MD17: WebSphere MQ for z/OS Highly Available System Design Version 1.0

Document:	MD17
Issued:	January 2006
Revision Date:	January 2006
Previous Revision:	None

WebSphere MQ for z/OS Development
IBM United Kingdom Laboratories
Mail Point 127
Hursley Park
Winchester
SO21 2JN

Before using this document be sure to read the general information under "**Notices**" on page 5.

First Edition (January 2006)

Contents

Figures

Notices

The information is not intended as the specification of any programming interfaces that are provided by WebSphere MQ. Full descriptions of the WebSphere MQ facilities reported are available in the product publications.

References in this report to IBM products or programs do not imply that IBM intends to make these available in all countries in which IBM operates.

Information contained in this report has not been submitted to any formal IBM test and is distributed “asis”. The use of this information and the implementation of any of the techniques is the responsibility of the customer.

Trademarks and service marks

The following terms, used in this publication, are trademarks of the IBM Corporation in the United States or other countries or both:

- IBM
- z/OS
- zSeries
- WebSphere
- CICS
- IMS
- DB2

Other company, product, and service names may be trademarks or service marks of others.

1 Introduction

1.1 The aim of this SupportPac

This SupportPac provides recommendations and best practices for the design and implementation of highly available systems using WebSphere MQ on z/OS. While it is envisaged that a lot of the information will relate to the implementation and use of shared queues, there will also be information that is valuable to a non-shared queues environment.

1.2 Intended audience

The intended audience is anyone who is interested in providing a highly available MQ environment on z/OS. There is information that is particularly relevant for System Architects, System Administrators and Application Developers.

1.3 What is meant by High Availability (HA)?

A highly available MQ environment is one that is able to provided the required services to the applications using it to meet the business needs and the service level agreements (SLAs) that are in place. The SLAs would place stringent requirements on the MQ environment for both the response time of messages and also the downtime of the environment. The goal is to be able to provide the required services and meet the response time targets when:

- there is a planned outage
- there is an unplanned outage (minimal interruption to service)
- there are alterations being made to the environment

1.4 Layout

This SupportPac is arranged into two main sections. The first of these contains additional information about WebSphere MQ, including more low level detail that is not found in the product manuals. The second section is in the form of a number of specific questions and answers, these will guide the reader in areas that should be considered when implementing a HA environment. This structure lends itself well to additional information being added when required. In particular, when additional questions are identified the second section will be updated accordingly. If you have any questions related to HA that you would like answered in this SupportPac then please submit them and we will endeavor to include them in a future update. Questions and feedback may be submitted using the feedback form on the download page for this SupportPac.

2 General guidance and information

2.1 Introduction to Shared Queues

A *shared queue* is a type of local queue. The queue may be accessed by multiple queue managers that are in the same sysplex. The queue managers that can access the same set of shared queues form a group called a *queue-sharing group*.

Using a shared queue means that the messages can be placed onto the queue by any queue manager in the queue-sharing group, and also retrieved from any queue manager. The messages placed onto shared queues are stored in the zSeries Coupling Facility, in contrast to messages on private queues which are on DASD pagesets. Figure 1: A queue-sharing group shows three queue managers (QM1, QM2 and QM3) in a queue-sharing group (QSG), all with access to the same shared queue (QREQ) being processed by the server applications.

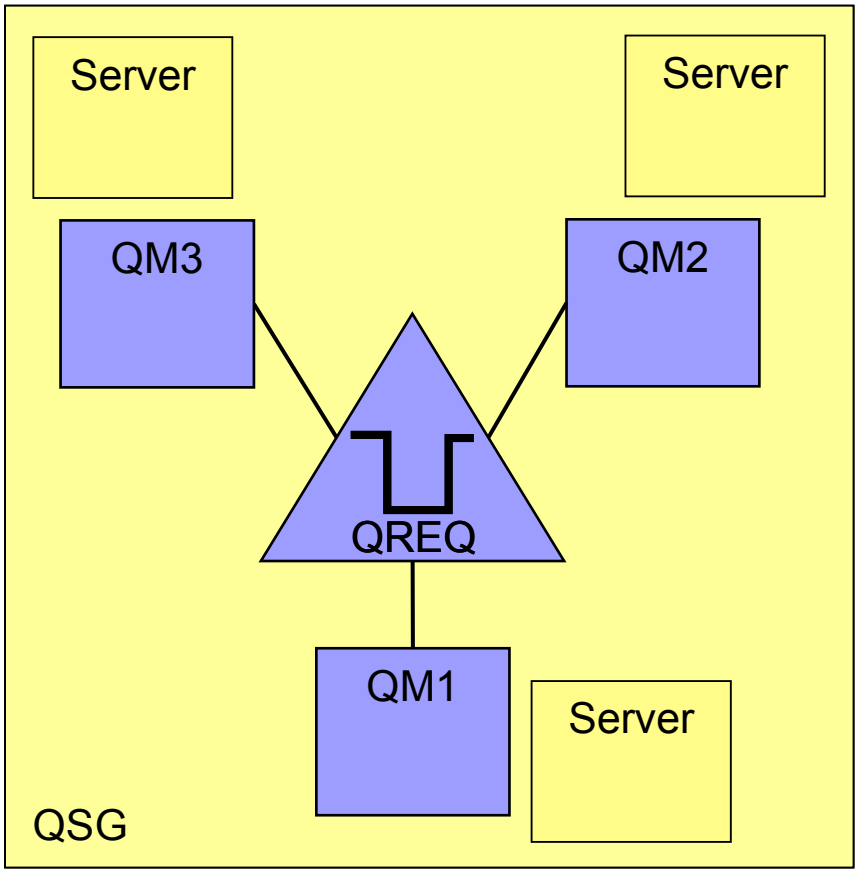

Figure 1: A queue-sharing group

When using private queues the messages on those private queues are only accessible if the queue manager hosting the queue is active. If the queue manager terminates then the messages on the private queues will not be accessible until the queue manager is restarted. By contrast, because a shared queue is accessible from all queue managers in the queue-sharing group the messages are still accessible even if a particular queue

manager terminates. This functionality makes it possible to design systems that minimize the impact of the unavailability of a queue manager and enables to processing of messages to continue.

The use of shared queues, however, is not without cost. The medium that is used to store the messages is very different when comparing private and shared queues. This gives rise to a number of considerations which include whether the benefits of using shared queues for a particular application are justified compared to the cost, and whether the application being considered is suitable for implementation in a shared queues environment given the amount of storage that is available. Consideration of differing application types and whether they are suitable for shared queues is made in Application Guidelines on page 15.

2.2 Coupling Facility Technology and Shared Queue Implementation

In this section we will consider the coupling facility technology that is available on z/OS and how MQ exploits this to provided the shared queues functionality.

Introduction to the Coupling Facility

The most basic way of thinking about the Coupling Facility is to consider it to be just a large area of memory shared between all the LPARs in the sysplex. It is of course a lot more complex than that, but is a good place to start. Users of the coupling facility do not access the storage directly, as they would normal memory, but use the coupling facility functions to store, read and manipulate data. The storage in the coupling facility is divided up into smaller areas called structures. There are three different types of structures that can be defined, these being “Cache”, “Lock” and “List” structures, the latter being the only one used by MQ. Coupling facility structures are created by making a coupling facility resource management (CFRM) policy definition that specifies the name, size and attributes of each structure – see “*z/OS V1R6.0 MVS Setting Up a Sysplex” {1} for further information*. All three types of structure can be defined in the same coupling facility and may be accessed by different applications. For example, both cache structures and list structures may be defined in the same coupling facility and used by DB2 and MQ independently.

MQ's use of list structures

WebSphere MQ has two uses for list structures when using shared queues. The first use is the Administration structure (CSQ_ADMIN) which is used by the queue managers in the QSG to ensure integrity of data when recovery is required, coordinate serialized access to queues and generally manage the shared queues environment. The second use for the list structures is as Application structures. These are the structures where actual MQ messages are stored, they are called application structures because they are primarily used by applications for message queues. Using list structures to store the messages means that MQ can take advantage of many of the facilities provided by the Coupling Facility to efficiently implement queuing primitives, including; the ordering of messages, atomic updates to the data and list depth monitoring (for triggering and get-wait processing).

Each shared queue is represented in the list structure by a separate list, up to 512 queues

can be defined in each structure. Each message is stored in the list as a list entry with one or more list elements containing the actual message data. The list entries are each assigned a key, which enables the list to be automatically sorted according to priority and age. The format of the key used when putting the messages means that when subsequently getting a message, the selection of the message can be performed by the CF, rather than the queue manager, which will return the highest priory oldest message that matches the selection criteria (assuming the queue is defined for priority order).

Most of the management of the coupling facility structures is performed using the standard coupling facility commands which enable many of the attributes (size, location etc) of the structures to be altered while the queue managers are still connected to the structures. More details on this is provided in Operational Practices on page 18.

Availability of structures

The structures defined in a coupling facility are by nature very reliable. The coupling facility is implemented in a combination of hardware and microcode. There is no user code that runs in the CF. It is possible to configure the CF to have multiple links to each machine that accesses it, thereby providing redundancy. However, where ultimate availability is required it is possible to duplex the structures that are defined in a CF. This means that two copies of the structure are maintained and in the event of a failure of one of them the other may continue to be used to provide access to the data. The duplexing of the structures is performed at the definition stage of the structures and is not something that MQ is aware of. It should be noted that while this provides an even greater level of availability there is also an additional overhead in terms of message throughput and CPU cost. In most circumstances duplexed structures are not required.

