25

MQ

July 2001

In this issue

3 MQSeries clustering hints and tips

8 OS/390 log datasets expansion

13 Deleting expired messages

16 Customizing files

24 MQBKQLST – CSQUTIL COPY a list of queues on OS/390

32 MQSeries message rate driver

40 Finding the maximum message length of a queue

44 MQ news

update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 38126
From USA: 01144 1635 38126
Fax: 01635 38345
E-mail: info@xephon.com

North American office

Xephon/QNA
Post Office Box 350100
Westminster CO 80035-0100, USA
Telephone: (303) 410 9344
Fax: (303) 438 0290

Contributions

When Xephon is given copyright, articles published in *MQ Update* are paid for at the rate of £170 ($260) per 1000 words and £100 ($160) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £50 ($80) per 100 lines. In addition, there is a flat fee of £30 ($50) per article. For more information about contributing an article you can download a copy of our *Notes for Contributors* from www.xephon.com/contnote.html.

MQ Update on-line

Code from *MQ Update*, and complete issues in Acrobat PDF format, can be downloaded from our Web site at www.xephon.com/mqupdate.html (you will need to supply a word from the printed issue).

Commissioning Editor

Peter Toogood
E-mail: PeterT@xephon.net

Managing Editor

Madeleine Hudson
E-mail: MadeleineH@xephon.com

Disclaimer

Readers are cautioned that, although the information in this journal is presented in good faith, neither Xephon nor the organizations or individuals that supplied information in this journal give any warranty or make any representations as to the accuracy of the material it contains. Neither Xephon nor the contributing organizations or individuals accept any liability of any kind howsoever arising out of the use of such material. Readers should satisfy themselves as to the correctness and relevance to their circumstances of all advice, information, code, JCL, scripts, and other contents of this journal before making any use of it.

Subscriptions and back-issues

A year's subscription to *MQ Update*, comprising twelve monthly issues, costs £255.00 in the UK; $380.00 in the USA and Canada; £261.00 in Europe; £267.00 in Australasia and Japan; and £265.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the July 1999 issue, are available separately to subscribers for £22.50 ($33.50) each including postage.

Printed in England.

MQSeries clustering hints and tips

MODIFYING CLUSTER OBJECTS

When modifying the properties of any cluster object, make sure that you refresh the cluster repository so that all other queue managers inside that particular cluster have an updated view of the cluster object.

A good example of the potential problem that can occur here is changing a cluster local queue from PUT_ENABLED to PUT_INHIBITED. All applications should be returned an MQRC_CLUSTER_PUT_INHIBITED reason code, but if a remote queue manager has not had its view of the cluster queue updated it will still be able to put messages to the cluster queue, regardless of the queue having been set to PUT_INHIBITED.

(IBM has released a patch for the example described above.)

LOAD-BALANCE CLUSTER QUEUES

Load-balance cluster queues are defined by having multiple instances of a queue name inside any one cluster. Messages to these load-balance cluster queues will be put in a 'round-robin' fashion by default if the queues are available for message puts.

Load-balance cluster queues will not work across clusters even if the queue manager that is issuing a message put is a member of all of those multiple clusters. For example, the queue manager that hosts the putting application has a namelist with multiple clusters defined. If the load-balance queue exists in more than one of the clusters defined in the namelist it will only be able to put to the load-balance cluster queues that are members of the first cluster defined in the namelist.

Even if all the load-balance cluster queues in the first cluster are PUT_INHIBITED and the load-balance cluster queue of the same name in the next cluster defined in the namelist is PUT_ENABLED, the putting application will be returned an MQRC_CLUSTER_PUT_INHIBITED reason code.

WINDOWS NT DHCP

"It is not advisable to use clusters in an environment where IP addresses change on an unpredictable basis, such as on machines where Dynamic Host Configuration (DHCP) is being used." *MQSeries for Windows NT: Quick Beginnings Version 5.1*.

CLIENTS PUTTING MESSAGES TO CLUSTER QUEUES

Version V5.0 and V5.1 NT clients can put messages to cluster queues, and the queue manager to which the client connects can invoke the manually-written load balance exit to distribute the workload.

WORKLOAD BALANCE EXIT

A sample workload balance exit is provided with MQSeries V5.1 **amqswlm0.c**. For this to be executed, the queue manager must have its *clwlexit* and *clwldata* attributes modified, as instructed in the sample exit comments.

- On MQ V5.1 for Solaris 2, a call to the system clock from the 'time()' function in <*time.h*> and <*sys/time.h*> ISO C header will cause the queue manager to hang when the workload balance exit is invoked.
- On MQ V5.1 for Solaris 2, the workload balance exit will be loaded into memory when starting the queue manager. The queue manager must be restarted whenever the workload balance exit is updated.
- An effective technique for use in the workload balance exit is a call to 'rand()' function for workload balancing. As the workload balance exit is stored in memory, the 'rand()' call will not produce the same random number for each call.

CLUSTER-SENDER CHANNELS

Cluster-sender channels, when automatically defined or started from a channel initiator, will retrieve the definition of the cluster-receiver channel and override all channel attributes of the sender. It is particularly important to ensure that connection names and other attributes of the

sender and receiver are identical. For example, a channel receiver with a connection name of *host1* and a sender with an IP address of *x.x.x.x* as the connection name will run when manually started, but when started by a channel initiator the sender connection name will be overridden by *host1*. It is then crucial that *host1* is a recognized hostname or the channel will not start.

