MD01: MQSeries - Standards and conventions Version 1.0

Document Number MD01 SCRIPT

Unclassified

MQSeries

Commercial Messaging

Take Note!

Before using this Manual and the product it supports, be sure to read the general information under "Notices".

First Edition, August 1998

This edition applies to Version 1.0 of MD01: MQSeries - Standards and conventions, and to all subsequent releases and modifications until otherwise indicated in new editions.

A form for reader's comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM United Kingdom Laboratories
Transaction Systems Technical Sales Support (MP102)
Hursley Park
Hursley
Hampshire, SO21 2JN, England

Contents

Notices.

Preface

A key element of success in using MQSeries is to ***plan ahead***; and one important aspect of this consists of adopting a set of workable standards and conventions. It is a frequent question from those aspiring to get the best from MQSeries, and the aim of this document is to address that question.

There have been various suggestions for MQ standards since the product was first introduced. None has been comprehensive; some have offered contradictory advice; some advice would undo designed features of MQSeries if followed.

Of course, almost any standard is better than none, so all these proposals have had some support. This document aims to recommend some standards and hints, encompassing all aspects of MQSeries, and allowing MQSeries to be exploited the way it was designed.

Users are at liberty to use whatever from these standards they think is appropriate for them. You would in any case want to augment these suggestions with house standards as needed.

The emphasis is on production use of MQSeries. Some educational or test environments may be less rigorous in adhering to these standards, but they can use it as a base.

How this Document is Organised

The document is organised as follows:

Chapter 1, General Items
Provides a general introduction to these standards. There are some basic recommendations that would apply throughout MQSeries included here. It covers defaults as well as general naming standards.

Chapter 2, MQ Network Structure
The approach taken in this document is to discuss this topic separately from applications. When MQSeries is deployed on a small scale, the boundaries get blurred, and it is difficult to see them separately.

As the use of MQSeries grows, it can be useful to have considered this part of the configuration separately, the collection of queue managers and the connections between them. These components need have no specific knowledge of the applications they support. They are able to support multiple applications, or to run new applications without change.

Chapter 3, Applications
The general goal behind these recommendations is to make applications transparent to where they fit in the MQ network structure. They do depend on that structure for message delivery, but the application specific configuration should not have to depend on how that is done.

There is one aspect of MQSeries omitted from the first edition of this standard - the new area concerning MQSeries Business Integration.

Bibliography

- *Information about MQSeries on the Internet*

  The MQSeries home page

  The URL of the MQSeries product family home page is:

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

- *Evaluating products*

  GC33-0805 *MQSeries: An Introduction to Messaging and Queuing*

- *Planning*

  GC33-1349 *MQSeries Planning Guide*

- *Special topics*

  SC33-1872 *MQSeries Intercommunication*

- *Administration*

  GC33-1632 *MQSeries Clients*

  SC33-1369 *MQSeries Command Reference*

- *Application programming*

  SC33-0807 *MQSeries Application Programming Guide*

  SC33-1673 *MQSeries Application Programming Reference*

  SX33-6095 *MQSeries Application Programming Reference Summary*

Chapter 1. General Items

This document is intended for use by Management, Systems Administrators, Application Developers, Standards Committees, and any others that will support MQSeries networks or design MQSeries applications. It is designed to provide a common base from which all MQSeries personnel can work.

Its intended benefits are as follows:

- Consistency in applications and administration processes
- Maximum availability of applications
- Avoiding common mistakes made by beginners
- Assistance to those in the early stages of becoming MQSeries experts
- General assurance of a smooth start for successful MQSeries projects

Acceptance and implementation of these standards is at your discretion. The recommendations that follow have been built up to incorporate wide experience with MQSeries. You may in any case want to augment these suggestions by adding house standards as needed.

The emphasis is on production use of MQSeries. Some educational or test environments may be less rigorous in adhering to these standards, but they can use it as a base.

Administration

Identify the "MQA"

Successful users have identified an MQSeries Administrator, to keep control of the running systems, including the use of any standards. (Some have coined the term "MQA", inspired by the similar role of a "DBA".)

- You can have a single MQA; or there can be benefit in a small team, dividing the responsibilities for mainframe and distributed for example. As long as they work well together that can be successful.
- The MQAs need to have had appropriate MQSeries training; ideally the MQA should be an ***IBM Certified Specialist*** in MQSeries.

  Information about MQSeries education and the Certification program can be found on the MQSeries web site.
- The MQA, or this small team, will also need to work in conjunction with security and network administrators.

The main thing is that the role is identified. ***Do it sooner rather than later.***

Object Names

All MQSeries names follow the convention for MQSeries, rather than the standard for object names on each supported platform. Object names may need to be used across platforms for example.

Don't use lower case letters

MQSeries allows both upper and lower case letters in its names.

Remember, however, that MQSeries names are ***case-sensitive***. This is apt to be a common source of error. This is compounded because some tools fold strings to upper case.

Don't use % in names

This character is valid in all MQSeries names, although it is not commonly used in other names across platforms. (The property that determined which characters were permitted is that no conversion is required between ASCII, or between EBCDIC, code pages.)

