)](#)
[The migrate publish/subscribe information utility \(CSQUMGMB\)](#)

 This build: January 26, 2011 10:56:59

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
 This topic's URL:
 ps22900_

8.1. Access Control List (ACL) migration

The function that migrates publish/subscribe configuration data from WebSphere® Event Broker Version 6.0 to WebSphere MQ produces a file containing suggested security commands and creates topic objects as required.

The migration process does not migrate the Access Control List (ACL). On WebSphere Event Broker Version 6.0, the default is that all user IDs have permission to access any topic unless the ACL explicitly denies access. In WebSphere MQ, the default is that no user ID has access to any topic unless the ACL explicitly authorizes access. WebSphere MQ does not support ACL entries that deny access. Because of this difference in security approaches, the migration process cannot directly migrate a WebSphere Event Broker Version 6.0 ACL to a WebSphere MQ Version 7.0.1 or later version ACL. Instead, the rehearsal and initial phases of the migration process produce a security command file that is a best attempt at listing the security commands which, when run, create an equivalent ACL in the WebSphere MQ queue manager.

If the rehearsal or initial phase of the migration find an ACL entry that denies access, it cannot produce a WebSphere MQ equivalent command. Instead, it reports it in the security command file and advises that the ACL migration must be performed manually. You can either modify the broker security settings to match the WebSphere MQ security approach, and run the rehearsal or initial phase of the migration again to produce a new security command file, or modify the security command file as needed. You must set up a security environment in the queue manager, equivalent to the one that existed in the broker, before you run the completion phase of the migration.



Wildcard subscriptions

If a subscription uses a wildcard scheme to subscribe to multiple topics, but access to one or more of those topics is denied by its ACL, the result of the subscription in WebSphere Event Broker differs from the result in WebSphere MQ:

- In WebSphere Event Broker the subscription succeeds, but the subscription will not receive messages that are published to topics for which access authority is denied. For example, if access to a topic USA/Alaska is permitted but access to a topic USA/Kansas is denied, in WebSphere Event Broker a subscription to USA/# or USA/+ succeeds, but the subscription will not receive messages that are published to USA/Kansas.
- In WebSphere MQ, for a subscription to succeed, it must have access authority to all of the topics to which the wildcard scheme subscribes. Therefore the subscription fails. For example, if access to a topic USA/Alaska is permitted but access to a topic USA/Kansas is denied, in WebSphere MQ a subscription to USA/# or USA/+ fails, so the subscription will not receive messages that are published to USA/Alaska or USA/Kansas.

You might therefore find, even if you have examined the security command file and taken any action that is necessary to set up a security environment that is equivalent to the one that existed in the broker, that the migration process fails to migrate subscriptions that include a wildcard, if access to one or more of those topics is denied. The solution to this is not clear-cut, because of the difference in behavior between the two products, and requires manual intervention. Depending on your security requirements you might, for example, grant the required permission to the topic USA/Kansas, if you can accept that a subscriber to USA/# or USA/+ will receive messages published to the topic. Alternatively, you might replace the wildcard subscription with multiple subscriptions lower down the topic tree.



ACL migration on Unix platforms

On Unix platforms, it is possible for a username and groupname to have the same identifier. In this case, the migration process is unable to determine whether an ACL should apply to the username, the groupname, or both, and the ACL migration script therefore contains two setmqaut commands: a first command that applies to the username and a second command that applies to the group name. Here is an example of the commands:

```
setmqaut -m QM1 -n SAMPLE_TOPIC_OBJECT -t topic -p mquser +sub +pub
# setmqaut -m QM1 -n SAMPLE_TOPIC_OBJECT -t topic -g mquser +sub +pub
```

Note that the second command is prefixed by a number sign (#) to indicate that it is commented out. As part of reviewing the ACL migration script you should ensure that the correct command is executed.

Creation of topic objects by the migration

WebSphere Event Broker Version 6.0 provides the capability to define topic trees, but there is no capability to set specific attributes for a particular individual topic in a topic tree. WebSphere MQ Version 7.0 supports the concept of topic objects that allow you to set specific, nondefault attributes for a topic. An ACL is a property of a topic object. The initial phase of the migration creates topic objects speculatively, based on the ACL entries that are defined in the broker and in anticipation that you later run the security commands to create ACLs for the topic objects. When you have resolved what security settings you need, you might need to delete the topic objects that you do not require.