2.3 Shared Queues Configurations

In this section we will look at various possible configurations when using shared queues and consider aspects like triggering and clustering.

Simple private vs shared queue configuration

The first scenario to consider is shown in Figure 2: Comparison of private vs shared queue configuration. In this situation we have an initial configuration that is using private queues. There are no sequencing issues as we already have multiple instances of the server application that can run concurrently. From this configuration that is using a private queue with multiple servers we could migrate to using a shared queue with multiple servers, but also using multiple queue managers to provide access to the shared queue. The use of multiple queue managers enables the work to be distributed around the sysplex and provides improved availability as the requirement for a particular queue manager to be available is removed.

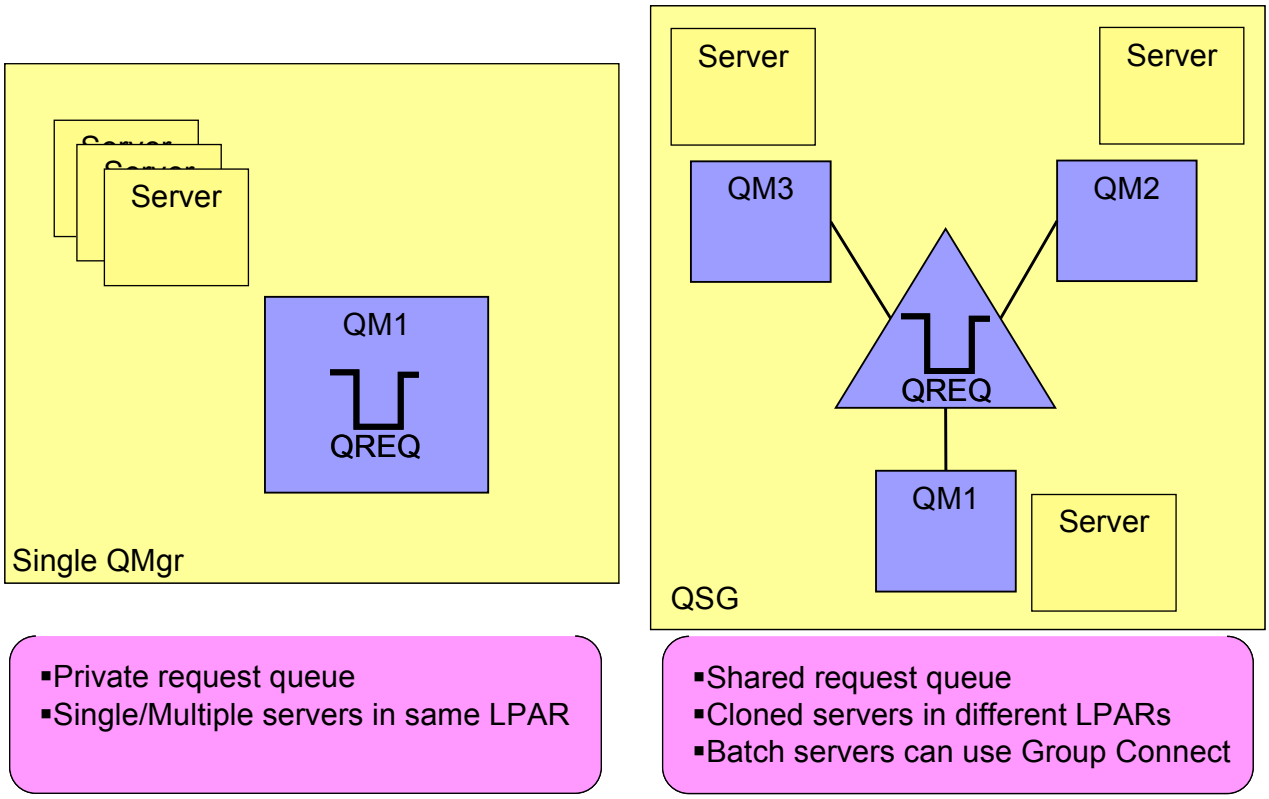

Figure 2: Comparison of private vs shared queue configuration

In subsequent scenarios we will look in detail at more specific areas of configurations like triggering and clustering.

Triggering with shared queues

When migrating triggered applications to use shared queues it is important to understand the triggering model that is used. This differs slightly from that used when triggering private queues, and also differs depending on whether trigger first/depth or trigger every is used. We will look in detail at both triggering models and the important considerations that need to be looked at in each case.

Triggering using trigger FIRST or DEPTH

The first important thing to note with trigger FIRST and DEPTH is that multiple trigger messages may be generated, this is shown in Figure 3: Triggering on shared queues using FIRST or DEPTH. With private queues a single trigger message would be generated and hence a single application started, but with shared queues multiple applications may be started with the arrival of a single message. This has both advantages and disadvantages.

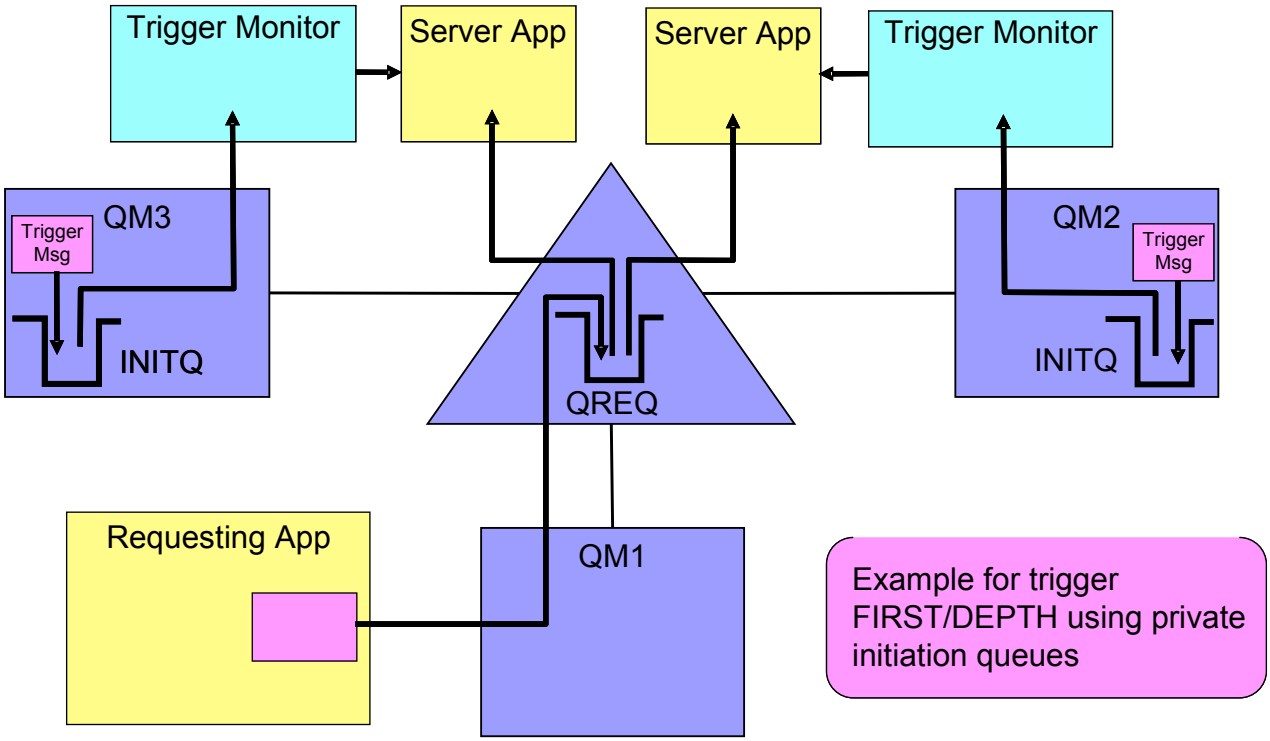

Figure 3: Triggering on shared queues using FIRST or DEPTH

The advantage of multiple trigger messages being generated is that servers may be started on multiple systems and the workload of processing the queue spread around the QSG. This is of particular benefit if more messages arrive on the queue shortly after the trigger is generated as the other servers that have been started can also process messages. Of course, if the queue is set to trigger FIRST and only one message arrives then there may be servers that are started and do not process any messages, this has an associated cost in starting multiple applications that don't actually perform any useful work.

When the number of messages on the queue meet the triggering requirements (0 to 1 for trigger FIRST or n-1 to n for trigger DEPTH) then a CF list structure exit is driven by the CF in the queue manager address space, this is driven in each queue manager in the QSG. Each queue manager will then check the triggering requirements independently of the other queue managers and generate a trigger message as appropriate.

Triggering using trigger EVERY

Trigger EVERY differs from FIRST and DEPTH in that only a single trigger message is generated each time the triggering rules are satisfied. When a message is put to the shared queue that is set to trigger EVERY, the queue manager where the put is occurring nominates a queue manager in the QSG to generate the trigger message and put it onto the trigger queue. This situation is shown in Figure 4: Triggering on shared queues using EVERY where the requesting application is connected to QM1 and then QM3 is nominated by QM1 to put generate the trigger message. The placement of the INITQ will affect where the server application is started. If each queue manager has a private INITQ then the server application will start on the queue manager that was nominated to generate the trigger message. If the INITQ is shared then there will be a race between the

trigger monitors to get the trigger message and start the server, this will mean that the application can start on a different queue manager than where the trigger message was created.