Choose meaningful names, within the constraints of the standards

This should be fairly obvious; help the MQA.

There is no implied structure, or hierarchy, in the name, such as you might find on many systems' file names. MQ just compares the strings.

These standards do recommend using hierarchical names in places; that is because they can be more useful that way. In some cases there is a recommendation for a suffix where there are multiple "versions" of an object.

Document the names

Remember users may be in different departments; using different platforms.

Always include a Description

All objects have a DESCR attribute for this purpose. MQSeries takes no action on the value, but allows it to be viewed.

- The character set in the Description is not limited to those used to construct MQSeries object names. Its purpose is to help the MQA. This may in fact be more readable in mixed case, and it can include national language, including DBCS, characters where appropriate.

Save the definitions

There are various reasons for doing this.

- In the case of a failure you may need to re-create the objects. This requires you save the definitions separate from the queue manager.
- Even without a failure, it can be useful to reset the attributes to a known state. For example if triggering has been turned off, or GET or PUT disabled, it is helpful to be able to restore the objects to their initial state.
- It can supplement the documentation

For example MQSC scripts, or CL programs on OS/400, would do; remember to include the REPLACE option.

Defaults

Ideally leave defaults unchanged

MQSeries generally keeps attribute defaults in standard objects, 'SYSTEM.DEFAULT.*'. When an object is defined, MQ takes any unspecified attributes from the corresponding default object.

The original intent of this approach was to support users who wanted to have different defaults. The various platforms supply these defaults in different ways.

- MQSeries for MVS/ESA provides a script which can serve as the "Initialization Input Data Set" in the queue manager JCL.
- MQSeries for OS/400 provides a CL program, AMQSDEF4; the source is in QMQMSAMP.

- Other systems have a supplied an MQSC script (AMQSCOMA), intended to be run once after the queue manager is created. Version 5 implementations do not supply this set of commands though; the standard default objects are created automatically when a queue manager is created.

Accept the MQSeries defaults, unless there is a good reason to change them - much care went into deciding what they should be.

If you must change defaults, use a Customization file

Don't change the supplied script, even though this was the original intended purpose - you would lose changes if there is a subsequent product update.

In fact there are more compelling reasons. Most queue managers require all attributes to be specified when DEFAULT objects are created. A modified script would fail if a later product release introduced new attributes. Even more compelling is that the Version 5 products don not even include a command file that can be edited.

A better approach is to have a separate Customization file; use ALTER commands to change just the attributes where the defaults are to be different.

- Concatenate with the Initialization Input Dataset on MVS
- Elsewhere, run the changes after the standard system defaults have been created.

Use the Customization file for Queue Manager attributes

Some characteristics are configured when a queue manager is created, and can not be changed after that. The following advice clearly does not apply to those attributes that can not change.

Specifying other queue manager attributes in a Customization file, in addition to being simpler, provides a direct way for all the values to be returned to a known state.

- Example 1, on an MVS queue manager:.

```
ALTER QMGR   +
  DESCR('Queue manager = MARS')   +
  DEADQ('SYSTEM.DEAD.LETTER.QUEUE')
```

- Example 2, on one of several AIX systems connected to that MVS queue manager:

```
ALTER QMGR   +
  DESCR('Queue manager = JUPITER4')   +
  DEADQ('SYSTEM.DEAD.LETTER.QUEUE')   +
  DEFXMITQ('MARS')
```

Use templates for default classes

Remember that an alternative to system defaults is to use DEFINE LIKE; objects are defined with reference to a known Defaults object, a template object. Identify these clearly by using "TEMPLATE" as part of the name.

Chapter 2. MQ Network Structure

The approach taken in this document is to discuss this topic separately from applications. When MQSeries is deployed on a small scale, the boundaries get blurred, and it is difficult to see them separately.

As the use of MQSeries grows, it can be useful to have considered this part of the configuration separately, the collection of queue managers and the connections between them. These components need have no specific knowledge of the applications they support. They are able to support multiple applications, or to run new applications without change.

Queue Managers

Assign unique names to production queue managers

This sounds obvious, but is often ignored - and it is a cause of problems.

A queue manager can be understood as a "container" for queues and related objects. There is typically one per system, but you can usually define additional queue managers.

- On a large system particularly, it may be useful to keep a test environment separate, on the same system.
- On systems that support fail over, a queue manager may be recovered on a processor that normally has its own queue manager.
- Where applications are constrained, for example by MQ log writes, multiple queue managers can be a way to increase capacity.

Queue Managers with the same name can be configured to exchange messages - by using Queue Manager aliases. ***But this is strongly discouraged.*** There are some examples where this can lead to ambiguity, and thus messages being sent to the wrong queue manager.

- If ReplyToQMgr is left blank in the Message Descriptor, MQSeries inserts the actual local Queue Manager name, not its alias.
- Dead Letter Queue messages identify the real Queue Manager, not any alias.

Don't just copy documentation examples

This is a sure way to produce queue managers with duplicate names; like CSQ1, neptune, etc. Instead, plan ahead the names of production queue managers.