Parent topic: [WebSphere Event Broker Version 6.0 or WebSphere Message Broker Version 6.0 or 6.1 publish/subscribe migration to WebSphere MQ Version 7.0.1 and later versions](#)

Related concepts
[Publish/subscribe security](#)

Related tasks
[Migrating publish/subscribe configuration data from WebSphere Event Broker V6](#)

Related information
[migmbbrk \(migrate publish/subscribe information\)](#)
[The migrate publish/subscribe information utility \(CSQUMGMB\)](#)

 This build: January 26, 2011 10:56:59

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
 This topic's URL:
 ps22910_

8.2. Migrating publish/subscribe configuration data from WebSphere Event Broker V6

Complete these tasks to migrate publish/subscribe configuration data from WebSphere® Event Broker Version 6.0 to WebSphere MQ Version 7.0.1 and later versions.

Before you begin

Make sure that you are familiar with the information in [WebSphere Event Broker Version 6.0 and WebSphere Message Broker Version 6.0 or 6.1 migration to WebSphere MQ Version 7.0.1 and later versions](#) and [Access Control List \(ACL\) migration](#).

Install WebSphere MQ Version 7.0.1 or a later version.

On distributed systems, set up and initialize a command environment for the WebSphere Event Broker from which you are migrating, in which WebSphere MQ commands and WebSphere Event Broker commands can run.

Set the queue manager **PSMODE** attribute to **COMPAT**, using the following command: `ALTER QMGR PSMODE (COMPAT)` - this stops the queue manager from processing any queued publish or subscribe messages.

Procedure

1. Optional: Run the rehearsal phase of the migration. For example, on supported platforms other than z/OS, use the following command to rehearse the migration from a broker named BRK1:

```
migmbrk -r -b BRK1
```

2. Review the contents of the log file and the security commands file to check what would happen in a real migration.

3. Run the initial phase of the migration. For example, on supported platforms other than z/OS, use the following command to run the initial phase from a broker named BRK1:

```
migmbrk -t -b BRK1
```

4. Review the commands in the security commands file `amqmigrateacl.txt`. Ensure that they will create a security environment that is equivalent to your broker security environment. If the migration rehearsal finds an ACL entry that denies access, it reports it in the security command file and advises that the ACL migration must be performed manually. There are two alternative ways that you can deal with this:
 - o Modify the security commands to ensure that they would set up a security environment in the queue manager that is equivalent to the security environment that existed in the broker. Then go to step 5.
 - o Revise the broker's security definitions to be similar to WebSphere MQ definitions, so that the migration can migrate them, by performing the following steps:
 - a. Revise the ACL entry for the root of the topic tree to be the same as for the root of the WebSphere MQ topic tree; that is, set Deny for both publish and subscribe. This is the exact opposite of the broker default and can result in many publishers and subscribers no longer having permission to perform operations that they have been doing successfully. However, the next two steps correct this.
 - b. Remove all ACL entries that deny access, apart from the entry in step 4a. Many of these entries are likely to be redundant following the previous step, however others might require more extensive changes.
 - c. Add any ACL entries that are needed to grant access, to restore a correct security environment.
 - d. Run the initial phase of the migration again and review the security command file. It should now create a security environment in the queue manager that is equivalent to the security environment that existed in the broker.

5. Run the security commands to set up the security environment before you run the completion phase of the migration, or the migration will fail.

6. Run the completion phase of the migration. For example, on supported platforms other than z/OS, use the following command to migrate the publish/subscribe configuration data from broker BRK1 and overwrite any subscription or retained publication that already exists in the queue manager and that has the same name as a migrating subscription or retained publication:

```
migmbrk -c -o -b BRK1
```

The completion phase migrates the publish/subscribe configuration data to the queue manager, creates a new log file and a new security commands file, and shuts down the broker.

Note: It is possible that the broker state has changed since the initial phase was run and that additional topic objects are now required. If so, the completion phase creates these topic objects as necessary. The completion phase does not delete topic objects that have become unnecessary; you might need to delete topic objects that you do not require.

7. Provide the queue manager with the name of every input queue to be monitored for publications, by adding the name of every queue that is currently named in an Event Broker MQInput node to the `SYSTEM.QPUBSUB.QUEUE.NAMELIST`. You must do this because, in Event Broker, published messages are put to queues and are read from the queues by the broker using an MQInput node. When the migration shuts down the broker, messages cannot be read in this way. Instead, the queue manager monitors the relevant queues, but you must provide the names of the queues to monitor. To edit a `namelist`, use either the WebSphere MQ Explorer or the following MQSC command:

```
ALTER NAMELIST
```

Note: Each queue name referenced in the `SYSTEM.QPUBSUB.QUEUE.NAMELIST` `namelist` also has an associated Topic object. You must define the Topic objects before adding the associated queue name to the `SYSTEM.QPUBSUB.QUEUE.NAMELIST` `namelist`. For more information about setting up the `SYSTEM.QPUBSUB.QUEUE.NAMELIST` `namelist`, see [Mapping between streams and topics](#).