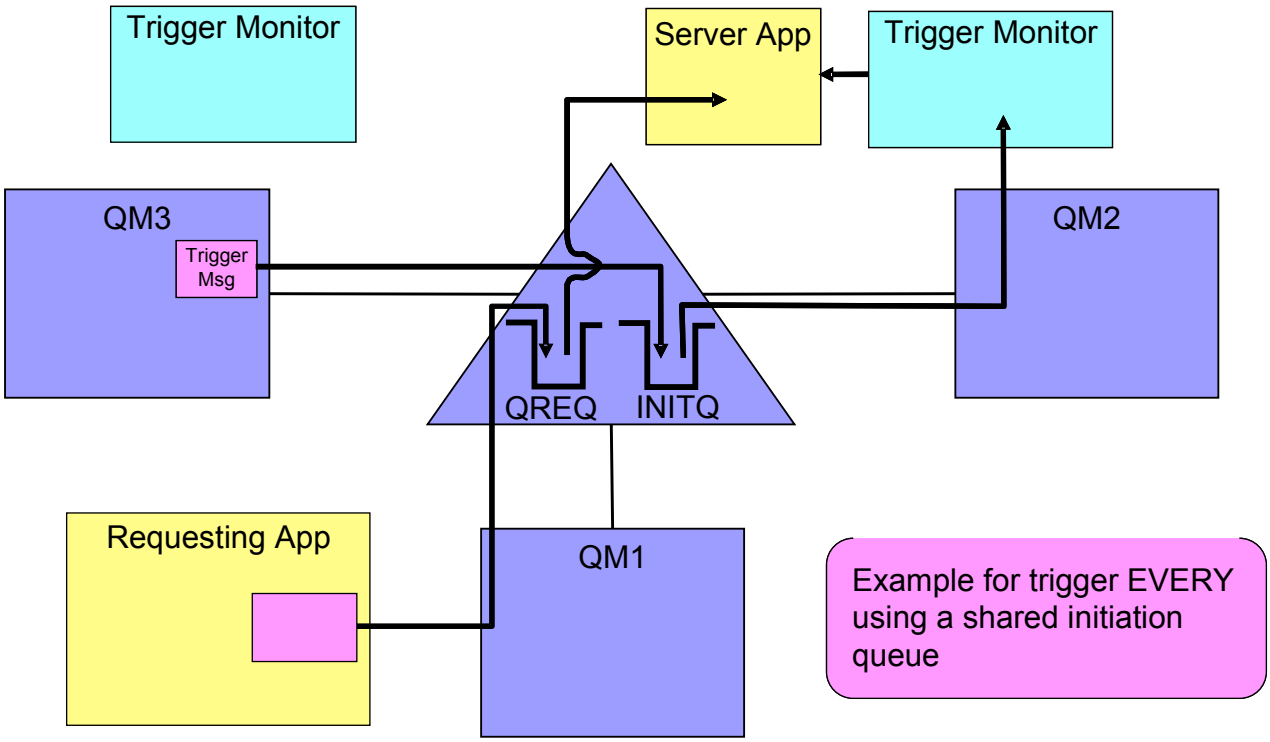

Figure 4: Triggering on shared queues using EVERY

Clustering

Just as private queues may be defined to be in a cluster, shared queues may also be defined to be in a cluster. The shared cluster queue would be advertised to the cluster by each queue manager in the QSG that is also in the cluster. This means that if there are three queue managers in a QSG that are also in the same cluster, and a shared queue is defined to be in the cluster then there will appear to be three instances of the queue to the rest of the cluster. Even though there appears to be three instances these all end up with the message being put to the same shared queue. This situation is shown in Figure 5: Clustering using private and shared queues.

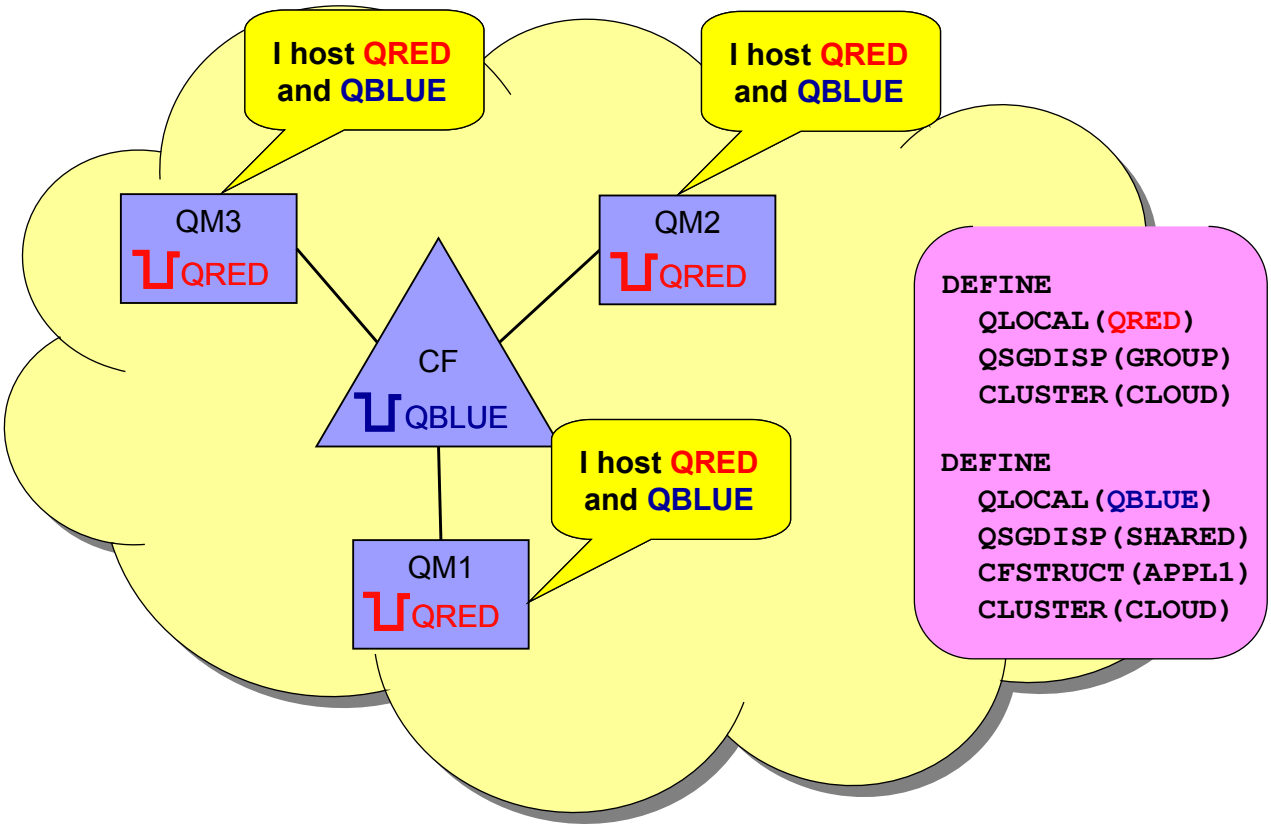

Figure 5: Clustering using private and shared queues

It should be noted that it is not possible to define a clustered channel to also be a shared channel, and by implication it is not possible to have a cluster repository hosted by the QSG rather than a particular queue manager.

Using shared channels

With traditional MQ channels, the channel would always start into (or out of) the same queue manager. When operating in a QSG environment it is possible to configure MQ channels to be shared. Shared channels can be setup for both inbound and outbound communications, this means that the channel can start using any available queue manager in the QSG, rather than always being fixed to using the same queue manager. This provides greater availability for messages being delivered into the QSG or sent out of the QSG than that available just using channels that are private to a single queue manager.

A shared channel is shared if:

- for an inbound channel, it is started via an inbound request which targets a shared listener port (ShrP in Figure 6 below)
- for an outbound channel, it is defined as having a transmission queue which is itself a shared queue.

In both cases, the shared channel saves state in a shared sync queue, `SYSTEM.QSG.CHANNEL.SYNCQ`. The remote system is not aware of the individual Queue

Managers, it just 'sees' and synchronizes with a single highly available 'queue manager', the queue sharing group.

The shared channel may be started on any available Queue Manager in the QSG, so we need to ensure that the same channel definition is available on each Queue Manager. A convenient way to do this is by defining the channel with a QSGDISP of GROUP, so that each queue manager maintains an identical definition.

In Figure 6: Using shared channels for inbound availability and balancing we can see the use of shared inbound channels to provide for workload balancing of the channels that start and also availability. In this scenario the target queues (QREQ) are actually private queues on each queue manager (QM1, QM2 and QM3). When a channel is started it can be routed (using a suitable routing mechanism like Sysplex Distributor or VTAM generic resources) to any available queue manager in the QSG where the message can be processed and any necessary response created. Of course, in this case, if a message is placed on the request queue and there is a failure before the message is processed the message could not be processed by another queue manager in the QSG as it is held on a private queue. This scenario would be similar to using clustering, except that once a particular channel has started all the messages that are placed onto the xmitq for that channel will be delivered to the same queue manager (unless the channel restarts).