Keep the queue manager name short

On MVS it has to be - the queue manager name corresponds to the MVS subsystem name. Hence the queue manager name is restricted to 4 characters. (It must be distinct from other subsystem names on the same MVS, and some users have taken to calling their queue managers "MQ..".) An alternative would be a convention such as the following.

- Example: ADDX

  A = geographic area
  DD = company division
  X = distinguishing identifier

Elsewhere, although a longer name is allowed, a queue manager is conventionally given a short name.

- There are not generally so many queue managers that this causes any problem.
- Many queue managers use the first 8 characters when generating unique message identifiers.

- MQSeries for Tandem NSK uses the first 7 characters as the root of subvolume names.
- The naming convention for channels in this document incorporates the connected queue manager names, and channel names are limited to 20 characters.

A typical choice would be to make it the same as the network host name. Otherwise, try a convention similar to these examples illustrated below.

- Example: CCCDDFNN

  CCC = city identifier
  DD = company division
  F = queue manager function (eg Test)
  NN = numeric identifier

- Example: SSSCCFNN

  SSS = stock ticker symbol
  CC = city identifier
  F = queue manager function
  NN = numeric identifier

The numeric identifier in these examples could be appropriate where a processor has multiple queue managers.

For a Queue Manager Alias, add a suffix to the name

The main use for this would be to support "classes of service". There are fewer constraints on the length of an alias name; it could be more than 8 (or 4 on MVS) for example.

In fact this feature is usually related to defining multiple channels between a pair of queue managers. In this case, use the same suffix for associated channels and queue manager aliases. The limit on the length of channel names suggests limiting this kind of suffix to 3 characters.

- Example 3, the AIX queue manager in Example 2 on page 3 needs an alias so it can receive very large reply messages on a separate channel.

```
DEFINE QREMOTE('JUPITER4_XL') REPLACE +
  DESCR('Queue manager alias for very big messages')  +
  RQMNAME('JUPITER4')
```

Default Queue Manager

Don't identify one Queue Manager as the default

Some environments can tolerate an exception, most notably CICS/ESA, where any CICS region is always connected to a single Queue Manager.

Most platforms can have more than one queue manager defined on a system. Don't pick one as the default; this is a common source of error, selecting the wrong queue manager.

Even when there is only one queue manager configured, don't define it the default. Doing so can make some operations a little easier, but it leaves open the scope for errors if another queue manager should be added at a later date.

MQSeries for OS/400 is limited to one queue manager on any system, so any queue manager is effectively a default one. However, in the interests of general MQSeries consistency, try to avoid an assumption of a default queue manager even here.

Pass the connection name as a program parameter

This allows a program to run unchanged on any appropriate Queue Manager. Hence it could have multiple concurrent instances; or a queue driven service could be moved to a different queue manager without affecting the code. The mechanism for passing this data can be any suitable programming technique; a system parameter might be an obvious choice, but including the name in a file could be acceptable too.

Note that triggering would usually provide this information as part of the MQTMC2 structure. There are a couple of exceptions.

- The supplied triggering functions for CICS/ESA and OS/400 do not include a queue manager name in the parameter. Programs triggered in these two environments already have the queue manager identified.
- Some compilers and systems restrict the length of system parameter they can accept, and so exclude this part of the MQTMC2.

Storage Class

Name it to describe the function

There shouldn't be too many of these so a simple name is sufficient. If it is a storage class for IMS Bridge queues, you could just call it "IMS" for example.

Note that there is no value in including the fact that it's a storage class as part of the name. They have a separate name space from other MQSeries objects, and the fact they appear in this object list should be sufficient indication.

Dead Letter Queue

If MQSeries can detect an error synchronously, it is reported directly to the application; if a message can not be delivered after that it is a candidate for the Dead Letter Queue. This preserves a message that can not be processed immediately, without stopping valid messages in the meantime.

- The facility is available on all platforms except MQSeries for Windows V2.
- MQSeries for OS/400 documentation refers to it as the "undelivered-message" queue.

Although normally described as a channel function, there are other MQSeries components that write to the Dead Letter Queue, including Trigger Monitors and the IMS Bridge.

Include a Dead Letter Queue on all queue managers

On all queue managers, use a local queue called ***SYSTEM.DEAD.LETTER.QUEUE***.

This is created automatically by some MQSeries platforms. On those platforms that do not, create a queue with this same name; it will cause less confusion to use a common name everywhere.

It is still necessary to configure the queue manager, by identifying this queue in its DEADQ attribute.

If a Dead Letter Queue is required, and is not available, a channel will fail.

Some users have avoided defining a Dead Letter Queue in order to detect errors sooner, but that is not recommended. The problem with this approach is that one rogue message is sufficient to stop all messages across a channel.

Consider ways to avoid unnecessary DLQ messages

Some platforms allow an automatic retry if a message can not be delivered immediately. It is specified by parameters on a receiving channel, and the conditions can be changed through a Retry Exit.

The channel is paused while such retry is in progress. Thus, transient errors can be tried again to avoid messages being written to the Dead Letter Queue unnecessarily.