8. Check the broker domain Event Log to confirm that the broker has been shut down.
9. Use the following command to set the **PSMODE** queue manager attribute to **ENABLED**.

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

This starts the queued publish/subscribe interface so that the queue manager deals with all subsequent publish/subscribe processing.

What to do next

The migration process uses a queue called `SYSTEM.TEMP.MIGMBBRK.REPLY.QUEUE` to receive messages from the broker. When the process starts it checks for the existence of the queue and creates or clears it as necessary. When it has finished it attempts to delete the queue; however, because the broker also uses the queue to send replies, it might still have the queue open. If so, the migration process is unable to delete the queue. When you have completed the migration, check if `SYSTEM.TEMP.MIGMBBRK.REPLY.QUEUE` exists, and if it does, delete it.

Parent topic: [WebSphere Event Broker Version 6.0 or WebSphere Message Broker Version 6.0 or 6.1 publish/subscribe migration to WebSphere MQ Version 7.0.1 and later versions](#)

Related information

[migmbrk \(migrate publish/subscribe information\)](#)

[The migrate publish/subscribe information utility \(CSQUMGMB\)](#)

 This build: January 26, 2011 10:57:00

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:
ps22920_

8.3. Migrating publish/subscribe configuration data from WebSphere Message Broker Version 6 to WebSphere MQ Version 7.0.1 or later versions and upgrading to WebSphere Message Broker Version 7

Complete these tasks to migrate publish/subscribe configuration data from WebSphere® Message Broker Version 6.0 or 6.1 to WebSphere MQ Version 7.0.1 or later versions and upgrade from WebSphere Message Broker Version 6.0 or 6.1 to WebSphere Message Broker Version 7.0.

Before you begin

It is assumed that you are familiar with the information in [WebSphere Event Broker Version 6.0 and WebSphere Message Broker Version 6.0 or 6.1 migration to WebSphere MQ Version 7.0.1 and later versions](#) and [Access Control List \(ACL\) migration](#).

Make sure that you are familiar with the information in [WebSphere Event Broker Version 6.0 and WebSphere Message Broker Version 6.0 or 6.1 migration to WebSphere MQ Version 7.0.1 and later versions](#) and [Access Control List \(ACL\) migration](#).

Install WebSphere MQ Version 7.0.1 or a later version.

On distributed systems, set up and initialize a command environment for the WebSphere Message Broker from which you are migrating, and in which WebSphere MQ commands and WebSphere Message Broker commands can run.

The WebSphere MQ queue manager is not currently handling any publish or subscribe messages.

Set the queue manager **PSMODE** attribute to **COMPAT**, using the following command: `ALTER QMGR PSMODE (COMPAT)`

Procedure

1. Optional: Run the rehearsal phase of the migration. For example, on supported platforms other than z/OS, use the following command to rehearse the migration from a broker named BRK1:

```
migmbrk -r -b BRK1
```

2. Review the contents of the log file and the security commands file to check what will happen in a real migration.

3. Run the initial phase of the migration. For example, on supported platforms other than z/OS, use the following command to rehearse the migration from a broker named BRK1:

```
migmbrk -t -b BRK1
```

4. Review the commands in the security commands file `amqmigrateacl.txt`. Ensure that they create a security environment that is equivalent to your broker security environment. If the migration rehearsal finds an ACL entry that denies access, it reports it in the security command file and advises that the ACL migration must be performed manually. There are two alternative ways that you can deal with this:

- o Modify the security commands to ensure that they would set up a security environment in the queue manager that is equivalent to the security environment that existed in the broker. Then go to step 5.
- o Revise the broker's security definitions so that they are similar to WebSphere MQ definitions, so that the migration process can migrate them, by performing the following steps:
 - a. Revise the ACL entry for the root of the topic tree to be the same as for the root of the WebSphere MQ topic tree; that is, set Deny for both publish and subscribe. This is the exact opposite of the broker default and can result in many publishers and subscribers no longer having permission to perform operations that they have been doing successfully. However, the next two steps correct this.
 - b. Remove all ACL entries that deny access, apart from the entry in step 4a. Many of these entries are likely to be redundant following the previous step, however others might require more extensive changes.
 - c. Add any ACL entries that are needed to grant access, to restore a correct security environment.
 - d. Rehearse the migration again and review the security command file. It should now create a security environment in the queue manager that is equivalent to the security environment that existed in the broker.

5. Run the security commands to set up the security environment before you run the completion phase of the migration, or the migration will fail.

6. Run the completion phase of the migration. For example, on supported platforms other than z/OS, use the following command to migrate the publish/subscribe configuration data from broker BRK1 and overwrite any subscription or retained publication that already exists in the queue manager and that has the same name as a migrating subscription or retained publication:

```
migmbrk -c -o -b BRK1
```

The completion phase migrates the publish/subscribe configuration data to the queue manager, creates a new log file and a new security commands file, and shuts down the broker.