HOSTNAMES

A common problem with cluster channels is unrecognized hostnames. The connection name on a cluster-receiver channel must be recognized literally in the Domain Name Server by the host of the cluster-sender channel. If a channel constantly fails to start, check your MQSeries error logs for the following message.

```
AMQ92Ø3: A configuration error for TCP/IP occurred.
EXPLANATION:
Error in configuration for communications to host NTCCHSIT0002.
Allocation of a TCP/IP conversation to host NTCCHSIT0002 was not
possible.
ACTION:
The configuration error may be one of the following:
1.If the communications protocol is LU 6.2, it may be that one of the
transmission parameters (Mode, or TP Name) is incorrect. Correct the
error and try again. The mode name should be the same as the mode
defined on host NTCCHSIT0002. The TP name on NTCCHSIT0002 should be
defined.
2.If the communications protocol is LU 6.2, it may be that an LU 6.2
session has not been established. Contact your systems administrator.
3.If the communications protocol is TCP/IP, it may be that the host name
specified is incorrect. Correct the error and try again.
4.If the communications protocol is TCP/IP, it may be that the host name
specified cannot be resolved to a network address. The host name may not
be in the nameserver.
The return code from the TCP/IP (gethostbyname) call was Ø (X'Ø').
Record the error values and tell the system administrator.
```

If you see this message consult your system administrator to make sure the hostname is recognized. A good way to test this is to perform an ‘ftp’ or ‘telnet’ function on the hostname. Be aware that a ‘ping’ works differently on ftp, telnet, and MQSeries.

SOURCE CODE OF AMQSWLM0.C

```
/* Program name: AMQSWLMØ (based on the IBM sample exit)            */
```

```
/* Sample C CLWL exit that chooses a destination queue manager       */
 /* Modifications : Markian Jaworsky (IBM GSA) [8, Sept, 2000]       */
 /*                 #1 - Only choose alternate queue manager, when   */
 /*                 cluster-sender channel is running.               */
 /*                 #2 - Add random number generator calls to break  */
 /*                 message flow evenly between local and remote     */
 /*                 queue manager.                                   */
 /* Function:                                                        */
 /*   AMQSWLM is a sample C exit which chooses a destination queue   */
 /*   manager. To use this exit:                                     */
 /*      -- Use RUNMQSC to set the queue manager attribute CLWLDATA  */
 /*         to the name of the channel to the remote queue manager.  */
 /*         (This information is passed to the CLWL exit in the      */
 /*         MQWXP structure in the ExitData field) eg:               */
 /*             ALTER QMGR CLWLDATA('TO.myqmgr')                     */
 /*      -- Use RUNMQSC to set the queue manager attribute CLWLEXIT  */
 /*         to the module name followed by the function name in      */
 /*         parenthesis. eg:                                         */
 /*             ALTER QMGR CLWLEXIT('amqswlm(clwlFunction)')         */
 /*      -- Make sure the CLWL module is accessible to MQ            */
 /*      -- End the queue manager then restart it, to pick up these  */
 /*         changes                                                  */
 #include <stdio.h>
 #include <stdlib.h>
 #include <string.h>
 #include <limits.h>
 #include <cmqc.h>
 #include <cmqxc.h>
 #include <cmqcfc.h>
 void MQENTRY clwlFunction ( MQWXP *parms )
 {
   MQLONG index;
   MQLONG GoodDestination;
   MQWDR  * qmgr;
   MQWQR  * queue;
   MQCD   * cd;
   MQBYTE16 counter;
   int value=0;
   int flag=0;
   /* Find the best destination in the destination array             */
   for ( index = 0; index < parms->DestinationCount; index++ )
   {
     GoodDestination = 1;
     qmgr = parms->DestinationArrayPtr[ index ];
     cd = (void*) ((char*) qmgr + qmgr->ChannelDefOffset);
     /* Does the channel to the destination qmgr match the one we    */
     /* are interested in ?                                          */
     if ( strncmp( cd->ChannelName
                 , parms->ExitData
                 , MQ_CHANNEL_NAME_LENGTH
                 )
```

```
   )
{
  GoodDestination = 0;
}
/* Is the queue PUT enabled?  Note that the QArrayPtr is        */
/* NULL in the case of choosing a destination queue manager     */
if (parms->QArrayPtr)
{
  queue = parms->QArrayPtr[ index ];
  if (queue && (queue->InhibitPut == MQQA_PUT_INHIBITED))
  {
    GoodDestination = 0;
  }
}
/* Don't choose destination qmgrs which have not joined the     */
/* cluster                                                      */
if ((qmgr->QMgrFlags & MQQMF_AVAILABLE) == 0)
{
  GoodDestination = 0;
}
/* Only choose destinations with channels in a good state       */
switch ( qmgr->ChannelState )
{
  case MQCHS_RUNNING:
    break;
  case MQCHS_INACTIVE:
  case MQCHS_BINDING:
  case MQCHS_STARTING:
  case MQCHS_STOPPING:
  case MQCHS_INITIALIZING:
  case MQCHS_RETRYING:
  case MQCHS_REQUESTING:
  case MQCHS_PAUSED:
  case MQCHS_STOPPED:
  default:
    GoodDestination = 0;
    break;
}
/* Did we send this message to the alternate queue manager in   */
/* the last call?  Interleave the message flow by random calls. */
  value = rand();
  flag = value % 2;
  if (flag) {
  GoodDestination = 0;
  }
/* If this destination is good, then choose it and stop looking */
/* for more                                                     */
if (GoodDestination)
{
  parms->DestinationChosen = index + 1;
  break;
```

```
    }
  }
  srand(value);
  return;
}
#if MQAT_DEFAULT == MQAT_UNIX
 void MQStart(){;}
#endif
```

Markian Jaworsky
Middleware Support, IBM-GSA (Australia)

OS/390 log datasets expansion

INTRODUCTION

OS/390 QMGR log datasets are normally created during the initial build of a QMGR using the *IDCAMS* utility. This defines the VSAM linear datasets to be used by MQSeries (see the example below). The MQSeries utility program *CSQJU003* is then employed to register the logs with the BSDS (BootStrap DataSets). Member *CSQ4BSDS* in the supplied *SCSQPROC* library provides the JCL (Job Control Language) to accomplish this task.

This article provides a procedure and a suite of jobs which you can use to enlarge the log datasets to prevent the situation whereby small log datasets fill so rapidly that the archival process cannot keep up, thus resulting in the QMGR hanging. This is not a situation recommended for a production system!

The situation will invariably arise whereby the initial sizing of logs will become insufficient due to growth in applications usage or number, and an increase in log sizes will become unavoidable.