A further possibility is for applications to specify MQRO_DISCARD as a Report Option. Such a message would not be placed on the Dead Letter Queue, but discarded instead. In fact this option would often be combined with MQRO_EXCEPTION_WITH_FULL_DATA, so an undelivered message would be returned to the Reply Queue, sometimes described as "return to sender".

Process the undelivered messages

Messages that are put on the Dead Letter Queue take the form of the original message data, preceded by a **dead letter header** - defined by the MQDLH structure. The header includes the intended destination queue, and queue manager, for the message, and the Reason it could not be delivered.

Listing the contents can be sufficient for a test system. A production environment must have a process, triggered or scheduled at intervals, to dispose of the messages appropriately. Some platforms supply a Dead Letter Queue Handler (rules driven); otherwise you would need a program for this purpose.

- Construct rules based on queue names, message type, feedback code, etc. It can be appropriate in some cases to retry or discard certain messages.
- Where no such action is appropriate, transfer the undelivered message to an application queue for action there.

Channels

Transmission Queue

Use exactly the same name as the destination queue manager

MQSeries will select this name in the absence of other information. Note you can not rely on there being a QREMOTE to define a transmission queue in all cases. A notable example is a message to the Reply Queue, which will only have a destination Queue Manager name from which to determine the routing.

- Example 4, the AIX queue manager in Example 2 on page 3 needs a transmission queue to access the MVS hub queue manager.

```
DEFINE QLOCAL('MARS') REPLACE    +
  DESCR('Transmission queue, sending to MARS')  +
  USAGE(XMITQ)  TRIGGER          +
  INITQ('SYSTEM.CHANNEL.INITQ')  +
  TRIGDATA('JUPITER4/MARS')
```

If there is more than one channel, add a suffix

This is connected to the earlier standard for Queue Manager aliases, and their association with classes of service. Note, the technique is to specify your queue manager alias as the ReplyToQMgr; the remote system would thus use that as the transmission queue for its reply.

Use the same suffix for a transmission queue and its destination Queue Manager alias in this situation.

- Example 5, the same AIX queue manager has a separate channel to receive very big messages.

```
DEFINE QLOCAL('MARS_XL') REPLACE    +
  DESCR('Transmission queue, big messages to MARS')  +
  USAGE(XMITQ)  TRIGGER          +
  INITQ('SYSTEM.CHANNEL.INITQ')  +
  TRIGDATA('JUPITER4/MARS_XL')
```

Take care with Default transmission queue

This feature is not available on all platforms; where it is supported, it is a convenient way to avoid having to define a transmission queue (and channel) for all possible destinations.

It is particularly useful for end point nodes in an MQSeries network. It can also be safe to use this facility for example when a branch office queue manager sends messages through a headquarters hub system.

The configuration that must be avoided is a loop of default transmission queues. MQSeries does not detect this situation, and continues to forward the messages as directed.

Make triggering standard for a Sender channel

Configure its transmission queue for triggering.

- Always use trigger type FIRST, and TRIGMPRI(0).
- On Version 5 platforms, the corresponding channel name is specified as Trigger Data. Elsewhere configure a Process object as documented.
- Use the supplied Initiation Queue name, 'SYSTEM.CHANNEL.INITQ'

Remember to have started the Channel Initiator.

A Requester channel is intended to initiate message transfer from the destination system. Its corresponding Server channel does not therefore need to be triggered.

Message Channels

Naming convention is <source>/<target>

<source> and <target> are the names of the communicating queue managers. The MQSeries limit is 20 characters for this name.

Note that this is equivalent to <source>/<xmitq> if you follow the standard naming for a transmission queue. Moreover, this correspondence can be generalized to multiple channels, and <target> is then the receiving queue manager alias. The previous examples illustrate channel names following this convention.

The recommendation, in order for this generalization to work, is that the channel, its transmission queue, and the destination queue manager alias, all have the same suffix.

The same convention applies to dynamic channels, introduced in MQV5. If a Sender channel is started, and the corresponding Receiver channel has not been defined, the Receiver is created automatically.

Include the transport type if it adds value

Some users have found it unnecessary to include the transport type in the naming convention for channels. If all you have is a TCP/IP network, it does not really help to use the limited characters in all the channel names to say so.

Other users though, particularly where a queue manager is in a mixed network, have found it a useful suggestion to indicate the network protocol in the naming convention. If this is needed, make the transport distinction evident in the ***class of service suffix***; for example 'MARS/JUPITER4***_SNA***'.

Client Connections

Don't create a channel for each separate client

In this case there is no source Queue Manager to construct the longer form. Defining a separate channel for each client represents unnecessary effort.

Use the same name, 'CLIENTS', on all queue managers. If multiple connections have to be configured, such as different transport types, add a suffix to this name.

This convention works when clients are configured with environment variables; it also works with a client definition table, for example a client that has multiple queue managers it can connect to.

MQSeries for Windows V2

These queue managers have two types of object not found elsewhere, Channel Groups and Connections. Since the names are not cross platform, there is less need to impose a system wide standard. (There is less need to be rigorous about upper case too; the same user must make them match either way.)