Note: At the end of this step, the broker is stopped and the queue manager is in COMPAT mode. Consequently the publish/subscribe configuration cannot be changed in WebSphere MQ or WebSphere Message Broker, allowing the migration to proceed.

It is possible that the broker configuration has changed since the initial phase was run and that new additional topic objects are now required. If so, the completion phase creates these new topic objects as necessary. The completion phase does not delete any topic objects that have become unnecessary; you might need to delete any topic objects that you do not require.

7. Check the broker domain Event Log to confirm that the broker has been shut down.
8. Upgrade from WebSphere Message Broker Version 6.0 or 6.1 to WebSphere Message Broker Version 7.0.
9. Run the **mqsimigratecomponents** command, as described in the WebSphere Message Broker documentation.
10. Use the following command to set the **PSMODE** queue manager attribute to **ENABLED**.

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

This starts the queued publish/subscribe interface so that the queue manager deals with all subsequent publish/subscribe processing.

11. Optional: If the broker is part of a publish/subscribe collective, do not restart the broker and any associated applications until you have migrated all the brokers in the collective.

12. Restart the broker. When the broker restarts, it continues to provide message services other than publish and subscribe functions.

What to do next

The migration process uses a queue called `SYSTEM.TEMP.MIGMBBRK.REPLY.QUEUE` to receive messages from the broker. When the process starts it checks for the existence of the queue and creates or clears it as necessary. When it has finished it attempts to delete the queue; however, because the broker also uses the queue to send replies, it might still have the queue open. If so, the migration process is unable to delete the queue. When you have completed the migration, check if `SYSTEM.TEMP.MIGMBBRK.REPLY.QUEUE` exists, and if it does, delete it.

Parent topic: [WebSphere Event Broker Version 6.0 or WebSphere Message Broker Version 6.0 or 6.1 publish/subscribe migration to WebSphere MQ Version 7.0.1 and later versions](#)

 This build: January 26, 2011 10:57:00

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps22930_



8.4. Publish/subscribe migration log file

An example publish/subscribe migration log file is provided to show you the format and some example contents.

Identifying subscriptions in the log file or error messages

When the log file needs to identify a particular broker subscription in the log file or an error message, it lists its properties, including its topic string. However, the topic string can be up to 10,240 characters long, and is expressed in Unicode in the broker. This can mean that the topic string might be too long or contain characters that cannot be output in the current character set.

To provide a precise way to identify subscriptions, the migration process assigns a sequence number to every broker subscription that it encounters, starting at 1. When the migration process needs to refer to a subscription, the process includes its sequence number and anything else that is known about the subscription. You can use the `mqsireportproperties broker` command to list the broker's subscriptions in the same order as the migration process. Therefore, for example, the fourth subscription that the migration process counts is also the fourth that is listed by `mqsireportproperties`. Using this information, you can take the sequence number from the migration process's log file or error message and use it to precisely identify the subscription in the broker.

The broker command to list the subscriptions is as follows:

```
mqsireportproperties brokername -e default -o DynamicSubscriptionEngine -r
```

To store the results in a file called, for example, `outfile.txt`, the command is as follows:

```
mqsireportproperties brokername -e default -o DynamicSubscriptionEngine -r > outfile.txt
```