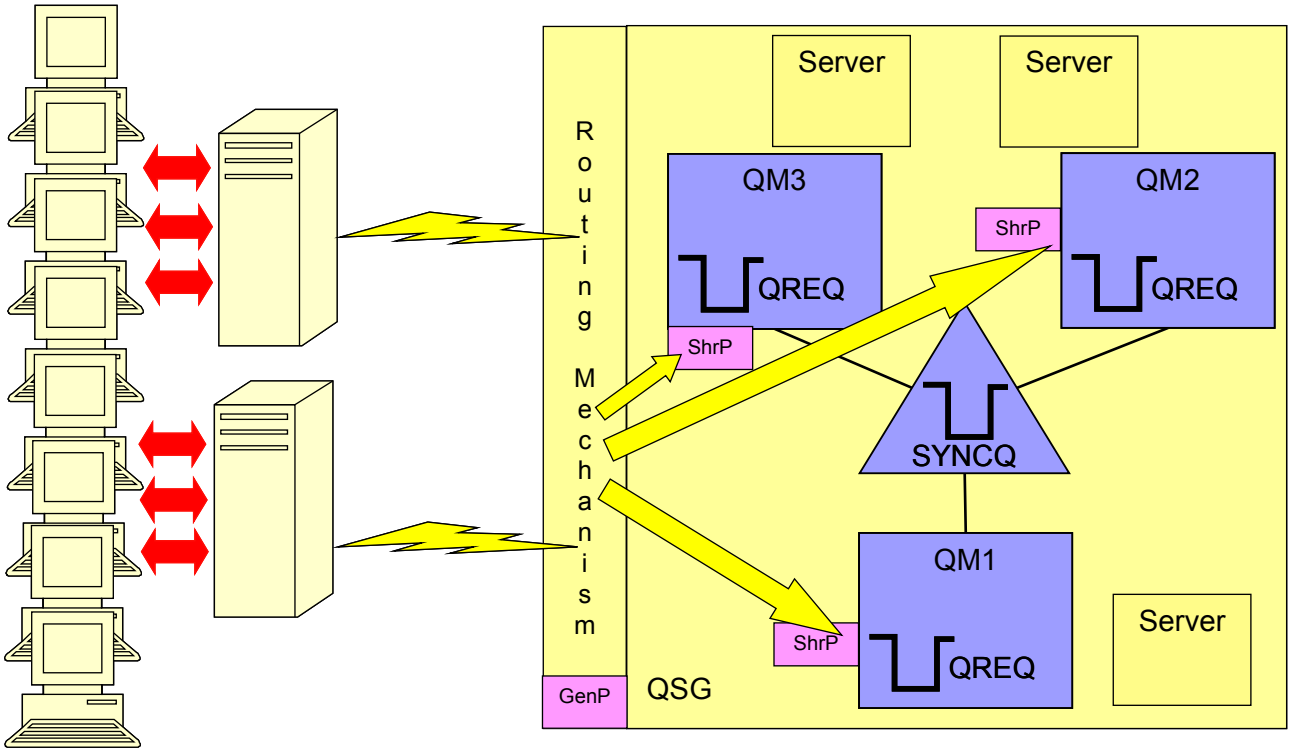

Figure 6: Using shared channels for inbound availability and balancing

In Figure 7: End to end solution providing high availability via shared queues and channels an infrastructure that uses both shared channels and shared queues to provide a highly available system is shown. Both inbound and outbound availability/redundancy is provided via the use of the shared channels, this means that no particular queue manager (QM2, QM3 or QM4) is required for the messages to be delivered to or from QM1. Also, once a

message has been placed on the request queue (QUERYREQ) it can processed via any of the queue managers in the QSG. This means that it is possible to take any of the queue managers in the QSG out of service at any point in time (assuming messages are processed under syncpoint control) without impacting the ability to process the messages in the system. In this example both remote queue and alias queue definitions have been used, this provides a means for being able to modify the underlying design while minimizing the impact and changes to the application and other MQ resources.

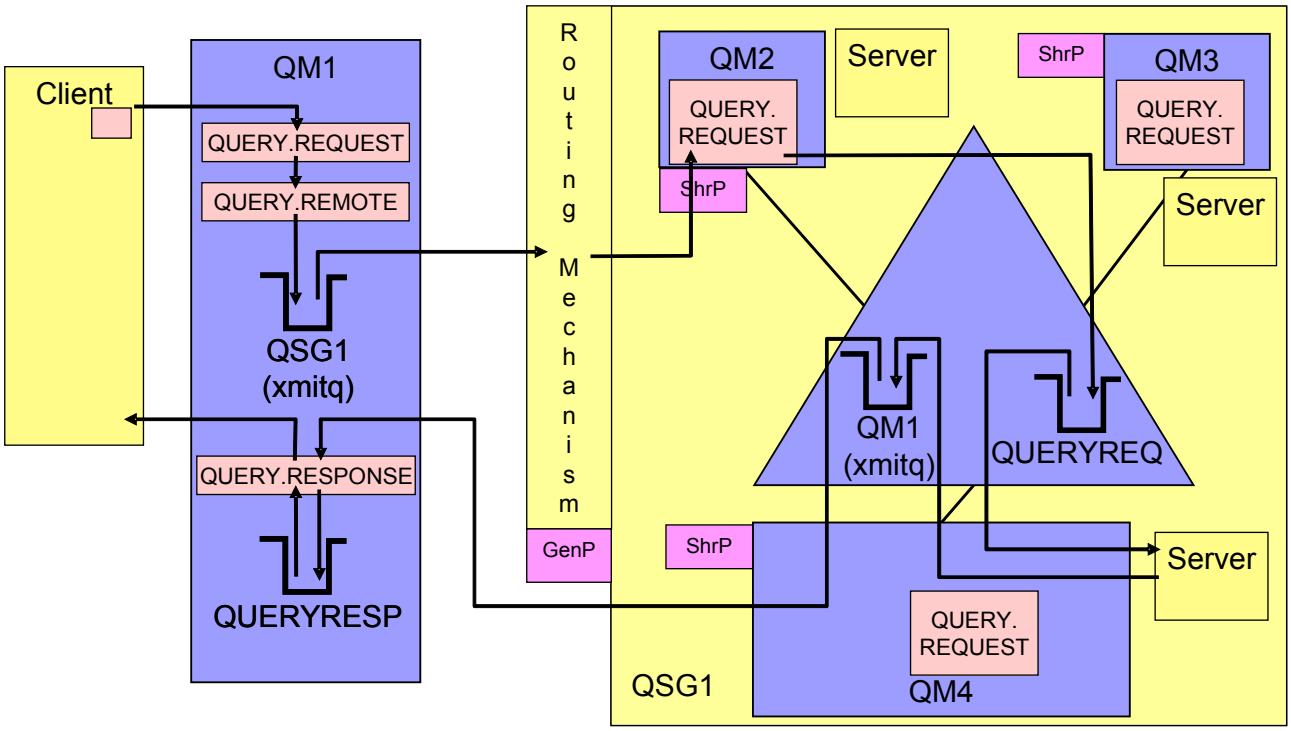

Figure 7: End to end solution providing high availability via shared queues and channels

2.4 Application Guidelines

An MQ infrastructure is in place to support the applications that require messaging capabilities. As such there needs to be a focus on the needs and requirements of the applications using MQ when considering a suitable infrastructure. This is particularly true when considering shared queues. When deciding whether to implement an application on shared queues it is necessary to identify the benefits that you want to achieve. After all, there must be a reason why the use of shared queues is being considered. When the aims have been identified it is then possible to determine whether the costs associated with using shared queues for a particular application are outweighed by the benefits.

If shared queues are being used then there is a wide range of considerations that can affect availability. Some of these will just affect the availability of a single application and others may have a more significant impact on the whole queue manager/queue sharing group. It is important to consider how different application designs can have an impact on MQ, both for shared queues, and also with private queues.

In this section we will look at a number of considerations when implementing applications, and how these can affect availability.

Change Control

Change control may appear on first thought a strange place to start when thinking about application design and the impact on availability. It is, however, one of the most important considerations to be made. There is a much greater probability of having a failure soon after making a change than once the change has been implemented for a period of time. Great care should be taken when changes are made to ensure that there is no adverse impact in making the change. This holds true for both application changes, and infrastructure changes.

Any changes should first be performed in a test environment, to ensure that it can be implemented with minimal impact. At the same time, precise documentation should be produced that details the changes that are made. It is advisable to maintain documentation not only of what is being changed, but also why the change is made, and why it is being made in a particular way. This will mean that in the future it is possible to see not only what changes have been made, but the process that was involved in the design. This is particularly valuable when reviewing architecture and making other changes as it means that the considerations and reasons behind a particular design can benefit any future designs.

Each company will implement this in a different way, depending on the companies working practices, standards and also culture. The key point though is that there is sufficient documentation to know what the configuration is, and the reasons why it is such. It should also be remembered that this documentation can be very useful in problem determination, especially if someone outside of the operational team needs to be involved. A better understanding of the environment can often speed up problem determination.