- For a Channel Group, name it to describe the function performed - that it is a dial up group, or the channel group to access a particular application for example.

  If the system has multiple queue managers, don't use duplicate channel group names on the system.

- A Connection identifies a combination of a queue manager and (optionally) a channel group. Name it the same as the Channel Group. If it is a Standalone connection use the queue manager name.

Chapter 3. Applications

These recommendations assume a suitable MQSeries network, such as that described in the previous chapter. The goal here is to make application code transparent to any configuration changes.

Queues

Names

Name a queue to describe its function
A message driven program provides some service. Naming a queue to describe this service seems obvious; the converse, excluding unrelated information from the name is less so.

Use hierarchical names for application queues
The form that is often recommended is as follows.

<application>.<function>

MQSeries uses the prefix 'SYSTEM.*' for objects it delivers; don't use this for application related queues.

Using a prefix to group related queues simplifies some areas of MQSeries administration. For example,

- enquiries about queues
- MVS security administration
- Dead Letter Queue handler

In a bigger application, it can be appropriate to adopt more levels in this naming hierarchy. For example,

<system>.<application>.<function>.<sub-function>

In test environments, you could similarly consider making the high level qualifier the User ID of the owner of a test queue.

Don't include the Queue Manager name
MQSeries generally identifies a queue by a pair of names, the queue name itself and the containing queue manager. Including the queue manager as part of a queue name is at least superfluous then.

If a queue is moved, a new queue manager name must be identified, but there is no need to change the queue name as well. Where MQSeries supports a directory function, applications would see no change at all.

Where an application is rolled out over multiple nodes there is no need to invent a new queue name for each instance.

Don't include the queue type in the name
MQSeries administration makes queue types transparent to applications. Don't make the type visible in the queue name; if the type is changed later, the queue name does not have to be changed as well.

Pass the name of the input queue by parameter

Each service needs a QLOCAL to provide its input. Generalise the application code by passing the queue name as parameter. Multiple instances of a service can use different local queues, without having to change the code.

Note that programs that are triggered will meet this condition; the local queue name is part of the trigger parameter.

Consider including program logic to test whether its parameter is really a trigger message structure, or something that might have been passed from the system environment. This would support a program that could be invoked either by triggering or by command line.

Versions

Indicate a version by a suffix to local queue name

There may be occasions when multiple versions of a queue exist at the same time. The reason may be related to different versions of the function driven by the queue; or the application may assign a different local queue for certain time intervals.

Indicate the version in the form of a suffix on a local queue name. For example,

<application>.<function>_TEST

<application>.<function>_V2.1

<application>.<function>_THURSDAY

A queue name as a parameter will ensure the application code is transparent to this.

Use aliasing to PUT messages to the right version

This is particularly useful where a message is PUT to a queue to request a service. The choice of the correct version of the local queue should not be the responsibility of the requesting program.

Use the same queue name across all platforms to PUT messages.

- Define it as QALIAS or QREMOTE as appropriate; don't include the queue type in the name.
- If you have a Directory service, use a QALIAS with SCOPE(CELL) instead.

Don't include the version suffix in this alias name. When the time comes to start using a new version of the local queue, just change these alias definitions. Programs will not need to be changed when the version changes in this way.

Note that there is an additional use on MVS for using aliasing in this way - it enables RACF permissions for GET and PUT to be separated.

Reply Queue

Naming convention <application>.REPLY

This fits in with the hierarchy convention described above.

Specifically don't include the queue type, QM or QL, since this is an aspect of the configuration that could be varied, such as for performance tuning.

An Alias could be also used, for example if a shared reply queue has multiple versions. Note that MQSeries will have resolved this to the correct local queue in any Message Descriptor that is sent.

Options for Reply Queue type

There are various application approaches to processing a reply queue which imply different queue types. The naming conventions above works in each case, though there are different considerations in each case. Where choices can be made through configuration, consider writing the program logic so that it is transparent to this tuning.

Exclusive

The fixed name is usually a Model Queue, opened to INPUT the replies; the generated Dynamic queue is specified as the MQMD.ReplyToQ. As a temporary dynamic queue it would be appropriate for replies to non-persistent requests. All replies belong exclusively to the requesting program, and the queue is deleted when it is closed.

In fact a similar program could also work when the reply queue is local, and opened for Exclusive Input; persistent messages could then be included.

Shared

Getting reply messages (selecting by CorrelId) from a shared local queue can have a performance advantage - certainly in avoiding the overhead of creating a new dynamic queue each time, but often in general message retrieval as well.

This of course requires each request to have been sent with a unique MessageId, and any intermediate server programs to process the Report options properly.

Note the design consideration in this case, that replies received after the requesting program has finished can remain unnoticed on the reply queue. Use of a shared reply queue in this way would need to have designed a convenient way to remove replies that are no longer wanted.

Class of service

A Reply Queue Alias would typically be specified in the Message Descriptor, and thus allow a class of service for replies to be determined by configuration instead of coding an explicit Reply Queue Manager.

Note that this name can not be opened for INPUT though; you would need the resolved name for that.