Example log file

The following example log files lists the actions that have been performed by a successful migration process. It shows the migration of subscriptions, ACLs and retained publications.


```
2009-01-28 11:43:54.187: Migrating Pub/Sub state from Websphere Message Broker: TEST_BROKER
2009-01-28 11:43:54.187: Into queue manager: TEST_QM
2009-01-28 11:43:54.187: Command switches:
2009-01-28 11:43:54.187:     -t
2009-01-28 11:43:54.187:     -z
2009-01-28 11:43:54.187:     -l
2009-01-28 11:43:55.484: Starting to parse subscriptions ...
2009-01-28 11:43:55.484: Migrating subscriptions for topic string RFH2/EU/France/Toison
2009-01-28 11:43:55.484: [1] Migrating subscription for:
2009-01-28 11:43:55.484:     Format: mqrh2
2009-01-28 11:43:55.484:     Queue Manager: PSMIG_QM
2009-01-28 11:43:55.484:     Queue: PUBSUB.FRANCE.QUEUE
2009-01-28 11:46:23.968: Migrating subscriptions for topic string RFH2/EU/France/Carnac
2009-01-28 11:46:23.968: [2] Migrating subscription for:
2009-01-28 11:46:23.968:     Format: mqrh2
2009-01-28 11:46:23.968:     Queue Manager: PSMIG_QM
2009-01-28 11:46:23.968:     Queue: PUBSUB.FRANCE.QUEUE
2009-01-28 11:46:23.968: Migrating subscriptions for topic string $SYS/STREAM/TEST_STREAM/RFH1/EU/France/Pontivy
2009-01-28 11:46:23.984: [3] Migrating subscription for:
2009-01-28 11:46:23.984:     Format: mqrh2
2009-01-28 11:46:23.984:     Queue Manager: PSMIG_QM
2009-01-28 11:46:23.984:     Queue: PUBSUB.FRANCE.QUEUE
2009-01-28 11:46:24.031: Migrating subscriptions for topic string $SYS/Broker+/warning/expiry/Subscription/#
2009-01-28 11:46:24.031: [4] Migrating subscription for:
2009-01-28 11:46:24.031:     Format: mqrh2
2009-01-28 11:46:24.031:     Queue Manager: PSMIG_QM
2009-01-28 11:46:24.031:     Queue: PUBSUB.SAMPLE.QUEUE
2009-01-28 11:46:24.125: Migrating subscriptions for topic string $SYS/Broker+/Subscription/#
2009-01-28 11:46:24.125: [5] Migrating subscription for:
2009-01-28 11:46:24.125:     Format: mqrh2
2009-01-28 11:46:24.125:     Queue Manager: PSMIG_QM
2009-01-28 11:46:24.125:     Queue: PUBSUB.SAMPLE.QUEUE
2009-01-28 11:46:24.140: Migrating subscriptions for topic string $SYS/Broker+/Status
2009-01-28 11:46:24.140: [6] Migrating subscription for:
2009-01-28 11:46:24.140:     Format: mqrh2
2009-01-28 11:46:24.140:     Queue Manager: PSMIG_QM
2009-01-28 11:46:24.140:     Queue: PUBSUB.SAMPLE.QUEUE
2009-01-28 11:46:24.156: Migrating subscriptions for topic string $SYS/Broker+/Status/ExecutionGroup/#
2009-01-28 11:46:24.156: [7] Migrating subscription for:
2009-01-28 11:46:24.156:     Format: mqrh2
2009-01-28 11:46:24.156:     Queue Manager: PSMIG_QM
2009-01-28 11:46:24.156:     Queue: PUBSUB.SAMPLE.QUEUE
2009-01-28 11:46:24.250: Migrating subscriptions for topic string $SYS/STREAM/TEST_STREAM/RFH1/EU/France/Kersaux
2009-01-28 11:46:24.250: [8] Migrating subscription for:
2009-01-28 11:46:24.250:     Format: mqrh2
2009-01-28 11:46:24.250:     Queue Manager: PSMIG_QM
2009-01-28 11:46:24.250:     Queue: PUBSUB.FRANCE.QUEUE
2009-01-28 11:46:24.281: ... finished parsing subscriptions
2009-01-28 11:46:24.281: Starting to parse topics ...
2009-01-28 11:46:24.281: Migrating ACLs for topic string
2009-01-28 11:46:24.281: Migrating ACLs for topic string RFH2/EU/France/Toison
2009-01-28 11:46:24.281: Migrating ACLs for topic string RFH2/EU/France/Carnac
2009-01-28 11:46:24.281: Migrating ACLs for topic string $SYS/STREAM/TEST_STREAM/RFH1/EU/France/Pontivy
2009-01-28 11:46:24.281: Migrating ACLs for topic string $SYS/Broker+/warning/expiry/Subscription/#
2009-01-28 11:46:24.281:     None found.
2009-01-28 11:46:24.281: Migrating ACLs for topic string $SYS/Broker+/Subscription/#
2009-01-28 11:46:24.281:     None found.
2009-01-28 11:46:24.281: Migrating ACLs for topic string $SYS/Broker+/Status
2009-01-28 11:46:24.281:     None found.
2009-01-28 11:46:24.281: Migrating ACLs for topic string $SYS/Broker+/Status/ExecutionGroup/#
2009-01-28 11:46:24.281:     None found.
2009-01-28 11:46:24.281: Migrating ACLs for topic string $SYS/STREAM/TEST_STREAM/RFH1/EU/France/Kersaux
2009-01-28 11:46:24.281: ... finished parsing topics
2009-01-28 11:46:24.281: Starting to parse retained publications ...
2009-01-28 11:46:24.296: Migrating retained publications for topic string $SYS/Broker/TEST_BROKER/Status
2009-01-28 11:46:24.296: Migrating retained publication for default subscription point.
2009-01-28 11:46:24.906: ... finished parsing retained publicatons
2009-01-28 11:46:24.968:
All Pub/Sub data has been retrieved from the broker.
2009-01-28 11:46:24.968: Applying changes to queue manager Pub/Sub state.
2009-01-28 11:46:24.972: Created topic object: MIGMBRK.TOPIC.00004
2009-01-28 11:46:24.972: Created topic object: MIGMBRK.TOPIC.00003
2009-01-28 11:46:24.972: Created topic object: MIGMBRK.TOPIC.00002
2009-01-28 11:46:24.972: Created topic object: MIGMBRK.TOPIC.00001
2009-01-28 11:46:24.977: Defining subscription [1]
2009-01-28 11:46:24.977: Defining subscription [2]
2009-01-28 11:46:24.977: Defining subscription [3]
2009-01-28 11:46:24.977: Defining subscription [4]
```

```

2009-01-28 11:46:24.993: Defining subscription [5]
2009-01-28 11:46:24.993: Defining subscription [6]
2009-01-28 11:46:24.993: Defining subscription [7]
2009-01-28 11:46:24.993: Defining subscription [8]
2009-01-28 11:46:29.996: Migration completion message written.