Unit of Work Size and Duration

The size and duration of a Unit of Work (UOW) can have an adverse impact on the running of a system. This is true for both shared queues and private queues. The two main areas that are affected by the UOW size are the availability of messages and the amount of recovery that is required in the event of a failure.

Any messages that are put or got inside syncpoint will require space on the coupling facility. They will take up this space from the time they are put, whether in or out of syncpoint, until the point at which they are got and the get has been committed. This means that the messages that are being processed by a long running UOW will be taking up space on the coupling facility. If messages are being put to be processed by another UOW then they will not be available to be processed until the putting UOW has been committed. The sooner this UOW can be committed, the sooner they will be available to be processed. Thus enabling the space in the coupling facility to be freed.

The duration of a UOW can have a significant impact on the amount of recovery that is required in the event of a failure. If there is a queue manager failure then it will be necessary to read the logs from the start of the oldest UOW that is still active in the queue manager at the point of the failure. The amount of log data that is required for recovery during restart is going to directly influence the time that restart is going to take. In fact, the UOW does not actually have to perform much work to adversely affect the restart time. If the UOW performs a single operation in syncpoint and then does not do any more MQ work for the next hour it would still be necessary to read the log data for the whole hour,

even though only a single operation had occurred.

For these reasons the size and duration of a UOW should be kept to a minimum, in particular you should avoid prompting a user for input while having a UOW in progress, if the user takes a long time to respond to the request then a long running UOW may well be created.

There are a variety of ways in which the size of a UOW can be reduced. The method that is employed depends on the exact operations that need to be performed. If a new application is being developed then it is important to give consideration to the size of UOW that may be created. For example, if an application needs to read records off an MQ queue and use the information in message to update a database, then there are two extremes that could be taken. The first would be to read a single message off the queue, update the database and then commit. The other extreme would be to read a message off the queue, update the database and repeat the operation until there are no messages remaining on the queue, and then commit. The second method may be fine most of the time if the number of messages on the queue is typically small, so a large UOW wouldn't be created, however, if there is an outage of the application processing the queue, then there could be a build up of messages. When the application is restarted, if it possesses all the messages on the queue a very large UOW could be created. The other alternative of processing a single message at a time might not be the best either as this would mean that the application would be committing frequently, which has an associated overhead. Of course, neither extreme is ideal and a balance needs to be found between having too short a unit of work, and too long a unit of work.

Sequencing Issues

There are instances where it is important to have an application that is highly available, but it also has strict sequencing requirements. There are therefore additional considerations that are required when moving to use shared queues. How is the sequencing going to be maintained on the queue? How can we ensure that there is not another instance of the application running on another queue manager in the Queue Sharing Group? What will happen in the event of a failure of the application or the queue manager that it is running on? All of these considerations are addressed by making the application a *serialized application*. This will ensure that only a single instance of the application can be run at any time in the Queue Sharing Group. Any application is serialized by using the MQCONNX[1] verb rather than MQCONN, and specifying in the MQCNO that serialization is required. A Connection Tag (ConnTag) also needs to be supplied to identify the serialization required. A single ConnTag can only be in use in the Queue Sharing Group by a single application at any time. Using this option ensures that only a single instance of an application can run at any time. If the application is processing messages in syncpoint then the ConnTag will remain in use until the application has disconnected and all the actions required for a Unit Of Work (UOW) have been completed. This means that if there is a queue manager failure while a serialized

1 There is a common misconception around the use of MQCONNX in different environments. While it is not necessary to use MQCONN in a CICS environment, there is absolutely nothing stopping the use of MQCONNX under CICS to serialize CICS transactions. It is of course important to check the return code from the MQCONNX call to ensure that the transaction has completed the MQCONNX successfully as the default MQHCONN is still usable after a failed MQCONNX call. Serialized applications are supported in all the environments that are supported on z/OS.

application is running then another instance of the application won't be able to connect to another queue manager until the UOW of the application that was running has been resolved. This ensures that the messages on the queue are always processed in the correct order, even in the event of a failure.

Batch v RealTime

Applications can be generalized into two main types, those that process messages as soon as they arrive (RealTime) and those that wait for a number of messages to arrive before processing them (Batch). The usage of MQ is different for these two applications, for the RealTime model messages are typically on the queue for a short period of time and a small amount of storage is required to hold the messages. For the Batch model the messages may be on the queue for a long time and a larger amount of storage is required.

While the Batch model could be used with shared queues, careful consideration needs to be given as to whether or not this is the best use of the available Coupling Facility storage. What benefits are achieved by using a Batch model with shared queues? Are these benefits sufficient to justify the use of the Coupling Facility to store the messages?

As RealTime applications are processing the messages as they arrive the depth of the queue will tend towards zero, this is as long as the potential rate at which messages may be processed is greater than the maximum arrival rate that occurs.

System Affinities

As far as possible you should try to remove affinities from your systems. Affinities can stop you being able to run instances of an application on multiple systems due to not all the required resources being available. In a shared queue environment if the affinities are removed then it is possible to move an application onto a different server if a failure occurs that means that application can't be run in the original location for a period of time. When thinking about the affinities that exist it is important to not just look at a single application but to consider all the applications, as one may place a restriction on others. For example, consider two MQ applications that run under CICS. One of them is designed to run on shared queues and does not have affinities to any particular queue manager and may run on any available queue manager in the QSG, the other application is not configured for shared queues and requires a specific queue manager to function. If the queue manager that is normally used by the CICS system is not available, would it be possible to connect the CICS system to another queue manager? Although the shared queues application could connect to the other queue manager the non-shared queues application would not function correctly and therefore it may not be possible to use a different queue manager, even for the application that does not explicitly have any restrictions itself.

2.5 Operational Practices

Resource Utilization

MQ Commands used in this section are fully documented in “*WebSphere MQ Script (MQSC) Command Reference*” {2}.

Regularly monitor the CF utilization

The `DIS CFSTATUS(*)` MQ command provides information about the size of the CF structures, and their current utilization. For example in the following output:

```
CSQM201I !MQ37 CSQMDRTC  DIS CFSTATUS DETAILS
CFSTATUS(APPLICATION1)
TYPE(SUMMARY)
CFTYPE(APPL)
STATUS(ACTIVE)
SIZEMAX(10240)
SIZEUSED(1)
ENTSMAX(2440)
ENTSUSED(39)
FAILTIME( )
FAILDATE( )
 END CFSTATUS DETAILS
```

SIZEMAX indicates the current CF structure allocation. This is the size in KB which was specified in the CFRM policy, but which may have been amended by the system as a result of auto alter processing.

SIZEUSED indicates the amount of the available space which has been used by MQ in that CF structure as a percentage of the SIZEMAX.

ENTSMAX indicates the maximum number of 'List Entries' there are currently available (roughly this corresponds to how many messages could be stored in total in the CF structure. MQ does make use of a capped number of list entries for internal use).

ENTSUSED indicates how many of the available list entries are currently used for holding message control information, or for MQ internal use. Note that this number will not immediately reduce when MQGETs are run against queues on the CF structure because MQ does some batching of deletes of messages for efficiency.

Equivalent historical information is available in SMF 74 subtype 4 records.

CF Structure Auto Alter

In the CFRM policy it is possible to enable a structure for auto alter. Having this option enabled allows the system to automatically adjust the size of the structure and also the entry to element ratio used.

Messages are stored in the CF Structure as a 'List Entry', which contains control information for that message, and 1 or more 'List Elements' which contain the message data. (By message data here we mean both MQ internal information, any MQ headers (eg MQMD, MQXQH) and user data). Each 'List Element' can hold 256 bytes of information. Empirical evidence suggests that messages stored on shared queues are, on average, between 1-2KB in size. So MQ asks the system to keep the entry:element ratio in the CF Structure at 1:6. This means that the CF will most efficiently store messages where each message has an associated 1.5KB (6*256) of data.

When messages are larger than this, the CF structure will tend to exhaust 'List Elements' as more messages are stored there, and conversely, if messages are smaller than this, the CF structure will tend to exhaust 'List Entries' first. In extreme cases, where the

average message size stored in the CF Structure is much smaller than this 1.5KB average, the CF structure will become 'full' due to running out of 'List Entries' for new messages.

Having auto alter enabled for the structure means that system can adjust this 1:6 ratio that is initially requested by MQ and use a ratio that is more suitable based on the sizes of messages in use on the structure.

Auto alter also means that the size of the structure will be adjusted automatically. The structure will start off being allocated with the INITSIZE specified in the CFRM policy but may be increased (or decreased if the CF is short of storage and needs it for another structure) when the structure is becoming full. If auto alter is being used then it is important to set the maximum size of the structure to a sensible size as otherwise it may keep growing up to the maximum set.

Feedback the CF Structure Utilization into the CFRM Policy

Over time, as new applications come on stream, and more queues are defined in an MQ CFSTRUCT, the utilization of the associated CF structure will tend to increase. To some extent, the system will hide this from you. Auto Alter processing will resize the CF structure within the limits set by the CFRM policy and the system utilization of Coupling Facility storage for other CF structures defined in the same CF. In a well behaved system, MQRC_STORAGE_MEDIA_FULL should be a rare event.

However, if there is a CF outage, which means that the CF structure has to be reallocated by MQ. The size of structure reallocated will be that defined in the CFRM policy. It may be the case that this initial allocation is not large enough to hold all the messages to be recovered. The rate of loading messages during a RECOVER CFSTRUCT operation is typically much faster than the AUTO ALTER processing can respond and resize the structure, which may lead to out of space errors during the RECOVER CFSTRUCT processing.