EXAMPLE OF A VSAM LINEAR DATASET CLUSTER DEFINITION

```
DEFINE CLUSTER -
        (NAME (cluster.name)   -
        LINEAR                          -
        VOLUMES(volser)         -
```



```
    RECORDS(nnnn )          -
DATA                                -
   (NAME(cluster.name.DATA) )
```

We have chosen to use dual logging, as this method of writing log records writes to two log datasets instead of one in an attempt to prevent data loss, which could impact the restart of a QMGR. The use of dual logging is specified in the *CSQ6LOGP* macro, which, when assembled and linked, provides the *ZPRM* module that MQ uses at startup *(TWOACTV=YES*, as shown below).

EXAMPLE OF CSQ6LOGP PARAMETERS

```
CSQ6LOGP INBUFF=28,   LOG INPUT BUFFER SIZE KB
      MAXALLC=3,      MAX ALLOCATED ARCHLOG VOLS
      MAXARCH=500,    MAX ARCH LOG VOLUMES
      OFFLOAD=YES,    ARCHIVING ACTIVE
      OUTBUFF=400,    LOG OUTPUT BUFFER SIZE
      TWOACTV=YES,    DUAL ACTIVE LOGGING      <==============
      TWOARCH=YES,    DUAL ARCHIVE LOGGING
      TWOBSDS=YES,    DUAL BSDS
      WRTHRSH=20      ACTIVE LOG BUFFERS
Our OS390 QMGRS currently run on a set of 6 log datasets.
cluster.name.LOGCOPY1.DS01       }
cluster.name.LOGCOPY1.DS02       }    Copy 1
cluster.name.LOGCOPY1.DS03       }
cluster.name.LOGCOPY2.DS01       }
cluster.name.LOGCOPY2.DS02       }    Copy 2
cluster.name.LOGCOPY2.DS03       }
```

The default space allocation on these, as you can see from the above example, is 1,000 records, which, on a 3390, equates to 90 tracks or six cylinders. This may be adequate for a little-used QMGR, however; should you find the need to expand these, read on.....

LOG EXPANSION PROCESS

The following procedure comprises five steps, which will allow you to enlarge your log datasets. If you have multiple QMGRS running on OS/390 it is advisable to create a separate JCL library for each to preserve copies of each QMGRS' developed utilities – just in case someone alters a JCL deck, and you run it, 'gubbing' an otherwise happy QMGR! ('Gubbing' – to threaten one's livelihood by carrying out an act which threatens your production systems.)

1 Run job one as shown below.

```
//jobname JOB #,'WINDY M.',CLASS=x,MSGCLASS=X,
//         NOTIFY=&SYSUID
//*-------------------------------------
//* Job one in Log Reallocation suite.
//*
//* STEP1 - IDCAMS
//*   This step will initially perform a DELETE of the .NEW
//*   clusters it later creates to allow it to be rerun in the case
//*       of, say, a space failure.
//*       It also performs a DELETE of any .OLD clusters left
//*       around from a previous run.
//* STEP2 - IDCAMS
//*     Create larger log clusters specifying a VOL parameter off
//*     '*' allows SMS to pick correct volume. If you are not using
//*       SMS specify a volser here.
//*       REMEMBER to check space allocation !!!!!
//*       1000 Records = 6 Cylinders on a 3390
//*-------------------------------------
//STEP1    EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD *
   DELETE (cluster.name.LOGCOPY1.DS01.NEW) ERASE CLUSTER
   DELETE (cluster.name.LOGCOPY1.DS02.NEW) ERASE CLUSTER
   DELETE (cluster.name.LOGCOPY1.DS03.NEW) ERASE CLUSTER
   DELETE (cluster.name.LOGCOPY2.DS01.NEW) ERASE CLUSTER
   DELETE (cluster.name.LOGCOPY2.DS02.NEW) ERASE CLUSTER
   DELETE (cluster.name.LOGCOPY2.DS03.NEW) ERASE CLUSTER
   DELETE (cluster.name.LOGCOPY3.DS01.NEW) ERASE CLUSTER
   DELETE (cluster.name.LOGCOPY3.DS02.NEW) ERASE CLUSTER
   DELETE (cluster.name.LOGCOPY3.DS03.NEW) ERASE CLUSTER
   DELETE (cluster.name.LOGCOPY1.DS01.OLD) ERASE CLUSTER
   DELETE (cluster.name.LOGCOPY1.DS02.OLD) ERASE CLUSTER
   DELETE (cluster.name.LOGCOPY1.DS03.OLD) ERASE CLUSTER
   DELETE (cluster.name.LOGCOPY2.DS01.OLD) ERASE CLUSTER
   DELETE (cluster.name.LOGCOPY2.DS02.OLD) ERASE CLUSTER
   DELETE (cluster.name.LOGCOPY2.DS03.OLD) ERASE CLUSTER
   DELETE (cluster.name.LOGCOPY3.DS01.OLD) ERASE CLUSTER
   DELETE (cluster.name.LOGCOPY3.DS02.OLD) ERASE CLUSTER
   DELETE (cluster.name.LOGCOPY3.DS03.OLD) ERASE CLUSTER
   SET MAXCC = 0
 DEFINE CLUSTER -
        (NAME (cluster.name.LOGCOPY1.DS01.NEW) -
        LINEAR                          -
        VOLUMES(*)                      -
        RECORDS(10000) )                -
     DATA                               -
       (NAME(cluster.name.LOGCOPY1.DS01.NEW.DATA) )
 DEFINE CLUSTER -
```

```
        (NAME (cluster.name.LOGCOPY1.DSØ2.NEW) -
         LINEAR                          -
         VOLUMES(*)                      -
         RECORDS(1ØØØØ) )                -
     DATA                                -
        (NAME(cluster.name.LOGCOPY1.DSØ2.NEW.DATA) )
 DEFINE CLUSTER -
        (NAME (cluster.name.LOGCOPY1.DSØ3.NEW) -
LINEAR                          -
          VOLUMES(*)                      -
          RECORDS(1ØØØØ) )                -
      DATA                                -
         (NAME(cluster.name.LOGCOPY1.DSØ3.NEW.DATA) )
  DEFINE CLUSTER -
         (NAME (cluster.name.LOGCOPY2.DSØ1.NEW) -
          LINEAR                          -
          VOLUMES(*)                      -
          RECORDS(1ØØØØ) )                -
      DATA                                -
         (NAME(cluster.name.LOGCOPY2.DSØ1.NEW.DATA) )
  DEFINE CLUSTER -
         (NAME (cluster.name.LOGCOPY2.DSØ2.NEW) -
          LINEAR                          -
          VOLUMES(*)                      -
          RECORDS(1ØØØØ) )                -
      DATA                                -
         (NAME(cluster.name.LOGCOPY2.DSØ2.NEW.DATA) )
  DEFINE CLUSTER -
         (NAME (cluster.name.LOGCOPY2.DSØ3.NEW) -
          LINEAR                          -
          VOLUMES(*)                      -
          RECORDS(1ØØØØ) )                -
      DATA                                -
(NAME(cluster.name.LOGCOPY2.DSØ3.NEW.DATA) )
/*
//
```