Asynchronous

Handling replies in a separate process from requests is less simple for the application, but its uses can be more general.

- Consider triggering the reply queue process.
- This approach works well with a permanent dynamic queue too. The queue that follows the naming convention is the model queue.

  A permanent dynamic queue should be deleted when all its messages have been processed, but it can remain in existence due to a failure. Consider specifying a Retention Interval. It can be used, in combination with Creation date and time, to highlight a dynamic queue which had not been deleted in a reasonable time. It would still need some administration process to remove such unwanted queues.

Design for old replies

These occur when a requesting program has a time limit to wait for a reply message. If a reply arrives after that time, the application must be designed so that such messages are either discarded or processed later.

Dynamic Queues

When MQSeries creates a dynamic queue, the first part of the resulting queue name can be controlled through the Object Descriptor. The appropriate name standard depends on the type of dynamic queue created.

Temporary - accept the MQSeries default

The MQSeries default for a dynamic queue prefix is 'CSQ.*' on MVS, 'AMQ.*' on other systems. Since temporary dynamic queues are deleted on MQCLOSE, they will not have to be controlled by the ***MQA***; so leave the default unchanged.

Permanent - supply an application prefix

A permanent dynamic queue can remain across application invocations. It may need to be managed by an ***MQA***, so ensure the queue follows the hierarchical naming convention. Specify an application prefix in MQOD.DynamicQName, followed by an asterisk.

Note that this application prefix must not exceed 32 characters, in order that MQSeries may generate a unique name with the remaining characters.

Queues for Bridges and Links

Include the bridge or link type in the application hierarchy.

For example,

<application>.IMS

<application>.CICS

<application>.R/3

Namelists

Use a hierarchical name as before

Don't indicate in the name that it is a Namelist; they have a separate name space, and so the fact that they are Namelists is completely clear from the context.

Triggering

You do not need to have triggering in all cases. For example a program could instead be scheduled in other ways - for example on demand, at a time of day, or as part of the system start.

Programs

Write programs to recognize whether they have been triggered

This recommendation applies even if the immediate intent is to schedule a program without triggering. It requires little extra code, and gives the application an ability to be scheduled differently in the future, without having to revisit the program logic.

- A program initially written to be invoked from the command line can subsequently be configured for triggering.
- A function designed for an automated set of application processes can be invoked as a stand-alone task.

Remember that triggered programs must tolerate finding an empty queue; there are conditions that generate an extra trigger message rather than risk missing a trigger.

A tip to avoid timing problems, particularly when using groups and segmented messages, is to specify a longer Wait Interval for the initial MQGET in a triggered program.

Process

If a queue has its own Process, use the same name as the queue

Include the version suffix if the queue has one; there may in any case be a separate executable for each instance of the queue.

Note that Processes have an independent name space. Hence there is no value including the fact it is a PROCESS as part of the name.

If a Process is shared, describe the collective function

Where several queues are handled by a common program, define a single Process object. Use a suitable hierarchical name for the collective function.

If multiple versions of a queue are read by the same program, just drop the version suffix from the queue name.

Use Environment Data as a parameter to the trigger monitor

This particularly applies if writing your own Trigger Monitor.

User Data was intended to be used as parameter information to the triggered program; Trigger Data similarly provides a parameter that is specific to one queue.

All fields are passed to the program in any case, but the original intent for the separate Environment Data was that it could be a parameter to control the function of a trigger monitor.

Some supplied trigger monitors do not use this information. On OS/400 it can be used for example to select a job priority, or CICS region, for the task that gets run. On UNIX, a value of '&' causes the program to be triggered as an asynchronous process.

Initiation Queue

Use system defined queues for simple general triggering

Some platforms define standard initiation queues when a queue manager is created. These are the defaults for supplied trigger monitors. For example,

SYSTEM.DEFAULT.INITIATION.QUEUE

SYSTEM.CICS.INITIATION.QUEUE

Where these are created, and triggering requirements are simple, the best approach is to use the supplied initiation queue.

Otherwise, use a hierarchical name

A reasonable approach may be to have an initiation queue for the various functions in an application. Then use a name of the form,

<application>.INITQ

Hint - to stop any trigger monitor, disable GET for its INITQ

Trigger monitors are designed to be long running. They will stop when MQSeries or the systems ends; or the trigger monitor task can be cancelled by an operator.

MQSeries for MVS/ESA provides an interface to stop its CICS Task Initiator function cleanly, without disrupting other operations. A more general way to close a trigger monitor, in any environment, would be to disable GET on the Initiation Queue; it works where trigger monitors allow shared input too.

Trigger Control

For temporary disabling use NOTRIGGER

This was the intent of this parameter, when there is an application need to suspend triggering temporarily. (Compare this with the operation of STOP CHANNEL for example.) Use trigger type NONE for a queue that must never be triggered.

Avoid trigger type DEPTH

The original intent of this feature was to support consolidation of replies to related parallel requests. The reply queue for the set of related messages would be a permanent dynamic queue, triggered when all the replies had arrived.