To avoid this situation, update the CFRM Policy so that the initial CF structure allocation (INITSIZE) is close to the usual SIZEUSED.

Structure Rebuild

WebSphere MQ supports System Managed Rebuild processing for CF structures. This means, that it is possible to non-disruptively move some or all MQ structures from one coupling facility to another, for example, to allow a planned outage of the coupling facility. For example, to move the structure SQ03APPLICATION1 into a different coupling facility named on its preference list in the CFRM policy, the following MVS command is issued:

```
SETXCF START,REBUILD,STRNAME=(SQ03APPLICATION1),LOCATION=OTHER
```

This also enables structures to be rebuilt if they need to be expanded above the maximum size defined in the CFRM policy. The first step would be to modify the CFRM policy with the new size for the structure, and where necessary updating the preference list to include the appropriate coupling facilities that may be used for the structure. Once this CFRM policy has been updated it can be activated, and then the structure rebuilt with the new attributes. This rebuild enables the structure to be enlarged (above the max initially in the CFRM policy) without needing to take the QSG out of service while the change is performed. It should be noted however that access to the structure will be restricted by the

CF during the period of the rebuild, so although the queue managers will not be aware of the rebuild taking place any application attempting to access the queues on the structure will be held up until the rebuild has completed. This would typically mean that there would be a number of seconds during which it would not be possible to put to or get from the queues on the structure.

2.6 Backup and Recovery

In the same way that it is necessary to regularly take backups of pagesets to ensure that messages can be recovered in the event of a DASD failure, you should also ensure that regular backups are made of shared queue messages in the CF structures[1].

MQ provides a command, `BACKUP CFSTRUCT(structname)`, which copies the current contents of the named CFSTRUCT to the log of the queue manager where the command is issued.

In the unlikely event of a coupling facility failure, shared messages can be recovered with the `RECOVER CFSTRUCT(structname)` command. Conceptually `RECOVER CFSTRUCT` has a number of phases:

1. Find the most recent structure backup, and restore it into the CF. The most recent backup may have been made on any queue manager in the QSG.
2. Forward recover the contents of the CFSTRUCT to the point of failure by application of log records. Again, any queue manager in the queue sharing group may have done a PUT or GET of a persistent message and recorded data about the operation on it's log. This means the forward recovery phase involves merging log data from all queue managers in the queue sharing group.
3. Tidying up the recovered state of the CFSTRUCT to ensure transactional integrity of transactions running at the time of failure.

In practice, `RECOVER CFSTRUCT`, works backwards as there are lots of optimizations that can be made, for example, there is not need to restore a message, if it has been subsequently got by a committed unit of work.

Take Frequent Structure Backups

The key point to make is that the length of time taken to recover the contents of a CF structure is overwhelmingly governed by the time to read and merge log data from queue managers which touched the CF structure between the time of the last backup and the time of failure.

If structure recovery time is important, you should ensure that structure backups are taken regularly and often.

(Maybe a small example will help convince you of this. Suppose your queue manager is logging persistent data at an average rate that is half that sustainable by your logging hardware, maybe 8MB/s. You took a structure backup at 08:00 this morning, and the CF failed 3 hours later at 11:00. Now let's imagine that all the log data from these 3 hours is available on your fastest disks, there is no contention, and you can read at the full

1 Unlike DB2, which only caches tablespace data in share buffer pools in the coupling facility, MQ stores shared messages ONLY in the coupling facility.

sustainable rate, eg 16MB/s. It will still take of the order of 1hr 30mins to recover the CF structure.)

The overhead of a structure backup has been designed to be low. Only persistent messages which have been resident on a shared queue for longer than a threshold interval are backed up. The threshold defaults to 30s, but can be specified on the `BACKUP CFSTRUCT()` command with the `EXCLINT()` parameter. Increasing EXCLINT will potentially reduce the size of the structure backup.

If the size and impact (in terms of log I/O or queue manager CPU consumption) of structure backup is too large, consider starting another queue manager in the queue sharing group whose sole function is to periodically perform structure backups.

2.7 Release Migration

MQ releases and maintenance are planned and shipped in such a way that it is possible to apply service to a queue manager, or, to migrate a queue manager to a new release, without requiring a concurrent outage of the entire queue sharing group.

In outline, this is achieved by a process of 'rolling' maintenance through the queue sharing group. For example, consider a QSG containing 3 QMGRs, A, B, C.

1. Initially they are all at release 1.
2. A new release of code, or libraries with maintenance applied, are made available by SMP/E.
3. QMA is stopped. It's STEPLIB changed to pick up the new code libraries.
4. QMA is restarted.
5. QMA is now operating in the QSG at a higher level of code than the other qmgrs. It can begin to pick up the workload.
6. Once QMA has proved to be stable at the new level of code, QMB can be stopped, it's STEPLIB libraries changed, and then restarted.

Release Migration and DB/2

This section describes how maintenance affecting the queue manager's connection to a data sharing group is shipped to support the 'rolling maintenance' scheme.

Manipulating the shared repository of queue sharing group data is performed by a few modules which contain SQL statements. When these are built, DBRMs are created which are shipped as part of the product or PTF, and installed into the SCSQDEFS library. The DBRMs must be 'bound', on the Data Sharing Group where the QSG is to run, into PLANS. The PLAN itself is held in DB/2. A load module, CSQ5PLAN, tells a queue manager what plan name to use for a given function.

When installing a PTF which changes a module which performs SQL, the following points pertain:

1. The PTF will ship the matching DBRM for the changed module
2. The PTF will contain a change for the CSQ5PLAN module so that it contains a new name for the relevant plan.

3. After installing the PTF, DB/2 still contains the old plan, so unserviced code still continues to run.
4. To enable the serviced level of the code to run, the DBRM from the PTF level of SCSQDEFS must be BOUND against the DSG to create a plan with a new plan name. The new plan name matches that held in the PTF level of CSQ5PLAN.
5. Hold data in the PTF, or installation instructions in the program directory, reminds you to perform this BIND step.

Queue Sharing Group Capability

There may be new capabilities or functions available to the QMGRs running the higher release of code which are still not available to the lower release ones. But all lower release function is available to all the queue manager.

Two principal mechanisms are used to support this migration:

The first is the CFLEVEL of MQ CFSTRUCTs. The CFLEVEL defines capabilities of shared queues defined in that structure, for example, support for persistent messages. Queue managers running with a level of code which cannot support that capability, are not permitted to connect to that structure – they don't know the rules for playing that game. Typically a new CFSTRUCT with the higher level of function can be defined and used from a higher function queue manager, but it is not possible to ALTER a CFSTRUCT to a higher level until ALL queue managers defined to the queue sharing group can support the function.

The second mechanism is not externalized. The queue managers maintain a minimum and maximum function level for the QSG. A queue manager is only allowed to join a QSG at startup if the level of code it is running is higher than the minimum function level of the QSG. Similarly, as it starts up, it may increase the maximum function level of the QSG.

Sometimes during the installation phase of a new release of code, it is necessary to run a utility which modifies the minimum and maximum function capability of the QSG.

3 Questions and Answers

3.1 How should I size my CF structures?

Details about how to size CF structures for use with MQ can be found in the SupportPac MP16 {3} and examples may be found in the developerWorks article “Best Practices: WebSphere MQ shared queues and application programs” {4}.

3.2 How can poison messages be handled?

A poison message is any MQ message that can not be processed correctly by the receiving application and is therefore backed out and remains on the application queue. This would typically be where an application removes the messages from the queue inside syncpoint control and then the application ends abnormally without having completed processing of the message. If the application termination is due to an error with the message then, if not handled correctly, the cycle of getting the message and ending abnormally might continue indefinitely. The application needs to be able to detect this situation and respond appropriately.

When a message is backed out the `BackoutCount` field in the MQMD is incremented, this enables an application to determine which messages have previously been got and subsequently backed out, and how many times this has occurred. Applications can cater for poison messages by taking into account the value of the `BackoutCount` field when getting messages. Initially the application may want to process a number of messages inside syncpoint control, so that it doesn't have to commit too often. It may, for example, get a message a process it, and repeat this up to ten times or until there are no messages left to process, and then commit. If the application got a message that it could not process, but did not cause it to end abnormally, it could put it to an error queue and carry on. If the application retrieved a message that caused the application to end abnormally, all the messages that had been retrieved up to that point inside syncpoint control would be backed out, and the `BackoutCount` would be 1 for each of the messages. The next time the application was started it would get a message that had been backed out, the application could take note of the fact that the `BackoutCount` was 1, and in this situation could just process that message before committing. If a message that had a `BackoutCount` of 1 was processed that caused the application to end abnormally again the message would be backed out for a second time and the `BackoutCount` would be 2. The next time the application got the message it would see that the message had a `BackoutCount` of 2 and rather than attempting to process the message it would move it straight to the error queue for the application. Having dealt with the poison message the application could return to processing number of messages before committing.