2 Stop the QMGR and ensure it has closed down cleanly.

3 Run job two as shown below.

```
//jobname JOB #,'WINDY M.',CLASS=x,MSGCLASS=X,
//         NOTIFY=&SYSUID
//*-------------------------------
//* JOB two in Log Reallocation suite.
//*
//* STEP1 - IDCAMS
//*       Rename existing LOG datasets with .OLD suffix
//*       then put in place pre-allocated larger  LOG
//*       datasets created in the previous job
```

```
//*
//*-------------------------------------*
//STEP1    EXEC PGM=IDCAMS
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
  ALTER cluster.name.LOGCOPY1.DSØ1                  -
  NEWNAME(cluster.name.LOGCOPY1.DSØ1.OLD)
  ALTER cluster.name.LOGCOPY1.DSØ1.DATA             -
  NEWNAME(cluster.name.LOGCOPY1.DSØ1.OLD.DATA)
  ALTER cluster.name.LOGCOPY1.DSØ2                  -
  NEWNAME(cluster.name.LOGCOPY1.DSØ2.OLD)
  ALTER cluster.name.LOGCOPY1.DSØ2.DATA             -
  NEWNAME(cluster.name.LOGCOPY1.DSØ2.OLD.DATA)
  ALTER cluster.name.LOGCOPY1.DSØ3                  -
  NEWNAME(cluster.name.LOGCOPY1.DSØ3.OLD)
  ALTER cluster.name.LOGCOPY1.DSØ3.DATA             -
  NEWNAME(cluster.name.LOGCOPY1.DSØ3.OLD.DATA)
  ALTER cluster.name.LOGCOPY1.DSØ1.NEW              -
  NEWNAME(cluster.name.LOGCOPY1.DSØ1)
  ALTER cluster.name.LOGCOPY1.DSØ1.NEW.DATA         -
  NEWNAME(cluster.name.LOGCOPY1.DSØ1.DATA)
  ALTER cluster.name.LOGCOPY1.DSØ2.NEW              -
  NEWNAME(cluster.name.LOGCOPY1.DSØ2)
  ALTER cluster.name.LOGCOPY1.DSØ2.NEW.DATA         -
  NEWNAME(cluster.name.LOGCOPY1.DSØ2.DATA)
  ALTER cluster.name.LOGCOPY1.DSØ3.NEW              -
  NEWNAME(cluster.name.LOGCOPY1.DSØ3)
  ALTER cluster.name.LOGCOPY1.DSØ3.NEW.DATA         -
  NEWNAME(cluster.name.LOGCOPY1.DSØ3.DATA)
//
```

4 Run job three as shown below.

```
//jobname JOB #,'WINDY M.',CLASS=x,MSGCLASS=X,
//         NOTIFY=&SYSUID
//*-------------------------------------
//* Job three in Log Reallocation suite.
//*
//* STEP1 - IDCAMS
//*       This job performs a REPRO (copy) of the
//*       existing logs previously renamed as .OLD
//*       into the newly allocated larger logs.
//*
//*-------------------------------------*
//*
//STEP1    EXEC PGM=IDCAMS
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
  VERIFY DATASET(cluster.name.LOGCOPY1.DSØ1)
  VERIFY DATASET(cluster.name.LOGCOPY1.DSØ2)
```

```
  VERIFY DATASET(cluster.name.LOGCOPY1.DSØ3)
  REPRO IDS(cluster.name.LOGCOPY1.DSØ1.OLD)          -
        ODS(cluster.name.LOGCOPY1.DSØ1)
  REPRO IDS(cluster.name.LOGCOPY1.DSØ2.OLD)          -
        ODS(cluster.name.LOGCOPY1.DSØ2)
  REPRO IDS(cluster.name.LOGCOPY1.DSØ3.OLD)          -
        ODS(cluster.name.LOGCOPY1.DSØ3)
//
```

5 Restart QMGR and ensure restart completes successfully.

Andrew Miller
Senior Technician (UK)

Deleting expired messages

Easy, isn't it? To delete expired messages from an MQSeries queue all you need to do is an MQGET (BROWSE is sufficient on the V5.1 platforms, but a 'destructive' read is required on OS/390 up to V2.1) – right? Please read on!

The use of expired messages is an important part of ensuring that the system does not get clogged up with messages because a component in the infrastructure is down. Typically, this is true of enquiry-type messages (where the user sits at a Web browser and requests some information), when the message can't be delivered. If the user expects a response within, say, five seconds, then the application program can set the expiry time at MQPUT time to five seconds. When a message expires it can no longer be read by any application, but remember that expired messages count towards the queue depth and 'depth triggering'. You can set a 'report option' so that a message is generated by MQSeries when a message 'expires'.

Further detailed information can be obtained from the *MQSeries Application Programming Reference manual SC33-1673-08* or directly downloaded from the Web at: *http://www-4.ibm.com/software/ts/mqseries/library/manualsa/*

The situation on OS/390 is a little different. This platform offers the administrator the ability to create an index for a queue for performance

reasons. The field INDXTYPE can have several values:

MQIT_NONE	No index
MQIT_MSG_ID	Index built on message-id
MQIT_CORRELID	Index built on correllation-id
MQIT_MSG_TOKEN	Index built on message token

In fact, some of the supplied MQSeries 'system' queues on MQ for OS/390 V2.1 have just such an index: *SYSTEM.CHANNEL.SYNCQ* has a MSGID index, and *SYSTEM.CLUSTER.TRANSMIT.QUEUE* has one on CORRELID.