The main problem is that this type of triggering is disabled when the trigger occurs. There is no automatic re-triggering if all the messages are not processed. This simple approach does not cater for cases where replies are incomplete within a time limit.

Never use trigger type DEPTH to monitor a queue threshold. The correct way to do that is using Performance Event messages.

Avoid trigger type EVERY

This might appear suitable for triggering transactions that each process just one message. The design was not originally in response to any known user requirement.

The problem occurs when the system is re-started and there are several messages recovered on a queue. Only one trigger is generated no matter how many messages are on the queue.

A preferred approach is trigger type FIRST, and write applications to continue processing more messages.

If a transaction really must process only one message, trigger type FIRST is still easier to get right. At least it would leave no messages untriggered, because closing a queue with any remaining messages results in another trigger.

Achieve parallel execution if needed through a user written trigger monitor; or have multiple queues.

Take care with groups and segmented messages

MQSeries Version 5 introduced Groups and Segments, and there are options on MQGET to wait for a complete collection of physical messages.

Triggering is still based on physical messages though. An application would be triggered when the first physical message arrives, but may find no messages available if using these new options.

You may need to wait longer when an application expects a complete group or logical message. This would be needed to avoid a triggering loop.

Programming Conventions

Accept queue manager and input queue name as parameters

As explained earlier, it enables a program or transaction to be run unchanged, and take input from any queue, and on any appropriate queue manager.

Test for Completion and Reason Codes

The purpose of having separate return values is that Completion Code offers a simple test of whether the MQI call worked at all; Reason Code gives the specific cause.

Test for any reasonably anticipated Reason Codes; report any others as a number.

Similarly, when processing a Reply Queue, check for Report Messages; treat the MQMD Feedback values in the same way as Reason Codes.

Detect the condition of a queue manager quiescing

The purpose of a quiesce mode of stopping a queue manager is to allow applications to end cleanly. Request FAIL_IF_QUIESCING where MQI provides this option. Always use this when MQGET has the WAIT option.

The exception is when using MQI to finish a transaction already in progress. Specify MQGMO_FAIL_IF_QUIESCING on the MQGET which starts a new transaction; then omit the option on further MQI calls needed to complete the unit of work.

Avoid repeated MQCONN and MQOPEN

Most MQSeries implementations particularly optimize the performance of MQGET and MQPUT where possible by having work done in the earlier calls. It is therefore more efficient to issue MQCONN and MQOPEN, and then use the resulting handles to process several messages where possible.

Take particular care when MQI calls are grouped to form a higher level function. Some user implementations of such functions have led to repeated MQCONN or MQOPEN.

Generally use MQCONN

Most environments require an MQCONN call anyway. If MQCONN is called from an environment that is already connected, like CICS/ESA or a program called synchronously in the same process, MQCONN will complete quickly. It returns the connection handle that already exists, and a Reason Code of MQRC_ALREADY_CONNECTED. Hence its use can be appropriate in all environments.

MQSeries for OS/400 performs an implicit MQCONN whenever MQOPEN is called without having first connected to the queue manager. In this case there is an implicit MQDISC when the last, or only, queue is closed. This can result in multiple MQCONNs in a program.

Use ***CCTMQM*** in an interactive environment, or for CL programs that invoke MQSeries commands. This establishes an MQSeries connection, and so precludes an overhead for repeated implicit MQCONN in that environment.

Use default priority for a new message

The intended basic convention for a new message, like a Request, was to use the queue defaults for persistence and priority. This would allow tuning to be performed readily in the queue configuration, rather than in the program; Alias queues could be used for messages of differing characteristics.

This is sound advice for Priority, but not for Persistence. An application would generally know whether the messages it originates need to be persistent, so an explicit MQMD option is quite reasonable. On the other hand there have been reported cases of lost messages, where remote queue definitions had incorrectly specified DEFPSIST(NO).

Select the Report Options required

The default is that MQSeries does not send a Report message to indicate an asynchronous exception.

If any Report Option is specified (or the message is a Request):

- Specify a Reply Queue in the Message Descriptor
- Clear the Message ID, so that MQSeries generates a unique identifier for the message.

Always specify MQMD.Format

Even where not immediately needed, there is no harm in doing this. The default is that the message format is ***undefined*** to MQSeries. That could prevent a future need for message data conversion, and can cause some applications to fail.

The associated representation fields are usually safe to leave as the default. An exception is where applications operate using a different CCSID from the queue manager, and must therefore specify the correct value in the Message Descriptor. Take care with certain workstation COBOL implementations that offer an option of using mainframe or workstation data representation.

Generally specify CONVERT on MQGET

This is the preferred way to perform a basic message conversion, like character strings, between disparate platforms. (More complex data transformation is provided by certain tools, like MQIntegrator; that is separate from this discussion.)

The message is converted only if necessary, and at most once. It also applies when MQGET is performed by an MQ-client.

A message whose conversion fails sets the MQMD Encoding and CCSID to the actual unconverted representation. Therefore reset these values before each MQGET.

Take care with unlimited GET WAIT

This is necessary with certain long running programs, like Trigger Monitors. For most applications it would be better to set some time limit; then take some other action, or close down and wait to be triggered when a message does arrive.