3.3 What should it do if my structure is low on space?

Firstly, check the CFRM policy for your structure, you can use the D XCF command to do this.

```
D XCF,STR,STRNAME=SQ27APPLICATION3

IXC360I  13.17.01  DISPLAY XCF 568
STRNAME: SQ27APPLICATION3
```



```
 STATUS: ALLOCATED
 TYPE: SERIALIZED LIST
 POLICY INFORMATION:
  POLICY SIZE    : 500000 K
  POLICY INITSIZE: 40000 K
  POLICY MINSIZE : 0 K
  FULLTHRESHOLD  : 80
  ALLOWAUTOALT   : NO
  REBUILD PERCENT: N/A
  DUPLEX         : DISABLED
  PREFERENCE LIST: P5CF01
  ENFORCEORDER   : NO
  EXCLUSION LIST IS EMPTY

 ACTIVE STRUCTURE
 ----------------
  ALLOCATION TIME: 10/14/2005 09:26:40
  CFNAME         : P5CF01
  COUPLING FACILITY: 002064.IBM.83.00000001169E
                     PARTITION: 0E   CPCID: 00
  ACTUAL SIZE    : 40192 K
  STORAGE INCREMENT SIZE: 256 K
  ENTRIES:  IN-USE:       5035 TOTAL:      18490,  27% FULL
  ELEMENTS: IN-USE:      20067 TOTAL:      72943,  27% FULL
  EMCS:     IN-USE:          0 TOTAL:       9850,   0% FULL
  LOCKS:     TOTAL:       1024
  PHYSICAL VERSION: BDC23BE6 CC108060
  LOGICAL  VERSION: BDBEC3D8 324E6546
  SYSTEM-MANAGED PROCESS LEVEL: 9
  DISPOSITION    : KEEP
  ACCESS TIME    : NOLIMIT
  MAX CONNECTIONS: 32
  # CONNECTIONS  : 0
```

The two interesting values in this example are highlighted in red. The actual size of the structure is 40192K but the policy size is 500000K. This means that the size of the structure can be altered up to 500000K without changing the CFRM policy. Any change over this size would require a CFRM policy change. The next check is to ensure that there is enough space available in the CF, using the D CF command will provide this information (D CF,CFNAME=P5CF01 in this case as the CF is P5CF01).

```
 IXL150I  11.22.32  DISPLAY CF 943
COUPLING FACILITY 002064.IBM.83.00000001169E
PARTITION: 0E  CPCID: 00

                  CONTROL UNIT ID: FFF1

 NAMED P5CF01
COUPLING FACILITY SPACE UTILIZATION
ALLOCATED SPACE                 DUMP SPACE UTILIZATION
STRUCTURES:      431360 K       STRUCTURE DUMP TABLES:          0 K
   DUMP SPACE:      8192 K                 TABLE COUNT:          0
  FREE SPACE:     547328 K       FREE DUMP SPACE:             8192 K
 TOTAL SPACE:     986880 K      TOTAL DUMP SPACE:             8192 K
                                MAX REQUESTED DUMP SPACE:     8704 K
```

```
    VOLATILE:           YES        STORAGE INCREMENT SIZE:        256 K
     CFLEVEL:            12
     CFCC RELEASE 12.DB, SERVICE LEVEL 10.A8
     BUILT ON 08/14/2003 AT 13:35:00
     COUPLING FACILITY HAS ONLY SHARED PROCESSORS

     CF REQUEST TIME ORDERING: NOT-REQUIRED AND ENABLED

 COUPLING FACILITY SPACE CONFIGURATION
                           IN USE            FREE           TOTAL
 CONTROL SPACE:          439552 K        547328 K        986880 K
 NON-CONTROL SPACE:           0 K             0 K             0 K

 SENDER PATH       PHYSICAL            LOGICAL       CHANNEL TYPE
   E0               ONLINE              ONLINE        ICP
   E6               ONLINE              ONLINE        ICP

 COUPLING FACILITY DEVICE     SUBCHANNEL     STATUS
                    FEB3         1990        OPERATIONAL/IN USE
                    FEB4         1991        OPERATIONAL/IN USE
```

Check the FREE SPACE quantity to know how much space is available in the CF. Having done this and determined how much to expand the structure by the SETXCF command is used to change the size of the structure. For example:

```
SETXCF START,ALTER,STRNAME=SQ27APPLICATION3,SIZE=100000
```

This would alter the size of the structure to be 100000K. The expansion for the structure can be performed while work is continuing to access the structure (it is not necessary to shut down the queue managers).

In the case of the POLICY SIZE not being large enough, it will be necessary to update the CFRM policy to be able to use a larger structure. Set the SIZE of the structure in the CFRM policy to be the maximum size you might want it to be and the INITSIZE to at least the current size of the structure. Having updated the policy it is necessary to activate it, this is done with the SETXCF START,POLICY command (eg. SETXCF START,POLICY,TYPE=CFRM,POLNAME=POL1).

Further information on the Display and Set XCF commands can be found in “z/OS V1R6.0 MVS System Commands” {5}.

3.4 How do shared channels and clustering differ?

One of the frequent questions that arises is whether shared channels should be used in a shared queues environment, rather than clustered channels. The answer will depend on exactly what is trying to be achieved. In both cases, high availability will be achieved as there are multiple remote instances that can be connected to, so there won't be a single point of failure at the remote end of the channel. Using clustered channels, each message will be targeted to a specific destination queue manager, but if that queue manager can't be reached then the message could be redirected to a different queue manager. In the case of a shared channel the message is destined for the queue sharing group, the channel will be started into an available queue manager, and then all the messages delivered down this channel. Fundamentally this all boils down to the fact that if clustered

channels are being used then there will be multiple concurrent routes available, whereas with shared channels there is will only be a single channel available. Therefore, if ordering is not an issue, using clustered channels should provide greater throughput when compared to shared channels.

3.5 What console messages should I be interested in?

Each MQ console message ends in a message type code, these identify if there is any action that is required as a result of the message. The message type codes are A, D, E and I. The meanings of these codes and actions for specific messages are detailed in the WebSphere MQ for z/OS Messages and Codes manual {6}. However, there are a number of information messages that are issued by the queue manager or channel initiator that should be flagged as having special interest. These messages do not in themselves indicate a problem, but may be useful in tracking because they do indicate a potential issue which may need addressing. This allows us to prevent issues before they arise.

For example, CSQI030I tells us that a page set has allocated a new pageset extent.

This may be ok, you may have small extents defined, but it may also indicate:

- A user application is putting lots of messages to the pageset – is it looping?
- A user application is putting lots of messages to the pageset – has the 'getter' failed and so is not processing messages as they arrive?

You should examine the list below and determine whether you feel that your automation should track them. If you decide message tracking is desirable, appropriate documentation for your installation should be provided so that the automation, problem revolvers can follow up the issue correctly.

CSQI031I *csect-name* THE NEW EXTENT OF PAGE SET *psid* HAS FORMATTED SUCCESSFULLY

Check the curdepth of the queues allocated to this pageset. Why are the messages not being processed ?

CSQI041I *csect-name* JOB *jobname* USER *userid* HAD ERROR ACCESSING PAGE SET *psid*

Is the pageset allocated to the queue manager?

Issue a DISPLAY USAGE command to determine the state of the pageset.

Check the queue manager joblog for additional error messages.

CSQJ004I ACTIVE LOG COPY *n* INACTIVE, LOG IN SINGLE MODE, ENDRBA=*ttt*

The queue manager has activated 'single' logging mode. This is often indicative of a log offload problem.

Issue a DISPLAY LOG command to determine your settings for duplexing of active and archive logs. This display also shows how many active logs need offload processing.

Check the queue manager joblog for additional error messages

CSQJ114I ERROR ON ARCHIVE DATA SET, OFFLOAD CONTINUING WITH ONLY ONE ARCHIVE DATA SET BEING GENERATED

Check the queue manager joblog for additional error messages.

Make a second copy of the archive log and update your BSDS manually.

CSQJ136I UNABLE TO ALLOCATE TAPE UNIT FOR CONNECTION-ID=*xxxx*CORRELATION-ID=*yyyyyy*, *m* ALLOCATED *n* ALLOWED

Allocate enough units to satisfy archiving.

Check the queue manager joblog for additional error messages.

CSQJ151I *csect-name* **ERROR READING RBA** *rrr*, **CONNECTION-ID=***xxxx* **CORRELATION-ID=***yyyyyy* **REASON CODE=***ccc*

Check the queue manager joblog for additional error messages

CSQJ160I LONG-RUNNING UOW FOUND, URID=*urid* **CONNECTION NAME=***name*

Check the queue manager joblog for additional error messages

Issue a DISPLAY CONN command to determine which connection is not committing its activity.

Ensure the application can commit its updates.

CSQJ161I UOW UNRESOLVED AFTER *n* **OFFLOADS, URID=***urid* **CONNECTION NAME=***name*

Check the queue manager joblog for additional error messages

Issue a DISPLAY CONN command to determine which connection is not committing its activity.

Ensure the application can commit its updates.

CSQP011I CONNECT ERROR STATUS *ret-code* **FOR PAGE SET** *psid*

Is the pageset allocated to the queue manager?

Issue a DISPLAY USAGE command to determine the state of the pageset.

Check the queue manager joblog for additional error messages.

CSQP013I *csect-name* **NEW EXTENT CREATED FOR PAGE SET** *psid*. **NEW EXTENT WILL NOW BE FORMATTED**

Check the curdepth of the queues allocated to this pageset.

Why are the messages not being processed?