When I was working for a particular customer in Germany, they were getting performance problems on queues built with a 'message id' index (MQ for OS/390 V1.2). The queues had several thousand entries and they were only accessed via MQGET with the 'match options' field set to MQMO_MATCH_MSG_ID , and the 'message id' set to a specific value.

All messages put to this queue were given an appropriate 'expiry' time.

The problem with this set-up is that the 'normal' rule for deleting expired messages did not work, ie the 'MQGET with MSGID' did not delete the expired messages, and the large number of expired messages was contributing to the poor performance.The statistics showing the effect of a large queue depth *versus* small queue depth are shown in Table 1.

I found this rather interesting article on the IBM problem database:

"Assuming you wish to use the expiry methodology of message

CPU	#gets	#puts	qdepth
3:16:19	288100	139723	35K-50K
0:18:09	338487	166237	0

Table 1: Effect of large queue depth and small queue depth

removal it will only ‘disappear from the current depth’ when a potentially successful non-browse MQGET has been performed on that message and the ‘clean-up’ task has run to remove the message from the queue.

The ‘clean-up’ task (also known as scavenger) will only run when there is sufficient work for it to do (at least 100 expired messages). So, to remove the message from the queue, first of all a non-browse MQGET must be performed on the message after its expiry time has elapsed. (This will mark the message to be deleted.) Second, you must wait for the ‘clean-up’ process to run.You could do the following:

- Do an MQGET FOR MSGID AND CORRELID where you know that combination won’t exist. In this case the index is skipped because you cannot define an index on both fields. Thus, the queue is scanned completely and all expired messages are marked for deletion. The actual deletion will happen once the limit is reached for the scavenger task.”

A utility was written to do this, and it worked (for both versions 1.2 and 2.1).

There has been a recent suggestion to solve this problem in a different way, but it is rather heavy-handed and fairly expensive in terms of the processing it requires:

- Read the whole queue using a destructive MQGET under syncpoint.When the whole queue has been read, issue a **BACKOUT** to undo all the updates.

Just think of the logging that would go on if you had a large number of messages. Also, whilst the ‘update’ task runs, every other task is locked-out from reading any of the messages that have been read until the task has finished (ie it has backed-out all the destructive reads).

I understand that our colleagues in Hursley are aware of the requirement to provide a better and more efficient way of deleting these expired messages, but my opinion is that it won’t happen until the next release of MQ V5.2.

Ruud van Zundert,
Independent MQSeries Consultant (UK)

Customizing files

This article provides two examples of how files may be customized according to the requirements of your installation. The first, customizing a channel initiator options module, shows how to define the correct LU name and TCP/IP task name.

The second, CSQ4CHNL, provides an example of how the files *CSQ4INP2* and *CSQ4DISX* (samples for distributed queueing not using CICS), which are supplied in the *MQM.SCSQPROC* dataset, can be customized. This sample contains channel definitions for remote locations which would communicate with the *MQxx* sub-system on the mainframe via a CHLTYPE (SVRCONN) connection definition.

CUSTOMIZING A CHANNEL INITIATOR OPTIONS MODULE

CSQXMQxx is an example of how the file *CSQ4XPRM*, which is supplied in the *MQM.SCSQPROC* dataset, can be customized to define the correct LU name and TCP/IP task name for your system.

This job is used to create a channel initiator options module. It assembles and links a new channel initiator parameter module for distributed queueing without CICS. Edit the parameters for the CSQ6CHIP macro to determine your parameters and re-link module CSQXPARM.

An example of the customized module CSQXPARM (*CSQXMQxx*) is shown below.

JOB CSQX4PRM

```
//JOBCARD
//* Customize Channel Initiator options module
//* The Macros in this deck are tracked by OS/390 SMPE usermod
//* CUSTOMIZE THIS JOB HERE FOR YOUR INSTALLATION
//* YOU MUST DO GLOBAL CHANGES ON THESE PARAMETERS USING YOUR EDITOR
//*********************************************************************
//*               IBM MQSeries for MVS/ESA
//*    This job assembles and links a new channel initiator
//*    parameter module for distributed queueing without CICS.
//*    Edit the parameters for the CSQ6CHIP macro to determine
```

```
//*    your parameters.
//*    See "MQSeries for MVS/ESA System Management Guide"
//*    for a full description of the parameters.
//*    Replace   ++THLQUAL++
//*                      with the high level qualifier of the
//*                      SCSQMACS target library.
//*    Replace   ++HLQ.USERAUTH++
//*                      with the data set name of the authorized
//*                      load library in which to store your
//*                      channel initiator parameter module.
//*    Replace   ++NAME++
//*                      with the name of your channel initiator
//*                      parameter module.
//*                      Note - do NOT use the default version
//*                      name of CSQXPARM if you are using the
//*                      IBM library SCSQAUTH to store your
//*                      channel initiator parameter module.
//*******************************************************************
//*             Assemble step for CSQXPARM
//CHIP   EXEC PGM=ASMA90,PARM='DECK,NOOBJECT,LIST,XREF(SHORT)'
//SYSLIB   DD DSN=MQM.SCSQMACS,DISP=SHR,UNIT=3390,VOL=SER=SYS001 <=vol?
//         DD DSN=SYS1.MACLIB,DISP=SHR,UNIT=3390,VOL=SER=SYS001  <=vol?
//SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSPUNCH DD DSN=&&CHIP,
//         UNIT=SYSDA,DISP=(,PASS),
//         SPACE=(400,(100,100,1))
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
      CSQ6CHIP ADAPS=8,           ADAPTER SUBTASKS           0-9999  X
               ACTCHL=200,        MAX ACTIVE  CHANNELS       1-9999  X
               CURRCHL=200,       MAX CURRENT CHANNELS       1-9999  X
               DISPS=5,           DISPATCHERS                1-9999  X
               LUNAME=MVSMQLU1,   LU NAME FOR OUTBOUND DATA (LPARx)  X
               LU62CHL=200,       MAX LU6.2   CHANNELS       0-9999  X
               TCPCHL=200,        MAX TCP/IP  CHANNELS       0-9999  X
               TCPKEEP=NO,        TCP/IP KEEPALIVE OPTION    YES|NO  X
               TCPNAME=TCPIPx,    TCP/IP ADDRESS SPACE NAME (LPARx)  X
               TRAXSTR=YES,       START TRACE AUTOMATICALLY  YES|NO  X
               TRAXTBL=2          TRACE DATASPACE SIZE IN MB 0-2048
      END
/*
//*  LINKEDIT CSQXPARM into a parameter module.
//LKED   EXEC PGM=IEWL,COND=(0,NE),
//       PARM='SIZE=(900K,124K),RENT,NCAL,LIST,AMODE=31,RMODE=ANY'
//*   OUTPUT AUTHORIZED APF LIBRARY FOR THE NEW CHANNEL INITIATOR
//*   PARAMETER MODULE.
//*YSLMOD  DD DSN=MQM.SCSQAUTH,DISP=SHR
//SYSLMOD  DD DSN=MQM.MQxx.PARMODS,DISP=SHR       <= MQxxCHIN steplib
//SYSUT1   DD UNIT=SYSDA,DCB=BLKSIZE=1024,
//            SPACE=(1024,(200,20))
```

```
//SYSPRINT DD SYSOUT=*
//CHIP     DD DSN=&&CHIP,DISP=(OLD,DELETE)
//*
//SYSLIN   DD *
   INCLUDE CHIP
 ENTRY CSQXPARM
 NAME  CSQXMQxx(R)     MQxx channel initiator parameter module name
/*
```