```

Parent topic: [WebSphere Event Broker Version 6.0 or WebSphere Message Broker Version 6.0 or 6.1 publish/subscribe migration to WebSphere MQ Version 7.0.1 and later versions](#)

 This build: January 26, 2011 10:57:09

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps23120_



8.5. Differences from WebSphere Message Broker Versions 6.0 and 6.1 publish/subscribe

There are a number of minor differences between the publish/subscribe support in WebSphere® MQ Version 7.0.1, and that in WebSphere Message Broker Version 6.0 and 6.1.

Parent topic: [WebSphere Event Broker Version 6.0 or WebSphere Message Broker Version 6.0 or 6.1 publish/subscribe migration to WebSphere MQ Version 7.0.1 and later versions](#)

Subscription names and traditional identities

In WebSphere Message Broker (and WebSphere MQ Version 6), a subscription with a subscription name (SubName) retains its traditional identity. This means that, for example, if a Deregister Subscriber message requests that all the subscriptions that have a given traditional identity are unsubscribed, the broker unsubscribes subscriptions that have a subscription name. In WebSphere MQ Version 7.0.1, the same Deregister Subscriber request leaves subscriptions with a subscription name unchanged, because WebSphere MQ Version 7.0.1 identifies subscriptions for that traditional identity by using the subscription name that is generated by the queued publish/subscribe interface.

Using a new subscription point

To use a new subscription point in WebSphere MQ Version 7.0.1 and later versions, you must perform the following actions:

1. Create a topic object whose topic string is the name of the subscription point
2. Add the topic object to the namelist SYSTEM.QPUBSUB.SUBPOINT.NAMELIST

In WebSphere Message Broker Versions 6.0 and 6.1 these actions are not necessary.

Publications and subscription points

In WebSphere Message Broker, RFH2 publishers did not specify a subscription point tag in the <psc> folder. In WebSphere MQ, publishers tell the queued publish/subscribe interface which subscription point they want to publish on by adding a subscription point to their <psc> folder.

Putting RFH2 publications on a queue

In WebSphere MQ, complete the following steps to put an RFH2 publication on a queue so that the queued publish/subscribe interface can read them:

1. Create a queue.
2. Create a topic object with the same name as the queue.
3. Add the topic to the namelist SYSTEM.QPUBSUB.QUEUE.NAMELIST.

Publication when subscribers are added or removed

In WebSphere Message Broker Version 6.1, an application can subscribe to topics that publish a message when a subscriber is added to the system and when a subscriber is removed. WebSphere MQ has no direct equivalent. The information is available in WebSphere MQ by using MQSC or PCF commands but this information has to be polled rather than being automatically sent to interested parties.

Fields that cannot be altered

In a WebSphere Message Broker subscription, the following fields can be altered:

- Topic
- Filter
- Destination queue
- Subscription name

In WebSphere Message Broker you can alter these fields only for subscriptions that have a subscription name. Otherwise the rename request in WebSphere Message Broker has no way to refer to the original subscription and instead creates a new subscription.

In WebSphere MQ you cannot alter these fields. You can add a subscription name to a subscription that does not have one, by using the registration options property AddName but when it has been added it cannot be renamed and the subscription can no longer be referred to by its traditional identity.

Publication or subscription to multiple topics

In WebSphere Message Broker a single command message can publish or subscribe to multiple separate topics. In WebSphere MQ the queued publish/subscribe interface supports this but it does so by creating multiple separate publications or subscriptions. Consider, for example, a subscribing application that uses a single command message to subscribe to two topics, topic1 and topic2, and a publishing application that uses a single command to publish a single message on both topic1 and topic2. In WebSphere Message Broker, the subscribing application receives a single message, which the MQRFH2 header lists as being on topic1 and topic2. However, in WebSphere MQ Version 7.0.1 and later versions, the subscribing application receives two separate copies of the message: one on topic1 and one on topic2.