CSQP014I *csect-name* **EXPANSION FAILED FOR PAGE SET** *psid***. FUTURE REQUESTS TO EXTEND IT WILL BE REJECTED**

Check the curdepth of the queues allocated to this pageset.

Why are the messages not being processed ?

You may wish to move some queues from this pageset to another one.

If the volume is full, you may wish to make the pageset a multi volume dataset.

If the the pageset is already multi-volume, consider adding more volumes to the storage group being used.

Once more space is available, it is possible to retry the expansion (in MQ V6) by setting the pageset EXPAND method to SYSTEM. (Toggling EXPAND to SYSTEM then back to your normal setting, will also allow a retry when appropriate).

CSQP016I *csect-name* **PAGE SET** *psid* **HAS REACHED THE MAXIMUM NUMBER OF EXTENTS. IT CANNOT BE EXTENDED AGAIN**

Check the curdepth of the queues allocated to this pageset.

Why are the messages not being processed ?

You may wish to move some queues from this pageset to another one.

CSQP017I *csect-name* **EXPANSION STARTED FOR PAGE SET** *psid*

Check the curdepth of the queues allocated to this pageset.

Why are the messages not being processed ?

CSQQ008I *nn* **units of recovery are still in doubt in queue manager** *qqqq*

Check the IMS log to determine why the UOWs are still indoubt.

Issue DISPLAY THREAD commands to determine the state of the UOWs in MQ

CSQQ113I *psb-name region-id* **This message cannot be processed**

Check the state of your dead letter queue (DLQ).

Ensure DLQ it is not PUT disabled.

Ensure the DLQ is not at the MAXMSG limit.

CSQX032I *csect-name* **Initialization command handler terminated**

Check the CSQOUTX dataset for why the commands in your CSQINPX failed.

There may be some commands not processed.

CSQX035I *csect-name* **Connection to queue manager** *qmgr-name* **stopping or broken, MQCC=***mqcc* **MQRC=***mqrc*

Check the MQRC as to why the call failed. These codes are documented in the Messages and Codes manual.

CSQX048I *csect-name* **Unable to convert message for** *name*, **MQCC=***mqcc* **MQRC=***mqrc*

Check the MQRC as to why the call failed. These codes are documented in the Messages and Codes manual.

CSQX107I *csect-name* **TCP/IP using TCPTYPE=***tcptype* **is not available**

Check the joblog for why TCPIP is not available.

Check the TCPIP address space for errors.

CSQX234I ***csect-name*** **Listener stopped, TRPTYPE=*trptype* INDISP=*disposition***

If the listener is not stopping because of a STOP command, check your TCPIP address space for errors.

Follow the Systems Programmer Response.

CSQX407I *csect-name* **Cluster queue** *q-name* **definitions inconsistent**

Some of the cluster queues within the cluster have different values. Check why they are different.

CSQX411I *csect-name* **Repository manager stopped**

If the repository manager has stopped because of an error, check the joblog for messages.

CSQX417I *csect-name* **Cluster-senders remain for removed queue manager** *qmgr-name*

Follow the Systems Programmer response.

CSQX418I *csect-name* **Only one repository for cluster** *cluster-name*

IBM recommends that all clusters have two full repositories.

CSQX419I *csect-name* **No cluster-receivers for cluster** *cluster-name*

Follow the Systems Programmer Response

CSQX420I *csect-name* **No repositories for cluster** *cluster-name*

Follow the Systems Programmer Response

3.6 Can I clear out the messages from my queue manager/chin JESMSGLG?

MQ records system status and abnormal situations by writing messages to the MVS console. These messages are also output on the JES message log of the QMGR and CHIN address spaces. A common issue for customers is that after a long period of running, the spool space used by the message log may become exorbitant. The problem may be particularly severe for a CHINIT where very many channels are constantly starting and stopping. A solution described below is to exploit the ability of JES (since z/OS 1.2) to periodically spin off datasets.

SVRCONN Channels

If your particular concern is the CSQX500I and CSQX501I messages produced in the CHINIT joblog in response to the starting and stopping of CLNTCONN / SVRCONN then also review the following APAR. APAR PK02576 for V530, PQ93792 for V531 and rolled into base V6, provides code in the CHINIT which recognises a new CHINIT service parameter (CHISERVP) and, when it is set, suppresses these messages entirely saving both CPU and log space. Once this fix has been installed, it is activated with a CHISERVP setting of x'20'.

Outline of method

Instead of running the QMGR and CHIN as started tasks, they are run as started jobs. This technique allows a SPIN specification to be setup on the jobcard. In our example, we setup the QMGR so that JES message log datasets are spun off at a fixed time every day, and the CHINIT, so that message logs are spun off, once a size threshold is reached.

Started Jobs

The "*z/OS JCL Reference*" {7} describes considerations for setting up started jobs or converting existing started tasks to be started jobs. In our environment we performed the following steps:

- Ensure the dataset containing the QMGR and CHIN JCL is included in the master JCL IEFPDSI concatenation. Edit the MSTJCLxx member of SYS1.PARMLIB if it's not there. This change requires an IPL to take effect.
- Edit the QMGR and CHIN JCL as follows:
- Replace the existing `//xxxxMSTR PROC` statement with a `// SET` for any substitution defaults
- Add a `//xxxxMSTR JOB` statement – this must be the first statement in the JCL

- Add an appropriate `JESLOG=(SPIN, )` parameter to the JOB statement

Examples

```
//MQ38MSTR JOB JESLOG=(SPIN,'18:30')
//*************************************************************
//* This is the procedure for starting the named MQ Queue Manager
//*************************************************************
//*MQ38MSTR PROC QMGR=MQ38,VER=V000,LVL1=CUR
// SET QMGR=MQ38,VER=V000,LVL1=CUR
//*
//PROCSTEP EXEC PGM=CSQYASCP,REGION=7168K
//STEPLIB   DD DSN=VB.&QMGR..APF.LOAD,DISP=SHR
//          DD DSN=MQB1.&VER..&LVL1..OUT.SCSQANLE,DISP=SHR
//          DD DSN=MQB1.&VER..&LVL1..OUT.SCSQAUTH,DISP=SHR
//          DD DSN=SYS2.DB2.V610.SDSNLOAD,DISP=SHR
```

The QMGR will SPIN off the JES message log at 18:30 every evening. The default JCL variables are now specified by a SET. Notice that the JOB statement is now the first line of JCL, whereas previously, the PROC statement could occur after a preamble of comments.

```
//MQ38CHIN JOB JESLOG=(SPIN,5000)
//*************************************************************
//* This is the procedure for starting the channel initiator
//*************************************************************
//*        PROC QMGR=MQ38,LVL=CUR
// SET           QMGR=MQ38,LVL=CUR
//PROCSTEP EXEC PGM=CSQXJST,REGION=7168K
//STEPLIB   DD DSN=VB.&QMGR..APF.LOAD,DISP=SHR
//          DD DSN=MQB1.V000.&LVL..OUT.SCSQANLE,DISP=SHR
//          DD DSN=MQB1.V000.&LVL..OUT.SCSQAUTH,DISP=SHR
//          DD DSN=MQB1.V000.&LVL..OUT.SCSQMVR1,DISP=SHR
```

The CHIN will SPIN off the message log every time 5000 lines of output have been written.

MQ Considerations

Both the START QMGR and START CHINIT commands can be passed a PARM, the name of the CSQZPARM or CSQXPARM module to provide initialization information. (Note that in WebSphere MQ for z/OS V6, CSQXPARM has been replaced by queue manager attributes). Additionally, an ENVPARM to pass JCL overrides may also be used. These have been tested with the scheme above work. eg

```
!MQ38 START QMGR PARM(JOBZPARM) ENVPARM('VER=V600')
```

Managing Spun Off Datasets

Once the output datasets have been spun off, they are available for printing or archiving to dataset and can then be purged from the spool freeing up spool space. (SDSF line operators XDC for printing, and P for PURGE).

Additionally, now the job is eligible for spinning off datasets, this can be performed manually via JES command (SDSF W line operator).

4 Resources

4.1 Bibliography

1 SA22-7625-10: "z/OS V1R6.0 MVS Setting Up a Sysplex" URL:
http://www.ibm.com/servers/eserver/zseries/zos/bkserv/zshelves6.html

2 SC34-6597-00: "WebSphere MQ Script (MQSC) Command Reference" URL:
http://www.ibm.com/software/integration/wmq/library/

3 WebSphere MQ Development: "SupportPac MP16" URL:
http://www.ibm.com/support/docview.wss?rs=171&uid=swg24007421&loc=en_US&cs=utf-8&lang=en

4 Lyn Elkins: "Best Practices: WebSphere MQ shared queues and application programs" URL:
http://www.ibm.com/developerworks/websphere/library/techarticles/0512_elkins/0512_elkins.html

5 SA22-7627-11: "z/OS V1R6.0 MVS System Commands" URL:
http://www.ibm.com/servers/eserver/zseries/zos/bkserv/zshelves6.html

6 GC34-6602-00: "WebSphere MQ for z/OS V6.0 Messages and Codes" URL:
http://www.ibm.com/software/integration/wmq/library/

7 SA22-7597-08: "z/OS V1R6.0 MVS JCL Reference" URL:
http://www.ibm.com/servers/eserver/zseries/zos/bkserv/zshelves6.html

