WebsphereMQ Coding Considerations

Overview

This document discusses some of the important application programming considerations that need to be thought about when designing and coding Websphere MQ enabled applications.

Whereas these considerations need to be 'thought about' they should also be discussed with the business analysts and business unit to determine what the actual business needs are.

This page will not provide a 'magic bullet', nor will it be a 'how to' document, what it should do is give an application programmer an idea of the questions that need to be answered in order to code the application to business specifications.

There is usually an urge for and clients to overestimate the importance of their application to the business and therefore become too aggressive in the requirements of their application's error handling process. In addition, one must take into account the effects of their choices on other business areas. For example, if they decide that their program has to keep running regardless of the impact on other applications you may need to discuss with other business areas the impacts of fulfilling any 'too aggressive' requirements.

Many of the coding considerations will depend upon lots of variables that covering how to handle all of them on this page is an unrealistic goal. The purpose of this is page is simply to raise awareness of some of the questions that should be asked and answered when designing a MQ enabled application.
Coding Considerations
As previously mentioned, many of the solutions to the coding considerations with an MQ enabled application revolve around the actual business requirements but some of the considerations should be universal (like what do you do when you get a particular MQ reason code returned?)

Often, when asked the question "what does MQ do when..?" the answer is "it depends"! This is often a source of frustration as the business user doesn't want (or need) to know exactly how MQ does things, but they do need to know what it can do in certain situations.

So, what should well-written, well-behaving MQ programs do?

The answer may seem obvious, but it is often ignored. They should handle non-zero MQ reason codes, especially if they are to prevent corrupting data, losing data or causing unwanted bottlenecks.

Here is brief list of “do's” and “don’ts” for MQ enabled programs/applications.

Do's

- Check for a 2161 (MQRC-Q-MGR-QUIESCING) reason code on all applicable MQ calls
- Always check for a truncated message on an MQGET.
- Check for a "poison" message on an MQGET.
- Use CORRELID & MSGID whenever possible.
- Use the Persistence option "MQPER_PERSISTENCE_AS_Q_DEF" whenever possible.
- Use triggered channels where practical.
- Always use CONVERT on all MQGET calls.
- Use the Get Message Option "WAIT" wisely. (See next bullet point!)
- Have a shutdown message capability for a UNLIMITED wait on a MQGET.
- Use message expiry only when the receiving application is correctly coded to handle it.

- Monitor the "Dead Letter Queue" and decide what to do with any messages that are there. (This is more a system administrator role)

Dont's

- Assume that all messages on a queue are for your application! (Check for poison messages)
- Use Queue depth as a loop counter. Just don't, it won't turn out well!
- Hold inactive channels in a "running" state.
- Assume that message persistence is 'a given' for production, it isn't!
- Ignore truncated messages. This can lead to data loss.
- Use too short a wait time on MQGET, especially if the request is waiting for a reply from a different platform.
- Use too long a wait (or unlimited wait) without an escape plan!
- Assume messages will arrive in order!
- Ignore case! Case kills! All MQ objects are case sensitive. ie MyQueue ≠ MyQUEUE

Some Reason Code Handling Suggestions

You should always investigate and identify the cause of any unexpected MQ reason code as this could be an indication of an underlying system issue.

In most cases a non-zero reason code should always be displayed somewhere (console, JOBLOG etc.) even if it isn't going to be handled. Surprises happen and there is nothing worse than a program that fails but never tells you why without you having to dig around.

We won't discuss return codes that are really part of a set-up process and should be straightforward like a 2035 (MQRC_NOT_AUTHORIZED), 2011 (MQRC_DYNAMIC_Q_NAME_ERROR), 2019 (MQRC_HOBJ_ERROR), 2058 (MQRC_Q_MGR_NAME_ERROR) and other such errors which you should never get once the application has been developed. These are easy to diagnose and resolve.

The following list is not meant to be a comprehensive list of potential return codes rather it is a list of reasonably common return codes and some suggested actions. (Obviously your preferred action will depend on the application/business requirements.)

2009 (MQRC_CONNECTION_BROKEN)

This reason code is usually returned when a stop command has been issued for the queue manager.