CSQ4CHNL: EXAMPLE OF CUSTOMIZED CSQ4INP2(CSQ4CHNL)

```
* THIS MEMBER HAS ONLY CHANNEL DEFINITIONS REQUIRED FOR REMOTE LOCATIONS
* COPY OF MEMBER CSQ4DISX
* AMENDED Ø7/Ø2/99 removed redundant 1 remote defs
* AMENDED Ø9/1Ø9/99 added 24 remote definitions
*                  commented out 1212 remote defs
* AMENDED 24/11/99 1212 remote defs re-instated
*************************************************************
* This definitionn is for Client Attachment Feature using LU62 + TCP/IP
* SVRCONN connection defined for the OS/2, UNIX MQ system and other
* locations for MQxx subsystem.
* The channel name must be unique hence the suffix of LUC for LU62
connection
*************************************************************
DEFINE CHANNEL( 'CH.2ØØØØ.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR( 'Server connection for OS/2 via LU62' ) +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 786 )
*
DEFINE CHANNEL( 'CH.685ØØ.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR( 'Server connection for UNIX via LU62' ) +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 3438Ø )
*
DEFINE CHANNEL( 'CH.62411.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR( 'Server connection for 'name of remote location' LU62' ) +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 543210 )
*
DEFINE CHANNEL( 'CH.6714Ø.TO.MQxx' ) +
 REPLACE +
```

```
 CHLTYPE( SVRCONN ) +
 DESCR('Server connection for 'name of remote location' via TCPIP') +
 TRPTYPE( TCP ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 7654321 )
*
DEFINE CHANNEL( 'CH.64277.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR('Server connection for 'name of remote location' via LU62') +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 4536180 )
*
DEFINE CHANNEL( 'CH.68477.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR('Server connection for 'name of remote location' 100 via LU62') +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 6342312 )
*
DEFINE CHANNEL('CH.60778.TO.MQxx.LUC') +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR('Server connection for 'name of remote location' via LU62') +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 7254420 )
*
DEFINE CHANNEL( 'CH.WASV017.TO.MQxx' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR('Server connection for 'WASV017' via TCPIP') +
 TRPTYPE( TCP ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 1234567 )
*
DEFINE CHANNEL( 'CH.62371.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR( 'Server connection for 'name of remote location' via LU62' ) +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 1172606 )
*
DEFINE CHANNEL( 'CH.64810.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR( 'Server connection for 'name of remote location' LU62' ) +
```

```
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 4Ø13442)
*
DEFINE CHANNEL( 'CH.64898.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR('Server connection for 'name of remote location' via LU62') +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 4194Ø4 )
*
DEFINE CHANNEL( 'CH.6971Ø.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR('Server connection for 'name of remote location' via LU62') +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 41Ø4943)
*
DEFINE CHANNEL( 'CH.MQUNIX.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR('Server connection for 'UNIX box'  via LU62') +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 4419430)
*
DEFINE CHANNEL( 'CH.61811.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR('Server connection for 'name of remote location' via LU62') +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 194Ø443)
*
DEFINE CHANNEL( 'CH.67Ø77.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR('Server connection for 'name of remote location' via LU62') +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 3Ø44194)
*
DEFINE CHANNEL( 'CH.68177.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR( 'Server connection for 'name of remote location' via LU62' ) +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
```

```
 MAXMSGL( 43Ø4419)
*
DEFINE CHANNEL( 'CH.6421Ø.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR( 'Server connection for 'name of remote location' via LU62' ) +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 4143Ø49)
*
DEFINE CHANNEL( 'CH.41693.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR('Server connection for 'name of remote location' via LU62') +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 4194Ø4 )
*
DEFINE CHANNEL( 'CH.Ø6611.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR('Server connection for 'name of remote location' via LU62') +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 441943)
*
DEFINE CHANNEL( 'CH.1Ø646.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR('Server connection for 'name of remote location' via LU62') +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 4143Ø4 )
*
DEFINE CHANNEL( 'CH.64273.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR('Server connection for 'name of remote location' via LU62') +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL(43 419Ø4 )
*
DEFINE CHANNEL( 'CH.76747.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR('Server connection for 'name of remote location' via LU62') +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 3Ø44194)
*
```

```
DEFINE CHANNEL( 'CH.67746.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR('Server connection for 'name of remote location' via LU62') +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 43Ø4419)
*
DEFINE CHANNEL( 'CH.76111.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR('Server connection for 'name of remote location' via LU62') +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 1943Ø4 )
*
DEFINE CHANNEL( 'CH.65256.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR( 'Server connection for 'name of remote location' via LU62' ) +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 4194Ø4 )
*
DEFINE CHANNEL( 'CH.76755.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR('Server connection for 'name of remote location' via LU62') +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 1944Ø4 )
*
DEFINE CHANNEL( 'CH.87673.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR( 'Server connection for 'name of remote location' via LU62' ) +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 4341Ø4 )
*
DEFINE CHANNEL( 'CH.MQGAMSTON.TO.MQxx' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR( 'Server connection for Gamston via TCPIP' ) +
 TRPTYPE( TCP ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 3414Ø94 )
*
DEFINE CHANNEL( 'CH.56779.TO.MQxx.LUC' ) +
 REPLACE +
```

```
 CHLTYPE( SVRCONN ) +
 DESCR('Server connection for 'name of remote location' via LU62') +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 4419304 )
*
DEFINE CHANNEL( 'CH.16103.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR( 'Server connection for 'name of remote location' via LU62' ) +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 4341904 )
*
DEFINE CHANNEL( 'CH.MQREADING.TO.MQxx' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR('Server connection for Reading via TCPIP') +
 TRPTYPE( TCP ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 61921314 )
*
DEFINE CHANNEL( 'CH.10641.TO.MQxx.LUC' ) +
 REPLACE +
 CHLTYPE( SVRCONN ) +
 DESCR('Server connection for 'name of remote location' via LU62') +
 TRPTYPE( LU62 ) +
 MCAUSER( ' ' ) +
 MAXMSGL( 3041944 )
*****************************************************************
* End of CSQ4CHNL
*****************************************************************
```