User separation of traditional identities

In WebSphere Message Broker publish/subscribe, when a user has registered a subscription using a particular traditional identity, no other user can register a subscription on the same traditional identity, unless the owning user removes their subscriptions. In WebSphere MQ there is no restriction on users creating subscriptions to the same traditional identity as other subscribers.


Persistence property of retained messages

WebSphere Message Broker and WebSphere MQ differ in how they interpret the persistence property for retained publication messages.


In WebSphere Message Broker, the persistence property controls only how the message is delivered; that is, whether it is persistent or nonpersistent when it is delivered to a subscriber. Unless the subscriber requests persistence, nonpersistent retained publications are delivered as nonpersistent. However, a persistent or nonpersistent retained publication is stored by the broker until it is explicitly deleted, either by an RFH2 delete publication message or because the expiry time for the message has passed.

In WebSphere MQ, the persistence property controls the life span of the retained publication and whether the publication message is delivered to a subscriber as persistent or nonpersistent. Unless the subscriber requests persistence, nonpersistent retained publications are delivered as nonpersistent. A persistent retained publication survives an event that would cause a nonpersistent publication to be discarded, such as restarting a queue manager. However, unlike in WebSphere Message Broker, a nonpersistent retained message is discarded when a similar event occurs.

Consequently, when a nonpersistent retained message is migrated from WebSphere Message Broker to a WebSphere MQ Version 7.0.1 queue manager, the fact that it is marked as nonpersistent causes it to be discarded at the first queue manager restart. In general, this difference in behavior is usually not significant because it is not common for retained messages to be marked as nonpersistent. However, you must code your application to recover nonpersistent messages if their recovery is required.

 This build: January 26, 2011 10:57:10

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

 Copyright IBM Corporation 1999, 2009. All Rights Reserved.

This topic's URL:

ps23130_

8.6. Migrating a WebSphere Event Broker publish/subscribe collective to a WebSphere MQ publish/subscribe cluster

Complete these tasks to migrate a publish/subscribe collective from WebSphere® Event Broker Version 6.0 or WebSphere Message Broker Version 6.0 or 6.1 to a WebSphere MQ Version 7.0.1 or later version publish/subscribe cluster.

Before you begin

Install WebSphere MQ Version 7.0.1 or a later version.

Make sure that you are familiar with the information in [WebSphere Event Broker Version 6.0 migration to WebSphere MQ Version 7.0.1 and later versions](#) and [Access Control List \(ACL\) migration](#).

Make sure that you are familiar with the first tasks to set up a new cluster and add a queue manager to it, described in [WebSphere MQ Queue Manager Clusters](#).

On distributed systems, set up and initialize a command environment in which WebSphere MQ commands and WebSphere Event Broker commands can run.

Set the queue manager **PSMODE** attribute to `COMPAT`, using the following command: `ALTER QMGR PSMODE(COMPAT)`

Procedure

- Stop all the publish/subscribe applications in the topology and allow all in-flight messages to be processed.
- Use the MQSC command `DISPLAY QUEUE(*)` to check that the current depth on the following queues is zero:
 - `SYSTEM.BROKER.INTERBROKER.QUEUE.1A`
 - `SYSTEM.BROKER.INTERBROKER.QUEUE.1N`
 - `SYSTEM.BROKER.INTERBROKER.QUEUE.1T`
 - `SYSTEM.BROKER.CONTROL.QUEUE`
 - Transmit queues
 - Message flow input queues
- Upgrade the underlying queue manager of each broker in the collective to WebSphere MQ 7.0.1 or a later version.
- If any queue manager is currently connected using a manually defined channel and transmit queue, delete the manually defined channel now. During the removal of the transmit queues, because all applications are stopped, it is possible that messages from the WebSphere Message Broker configuration manager might be left stranded on transmit queues. However, this does not present a problem because the configuration manager automatically attempts to re-establish communication if it does not receive a timely response. When migration is complete, the configuration manager is no longer used.