Please refer to the suggestions for a 2161 (MQRC_Q_MGR_QUIESCING) for how to deal with this reason code.

2016 (MQRC_GET_INHIBITED)

This is seldom a good thing; however, it may not be entirely a reason to panic. Queues can be set to GET inhibited for backup purposes and so it may just be a case of waiting a while and then retrying the MQGET. Of course, it could also be that the queue has been 'removed from service' for some other reason so you should always investigate the cause before deciding what to do with the failing application.

2030 (MQRC_MSG_TOO_BIG_FOR_Q)

We have included this reason code as it could be an indication of an application error. For example, an application has been in production for a while and suddenly somebody has tried to put the entire works of William Shakespeare into one long message!

Checks should be made to ensure that no abnormally large messages are allowed to be put to a queue. If they are needed, then maybe a redesign of the application could be advantageous.

Other potential solutions would be to split the message into smaller chunks and process them...Danger, danger never assume messages will arrive on a queue in the same order that they were sent!

2033 (NO_MESSAGE_AVAILABLE)

This may or may not be an error!

This return code is issued if there are no more messages to be retrieved from the queue that match your message options. This may indicate the queue is genuinely empty or that you have got your message options incorrect. It could also be an issue if you are waiting for a REPLY message that hasn't returned to the queue yet.

There are several things to consider here in order to resolve the 'issue':

- Were you expecting a message? (is the queue empty?)
- Are there any 'uncommitted' messages on the queue?
- Has the reply message actually been issued in a timely manner? (is something stuck somewhere!)
- Are your CORRELID and MSGID options correct?
- Was the REQUEST message actually committed to the queue successfully?
-

Sometimes issues within the network can result in a 2033 being issued for a reply from a remote application. For example, you specify an unrealistic GET wait time for a response from a remote server application, or you don't take into consideration the elapsed time taken by an application to process your request.

You should investigate why you got no message returned and take action accordingly. The action may just be to try the GET, or it could be to close the program.

2034 (MQRC_NO_MSG_UNDER_CURSOR)

This is usually symptomatic of an application error. An attempt has been made to get a message using either the MQGMO_MSG_UNDER_CURSOR or the MQGMO_BROWSE_MSG_UNDER_CURSOR option but there isn't an applicable message under the cursor.

The usual reason for this is that you are either, attempting to retrieve a message that has expired or someone else read (and removed) the message before your program could (hence you code a receiving application to handle this).

2042 (MQRC_OBJECT_IN_USE)

This error is fairly self-explanatory but is an indication of there being a badly behaving application. Essential one process has the queue open as exclusive. It is much the same as with a dataset enqueue.

The best fix is to identify the application opening the object as exclusive and either change it or have your application use a different queue. If you know that the application holding the queue as exclusive is just a momentary application or is due to end you could decide to wait until it terminates and retry your call. (See our **MQERROR** page.)

2051 (MQRC_PUT_INHIBITED)

This is much the same as a 2016 (MQRC_GET_INHIBITED). Queues can be set to PUT inhibited for backup purposes and so it may just be a case of waiting a while and then retrying the MQPUT (or MQPUT1) Of course it could also be that the queue has been 'removed from service' for some other reason.

Additionally, this could be an indication that a message was unable to be placed onto a remote queue manager so you should always investigate the cause before deciding what to do with the failing application.

2053 (MQRC_Q_FULL)

Obviously, this reason code is self-explanatory...the queue is full. The trick is to find out why! It could be that you simply need to increase the MaxQDepth for the queue in question, however, it could also be an indication that a 'consuming' application has stopped. It is therefore a good opportunity to inform the operators to check for a failed or stopped consuming application. (See our **MQERROR** web page.)

2059 (MQRC_Q_MGR_NOT_AVAILABLE)

This reason code, as the text implies, happens when an attempt is made to connect to a queue manager, and it isn't active. It may be a good idea to notify someone when this happens. The program could wait if the queue manager is temporarily down. (See our **MQERROR** web page.)

2063 (MQRC_SECURITY_ERROR)

If you get this reason code then something is really messed up with the security system (RACF, ACF2 etc) and, if operations, don't already know it, you need to let them know. This is not truly an application error, but it may be a condition that is truly recoverable by your program, however, it could be that system functions need to be restarted so it may be better to gracefully end your program.