Saida Davies
IBM (UK)

MQBKQLST – CSQUTIL COPY a list of queues on OS/390

This REXX script performs a SCQUTIL COPY for a list of queues to sequential datasets and outputs a JCL that can be submitted later as a job to LOAD the queues back from the files.

This is useful in case a regularly-scheduled backup is needed for certain queues or whenever many queues need to be copied to sequential files for potential subsequent restores.

The script can be used together with MQLST (see the June 2001 issue of *MQ Update*) to perform the copy for queues whose names contain a certain string (eg copy all *PROD.** queues). A sample JCL for this task is also supplied.

USAGE

Please read the *Usage notes and restrictions* section (below) before using the script.

- Parameters are separated by commas, eg:

  MQBKQLST qmgr,pfx

- Qmgr: the name of the queue manager to connect to.
- Pfx: prefix of sequential files to which queues will be copied.
- The script will read from the list of queues to copy from a DD card named QLST.
- A JCL to restore (LOAD) the queues back to their original names is output to the DD card BKQOUT.
- Prerequisites: none.
- For the sample JCL to work, the script MQLST must be installed.

Usage notes and restrictions

- This script should only run in a job – do not activate from TSO.

- The SYSIN DD is reserved because this is where CSQUTIL gets its input from – so don't use it.
- Replace MYVOL*xx* in the supplied REXX code with the volume serial on which the output files of CSQUTIL COPY are created.

EXAMPLE

The following is JCL that uses MQLST to create a list of queues and activates MQBKQLST with the created list.