Note: If you need to roll back the publish/subscribe configuration migration, you must recreate these manually defined channels and transmit queues, or the cluster channels will be used for communication between the brokers.
- Use MQSC commands to set up a queue manager cluster containing all the queue managers that are associated with the brokers:
 - Decide on a cluster name
 - Nominate two queue managers as full repository queue managers:


```
ALTER QMGR REPOS('clusname')
```
 - Define the cluster receiver channel on each queue manager:


```
DEFINE CHANNEL('to.qmgr_name') CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('hostname(fr_listener_port)') CLUSTER('clusname')
```
 - Start the channel:


```
START CHANNEL('to.qmgr_name')
```
 - Define the cluster to send to the full repository on each queue manager:


```
DEFINE CHANNEL('to.fr_qmgr_name') CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('fr_hostname(fr_listener_port)') CLUSTER('clusname')
```
 - Start the channel:


```
START CHANNEL('to.fr_qmgr_name')
```
- Create the publish/subscribe cluster. On each queue manager, alter the topic `SYSTEM.BASE.TOPIC` to add it to the cluster:


```
ALTER TOPIC(SYSTEM.BASE.TOPIC) CLUSTER('clusname')
```
- Allow time for the cluster topics to propagate and then check that all queue managers are correctly participating in the cluster:


```
DISPLAY CLUSQMGR(*)
```

8. For each queue manager, migrate the publish/subscribe configuration data to WebSphere MQ. See [Migrating publish/subscribe configuration data from WebSphere Event Broker V6](#).
9. When each broker has had its publish/subscribe configuration data migrated to the queue manager, use this MQSC command to trigger the resynchronization of proxy subscriptions with all the other queue managers in the publish/subscribe cluster:


```
REFRESH QMGR TYPE (PROXYSUB)
```

Note: If, for any reason, you have cause to roll back and rerun the broker publish/subscribe migration, you must run this resynchronization step and all the steps that follow it.
10. Allow all proxy subscriptions and retained publications to be propagated and check that the status of subscriptions and publications is what you expect to see:
 - a. Check the proxy subscriptions:


```
DISPLAY SUB (*) SUBTYPE (PROXY)
```
 - b. Check retained publications:


```
DISPLAY TPSTATUS ('#') RETAINED
```
 - c. Use this MQSC command to check that the current depth of the following queues is zero:


```
DISPLAY QLOCAL (*)
SYSTEM.INTER.QMGR.CONTROL
SYSTEM.INTER.QMGR.PUBS
SYSTEM.INTER.QMGR.FANREQ
SYSTEM.CLUSTER.TRANSMIT.QUEUE
```
11. Restart your publish/subscribe applications.

What to do next

The migration process uses a queue called `SYSTEM.TEMP.MIGMBBRK.REPLY.QUEUE` to receive messages from the broker. When the process starts it checks for the existence of the queue and creates or clears it as necessary. When it has finished it attempts to delete the queue; however, because the broker also uses the queue to send replies, it might still have the queue open. If so, the migration process is unable to delete the queue. When you have completed the migration, check if `SYSTEM.TEMP.MIGMBBRK.REPLY.QUEUE` exists, and if it does, delete it.

Parent topic: [WebSphere Event Broker Version 6.0 or WebSphere Message Broker Version 6.0 or 6.1 publish/subscribe migration to WebSphere MQ Version 7.0.1 and later versions](#)

 This build: January 26, 2011 10:57:13

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps29100_

8.7. Retained publications with headers in MQRFH format

Retained publications in MQRFH format might lose data when migrated to WebSphere MQ Version 7.0.1 or later versions.

WebSphere Message Broker and WebSphere Event Broker applications that communicate with each other by using publish/subscribe can do so regardless of the message format that they use. WebSphere Message Broker delivers the message in the format of the subscription and provides automatic conversion to ensure that a subscriber receives messages in the requested format.

WebSphere Message Broker and WebSphere Event Broker applications generally use the MQRFH2 message header, but it is possible that an application might have used the MQRFH format.

The migration of publish/subscribe information from WebSphere Message Broker or WebSphere Event Broker to WebSphere MQ Version 7.0.1 requests messages in MQRFH2 format. It is rare for WebSphere Message Broker and WebSphere Event Broker client applications to use messages in MQRFH format. However, if an application does use retained messages in MQRFH format, it is possible that some truncation of data might occur when migrated. In particular, data passed using the MQPSSStringData and MQPSIntData name/value pairs is not migrated.

The migration function checks for two conditions in the data that is returned from the broker:

- One or more retained messages stored in the broker
- One or more MQRFH subscription

If both these conditions are true, the migration function displays a warning message and writes a warning message in the migration log stating that MQRFH retained publications have been migrated with a possible loss of data.

MQRFH2 retained publications are migrated correctly.

Parent topic: [WebSphere Event Broker Version 6.0 or WebSphere Message Broker Version 6.0 or 6.1 publish/subscribe migration to WebSphere MQ Version 7.0.1 and later versions](#)

 This build: January 26, 2011 10:57:00

[Notices](#) | [Trademarks](#) | [Downloads](#) | [Library](#) | [Support](#) | [Feedback](#)

© Copyright IBM Corporation 1999, 2009. All Rights Reserved.
This topic's URL:
ps22940_