2079 (MQRC_TRUNCATED_MSG_ACCEPTED)

This reason code may (or may not) be considered an error. It depends on what you are doing with the message etc., however, you need to be aware that not all of the message has been returned. In other words you could be losing data.

This reason code is acceptable if you are clearing out a queue and don't care about the message payload or if the only portion of the payload you are interested in is contained in the portion of the message that was returned.

To fix this issue, if needed, you need to increase the size of the message buffer being used and then rerun the job.

Note: The message that is causing this reason code to be thrown may well have already been removed from the queue depending on if browse or unit-of-work are being used or not.

2080 (MQRC_TRUNCATED_MSG_FAILED)

This reason code is an indication that the buffer that a message is being read into isn't large enough to hold the entire message and the Get Message Option "Accept Truncated Message" has not been specified.
The error can be handled in one of the following ways.

- Increase the buffer size and rerun the job.

- Have the program increase dynamically the buffer size according to the length of the message.
- Accept the short message.
- Delete (or move) the long message to another queue for special processing. (This is not an ideal solution)

No matter the solution the 'too long' message will still be on the queue after this message is issued.

2071 (MQRC_STORAGE_NOT_AVAILABLE)

If you get this reason code check to make sure your application isn't looping (or if under CICS that no other application is looping). if everything looks OK then it may be that you need a larger region size. Contact your System Programmers to seek advice in resolving this error.

2085 (MQRC_UNKNOWN_OBJECT_NAME)

For this reason code you have probably misspelled an object name. It is included here simply because it could be that you have the case incorrect for the object name.

Remember that case counts MyQueue ≠ MyQUEUE.

This reason code is also possible when moving an application to a different queue manager (rolling from a development environment into production) and a target queue hasn't been defined yet.

2086 (MQRC_UNKNOWN_OBJECT_Q_MGR)

This is basically the same as the above 2085 reason code but for a Queue Manager. It is an indication that the target queue manager doesn't exist on the system you are running your application on. It could be that you have run the job on the wrong system (LPAR), but it could also be that you have misspelled the queue manager name (remember case counts!). The worst-case scenario is that the queue manager hasn't been built for that systems yet. This would require you to contact your system administrators and have it defined (assuming it is approved for that system).

2087 (MQRC_UNKNOWN_REMOTE_Q_MGR)

This reason code is returned for if the name of the remote queue manager cannot be resolved by the connected queue manager.

This is usually a setup error, but it could be that you have incorrectly coded (or corrupted) a remote queue object name. This error can also occur when the ObjectQMgrName is corrupt or incorrect.

2100 (MQRC_OBJECT_ALREADY_EXISTS)

If you receive this reason code it is an indication that a dynamic queue name already exists. This is really an indication that you are not using dynamic queue names correctly. You should only supply a root for the dynamic queue name and let the queue manager complete the name. If you truly need to supply the full name, then ensure that no other program is using the same name (this is really to a good practice).

2101 (MQRC_OBJECT_DAMAGED)

If you get this reason code, then you should close down your application and notify your support personnel as this reason code implies that there is a real issue with the queue manager, and it may need restoring and restarting.

2102 (MQRC_RESOURCE_PROBLEM)

This is another reason code that really requires you to shutdown your application as the operating system is under stress. It could be a temporary situation, but you need to quiesce gracefully and not contribute to the mayhem.

If you persistently get this reason code it could be your application needs tuning but more likely is that your system is in need of an upgrade.

2105 (MQRC_STORAGE_CLASS_ERROR)

This is another of those reason codes usually returned when migrating your application to a different queue manager. It is returned because the storage class hasn't been set up on the new target queue manager.

You should either get the storage class defined or (less desirably) change the storage class used by your application.

2161 (MQRC_Q_MGR_QUIESCING)

This reason code is issued if a shutdown has been issued for the target queue manager. The action you are trying to execute will be unable to proceed.

The actions you take are dependent on what you are trying to achieve, however, taking no action is really not an option. It must be assumed that the queue manager is being shutdown for a good reason and if you don't take any action you will be preventing it completing the shutdown in a timely manner. (See our **QMGRWAIT** web page.)