Removing a bad message

This is a common design question. A unit of work is driven by an input message but subsequently fails. Actions already performed should not be committed; but rolling back the transaction would leave the message remaining on the input queue, and prevents an error response being MQPUT under syncpoint.

MQSeries for MVS/ESA provides an MQGMO_MARK_SKIP_BACKOUT facility. It is the ideal way to program for this case.

The more general technique, requiring multiple MQGETs but available across platforms, involves testing the Backout Count on any message retrieved.

Allow for bigger messages

A common error is to make incorrect assumptions about the required buffer size. The arrival of a production message bigger than any tested then causes an application error.

- If the application processes messages of a limited size, the simplest approach is to specify the Accept Truncated Message option to remove bigger messages put on the queue in error.
- An application that processes messages of variable size should not use this option as a rule. A message too big for the supplied buffer thus remains on the queue, and MQGET returns the required Data Length. The program needs to be prepared to re-allocate a larger buffer, up to a reasonable limit, and then do the MQGET again.
- The arrival of over-sized messages in a queue can be prevented by using the Maximum Message Length attribute of the queue. Changing this attribute does not affect messages already on the queue though, so using this value to determine a buffer size would not entirely remove the need for an application to allow for bigger messages.

Don't assume a fixed output queue to send results

An initial implementation may involve communication between just two programs. For example A sends a request to B; B replies to A.

Rather than sending the reply to a fixed queue name, make the program more general by sending the result to the Reply Queue instead. Similarly don't assume the reply is local, but at the Reply Queue Manager.

Reply with like characteristics

There are several conventions when replying to a message.

- Generally reply with like characteristics such as persistence or priority. (Consider using the same MQMD for input and reply for example.)
- When passing context, specify Pass All Context if a message is forwarded unchanged; Pass Identity Context if the reply is the result of some processing.
- Process Message ID and Correlation ID as specified in the Report Options. Don't assume the standard convention - copy Message ID to Correlation ID, and request new Message ID.

  The Report Options are generally removed for the reply message.
- Where the request was sent with an Expiry value, it would be received, if not already expired, with an Expiry value which represents the amount of unexpired time remaining. A Reply with "like characteristics" would therefore imply a response message with an Expiry value.

  A design consideration is whether this is appropriate for the application. An alternative convention, when a message has been processed, is to send the reply with Unlimited Expiry instead. This is the convention used when MQSeries sends Report messages.

  If, instead of a Reply, the message is to be transferred to another queue, forward it with the Expiry value that was read - it will be either Unlimited, or the amount of unexpired time.

Avoid long-running units of work

Performance can be degraded as the duration of a unit of work becomes longer; and keeping them short allows a queue manager to quiesce faster.

Make MQDISC conditional

This is related to the earlier convention of including MQCONN. Call MQDISC before ending the program, but not if MQCONN had earlier returned with Reason Code MQRC_ALREADY_CONNECTED. This approach is appropriate in all environments.

Sending your comments to IBM

MD01: MQSeries - Standards and conventions
Version 1.0

MD01 SCRIPT

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book. Please limit your comments to the information in this book and the way in which the information is presented.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

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

You can send your comments to IBM in any of the following ways:

- By mail, use the Readers' Comment Form.
- By fax:
  - From outside the U.K., after your international access code use 44 1962 841409
  - From within the U.K., use 01962 841409
- Electronically, use the appropriate network ID:
  - IBMLink: IBMGB(TSCC)
  - Internet: tscc@hursley.ibm.com

Whichever you use, ensure that you include:

- The publication number and title
- The page number or topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

Readers' Comments

MD01: MQSeries - Standards and conventions
Version 1.0

MD01 SCRIPT

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

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

Be sure to print your name and address below if you would like a reply.

Name	Address
Company or Organization	
Telephone	Email

You can send your comments POST FREE on this form from any one of these countries:

Australia	Finland	Iceland	Netherlands	Singapore	United States of America
Belgium	France	Israel	New Zealand	Spain	
Bermuda	Germany	Italy	Norway	Sweden	
Cyprus	Greece	Luxembourg	Portugal	Switzerland	
Denmark	Hong Kong	Monaco	Republic of Ireland	United Arab Emirates	

If your country is not listed here, your local IBM representative will be pleased to forward your comments to us. Or you can pay the postage and send the form direct to IBM (this includes mailing in the U.K.).

1 Cut along this line

2 Fold along this line

By air mail
Par avion

IBRS/CCRI NUMBER: PHQ - D/1348/SO

NE PAS AFFRANCHIR

NO STAMP REQUIRED

IBM

REPONSE PAYEE
GRANDE-BRETAGNE

IBM United Kingdom Laboratories Limited
Information Development Department (MP 095)
Hursley Park
WINCHESTER, Hants
SO21 2ZZ United Kingdom

3 Fold along this line

From: Name ______________________
Company or Organization ______________________
Address ______________________
____
EMAIL ______________________
Telephone ______________________

1 Cut along this line

4 Fasten here with adhesive tape