If you get this reason code on a connect then you could simply wait for the queue manager to be restarted and retry the connect, or you could close down your application. Our **QMGRWAIT** program could be utilized in this situation. For details, please refer to the **QMGRWAIT** web page.

On other MQ calls it becomes somewhat of a deadly embrace situation. The queue manager can't shutdown because applications are connected but the applications can't complete because the queue manager is shutting down!

What all well behaving applications should do is:

- Assess the need to roll back already processed data.
- Disconnect from the Queue manager.
- Fail the code or call the **QMGRWAIT** program.
- If the code is failed, then normal abnormal end processing can take place.
- If **QMGRWAIT** is called, then when the queue manager returns to active state the program can reconnect to the queue manager and then retry the open.

Obviously, these decisions are business driven and those decisions should be taken with the business unit.

It should be noted that using an MQPUT1 would still have the same issues as an MQPUT1 still attempts to open a queue.

2162 (MQRC_Q_MGR_STOPPING)

This reason code basically has the same application implications as the 2161 (MQRC_Q_MGR_QUIESCING) and should have the same application considerations.

2192 (MQRC_PAGESET_FULL)

This reason code is an indication that the system is in dire need of some help! It is usually a pageset full which requires some action to be taken by a System Programmer.

It could be that there is a queue that is filling up without anything consuming the messages. This queue may or may not be your queue. It may, however, be that the queue manager will need to be shut down so your application should try to back out any incomplete unit of work and then close down tour application.

2195 (MQRC_UNEXPECTED_ERROR)

This reason code may or may not be due to an application error. It could also be a genuine system error that requires a System Programmer to research it.

If you have checked your parameter list and find no error, then it is likely a system issue.

The recommended action is to back out any unfinished unit of work, close the application and notify your site support System Programmer.

2196 (MQRC_UNKNOWN_XMIT_Q)

This is another of those setup/migration reason codes. The usual cause is that there is no transmission queue set up to the remote queue manager you are attempting to reach.

The solution is to have one defined and retry your application.

2197 (MQRC_UNKNOWN_DEF_XMITQ)

This is pretty much the same as the 2196 (MQRC_UNKNOWN_XMIT_Q) and should be resolved in the same manner.

Strictly speaking it is somewhat different, but the solution remains the same.

2202 (MQRC_CONNECTION_QUIESCING)

This can be considered the same as a 2161 (MQRC_Q_MGR_QUIESCING) reason code and has the same application implications.

2203 (MQRC_CONNECTION_STOPPING)

This can be considered the same as a 2161 (MQRC_Q_MGR_QUIESCING) reason code and has the same application implications.

2218 (MQRC_MSG_TOO_BIG_FOR_CHANNEL)

We have included this reason code as it could be an indication of an application error. For example, an application has been in production for a while and suddenly somebody has tried to put a huge message to a remote queue manager!

Checks should be made to ensure that no abnormally large messages are allowed to be put to a remote queue. If they are needed to be, then maybe a redesign of the application could be advantageous?

Other potential solutions would be to split the message into smaller chunks and process them...Danger, danger never assume messages will arrive on a queue in the same order that they were sent!

2223 (MQRC_Q_MGR_NOT_ACTIVE)

This can be considered the same as a 2059 (MQRC_Q_MGR_NOT_AVAILABLE) reason code and has the same application implications.

Additional Notes

It should be remembered that this is not a full nor comprehensive list of potential MQ reason codes. It has been provided as an aide-mémoire for potential application considerations.

Well behaving applications can save both resources and time.

Our recommendation is to handle whatever error recovery is appropriate and if you get a unhandled error to terminate the program using the MQ reason code as the condition code for the program. i.e., If you get a 2035 and choose to terminate the program then the step should terminate with a 2035 return code as this will aid in debugging the application.

Disclaimer

It is the responsibility of any user of this material to evaluate its usefulness to the user's environment.

Abbydale Systems LLC. does not guarantee to keep this or any related material current, nor does it guarantee to provide any corrections or extensions described by any users of this material or any corrections or extensions made in the future by Abbydale Systems LLC. itself.