- Replace *MY.EXEC.LIB* in the JCL with the PDS in which the MQBKQLST and MQLST reside.
- Replace *CSQ1* with the name of the queue manager.
- Replace *mqhlq* with the high-level qualifier of the MQSeries load library SCSQANLE in your installation (for example: *MQS.V2R1.SCSQANLE*).
- Replace *MY.SEQ.PFX* with the prefix of the sequential datasets to be created by CSQUTIL COPY.
- Replace *MY.RESTORE.JCL* with the name of the dataset to be overwritten with the JCL code to LOAD the queues.
- Replace *SOMESTR* with a string that will be used by MQLST to filter the queues that will be copied.

```
//* ADD JOB CARD HERE
//*-------------------------------------------------------------------*
//* THIS JOB COPIES ALL QUEUES WITH THE WORD 'PROJ1' IN THEIR     *
//* NAME TO BACKUP FILES WITH THE PREFIX 'MQSERIES.BACKUP'.        *
//* ONE CAN BACKUP UP TO 5Ø QUEUES, IF YOU WISH TO CHANGE THAT,   *
//* CHANGE 'DYNAMBR' PARAMETER IN STEP(2).                         *
//* STEP(1) MQLST    - LISTS ALL CONFIG QUEUES.                   *
//* STEP(2) MQBKQLST - BACKUP ALL QUEUES GOTTEN FROM STEP MQLST.  *
//* THIS JOB USES TWO REXX'S -                                     *
//*  MQLST    - TO CREATE A LIST OF QUEUES TO BACKUP.             *
//*  MQBKQLST - TO BACKUP THE LIST OF QUEUES.                     *
//* THESE REXXs USE THE FOLLOWING MODULES:                         *
//*  RXMQVC  - REXX TSO INTERFACE FOR MQSERIES SUPPLIED AS SUPPORT *
//*            PACK MA19.                                           *
//*  CSQUTIL - MQSERIES UTILITY MODULE, USED TO BACKUP THE QUEUES. *
//* DD CARDS WORTH MENTIONING:                                     *
//* STEP(1) SYSPRINT - IN HERE THE LIST OF QUEUES IS CREATED, THE *
//*                    LIST IS PASSED TO STEP(2).                  *
```

```
//* STEP(2) BKQOUT   - IN THIS FILE A JCL JOB IS CREATED THAT      *
//*                    CAN BE USED TO RECOVER ALL BACKED-UP QUEUES.*
//*-------------------------------------------------------------------*
//* STEP(1) MQLST                                                     *
//*-------------------------------------------------------------------*
//MQLST        EXEC PGM=IKJEFT1B
//SYSPROC      DD DISP=SHR,DSN=MY.EXEC.LIB
//SYSTSPRT     DD SYSOUT=*
//SYSPRINT     DD DISP=(NEW,PASS,DELETE),DSN=&&QLST,
//   SPACE=(TRK,(1,2)),UNIT=SORTWK
//SYSTSIN      DD   *
 PROF NOPREF
 MQLST CSQ1,QUEUE,SOMESTR
//*-------------------------------------------------------------------*
//* STEP(2) MQBKQLST                                                  *
//*-------------------------------------------------------------------*
//MQBKQLST     EXEC PGM=IKJEFT1B,
//    DYNAMNBR=5Ø,
//    COND=(4,LE,MQLST)
//SYSTSPRT     DD SYSOUT=*
//SYSPRINT     DD SYSOUT=*
//SYSPROC      DD DISP=SHR,DSN=MY.EXEC.LIB
//STEPLIB      DD DISP=SHR,DSN=mqhlq.SCSQANLE
//QLST         DD DISP=SHR,DSN=&&QLST
//BKQOUT       DD DISP=SHR,DSN=MY.RESTORE.JCL
//SYSTSIN      DD *
 PROF NOPREF
 MQBKQLST CSQ1,MY.SEQ.PFX
/*-REXX---------------------------------------------------------------*/
/*                          MQBKQLST                                 */
/*-------------------------------------------------------------------*/
/* DESCRIPTION :                                                     */
/*  This REXX does a CSQUTIL COPY from list of queues to             */
/*  sequential files.                                                */
/*  IT IS TO BE USED FROM JOBS ONLY.                                 */
/*  The list is received as input from DD card QLST.                 */
/*  It also creates an output file to DD card BKQOUT, this file      */
/*   contains a job that can be used to restore queues from the      */
/*   sequential datasets.                                            */
/* USAGE:                                                            */
/*  MQBKQLST <qmgrname>,<filesprfx>                                  */
/*   qmgrname   - MQSeries Queue manager name                        */
/*   filesprfx  - PS backup files prefix                             */
/* EXAMPLE OF USAGE :                                                */
/*  MQBKQLST QM1,MQACID.BACKUP                                       */
/*  the script will backup all queue in QM1 from QLST list to        */
/*  files:                                                           */
/*  MQACID.BACKUP.QM1.QBKUPxxx (xxx signifies queue number)          */
/*  and will create an output file to BKQOUT.                        */
/* DD CARDS:                                                         */
```

```
/*  QLST     LIST OF QUEUES TO BE BACKED UP.                      */
/*  BKQOUT   JOB USED TO RESTORE FILES FROM PS DATASETS.          */
/*  SYSIN    (INTERNAL USE, DO NOT ALLOC)                         */
/* REXXS USED:                                                   */
/*  NONE.                                                        */
/* MODULES USED:                                                 */
/*  CSQUTIL - Used with COPY command in order to copy queues     */
/*            contents to sequential files.                      */
/* RETURN VALUE:                                                 */
/*  highest CSQUTIL RC gotten.                                   */
/* Updates                                                       */
/* 31/10/2000 MQACID RRNO                                        */
/*            Creation                                           */
/* 22/11/2000 MQACID RRNO                                        */
/*            Customized for working from JCL only               */
PARSE ARG qmgr_name','files_prfx
/* uncomment for testing...                                     */
qmgr_name     = 'QM1';
files_prfx    = 'MQSERIES.BACKUP'
   /*                 Constants Definition                      */
   /*                                                           */
   TRUE                   = 1;
   FALSE                  = 0;
   NULL                   = '';
   INDEX_STRING_NOT_FOUND = 0;
   NEWF_PARM              = 'SPACE(1,2) CYL DSORG(PS) VOL(MYVOLxx)';
   /*                Variables Initialization                   */
   /*                       MAIN                                */
  /* Strip parameters                                          */
   qlist_file  = STRIP(qlist_file);
   qmgr_name   = STRIP(qmgr_name);
   files_prfx  = STRIP(files_prfx);
   maxcc       = STRIP(maxcc);
   /* get qlist from file                                       */
   get_qlist_ok = get_qlist();
   /* backup qlist to files                                     */
   backup_qlist_rc =,
      backup_qlist(qmgr_name,files_prfx,NEWF_PARM);
return (backup_qlist_rc);
/*-------------------------------------------------------------*/
/*                     backup_qlist                            */
/*-------------------------------------------------------------*/
/* DESCRIPTION:                                                  */
/*  This procedure backs up a list of queues (in qlist. stem).   */
/*  using CSQUTIL, of course.                                    */
/* PARAMETERS:                                                   */
/*  qmgr_name  - MQSeries QManager name.                         */
/*  files_prfx - Backup files prefix.                            */
/*  NEWF_PARM  - parameters for PS datasets (in which queues are */
/*               backed up).                                     */
```

```
/* RETURN VALUE:                                                  */
/*  Highest return code gotten from CSQUTIL.                      */
backup_qlist:    procedure,
                 expose TRUE,
                 expose FALSE,
                 expose INDEX_STRING_NOT_FOUND,
                 expose qlist.;
PARSE ARG qmgr_name,files_prfx,NEWF_PARM;
   /*                 Constants Definition                       */
   BACKUPPRFX          = 'QBKUP';
   CSQUTIL_INPDD       = 'SYSIN';
   CSQUTIL_INPFILE     = USERID()'.BQLSTOUT';
   /*                Variables Initialization                    */
   /*                      MAIN                                  */
   /* Allocate necessary files                                   */
   /* Allocate CSQUTIL SYSIN file                                */
   tmpfile_parms = 'SPACE(1,2) TRACKS DSORG(PS) UNIT(SORTWK) NEW'
   alloc_ok =,
      alloc_file(CSQUTIL_INPFILE,CSQUTIL_INPDD,tmpfile_parms);
   /* Table Heading                                              */
   say LEFT('',80,'-');
   say LEFT('QUEUE',40,' ')||'BACKUP FILE';
   say LEFT('',80,'-');
   /* Go through all queues, allocate them and create dd list,  */
   /*  jclddlist, copyqlist and loadqlist                        */
   ddlist.0    = qlist.0;
   jclddlist.0 = qlist.0;
   loadqlist.0 = qlist.0;
   copyqlist.0 = qlist.0;
   do currq = 1 to qlist.0;
      /* create new file name for queue backup                   */
      currddname = BACKUPPRFX||RIGHT(currq,3,'0');
      bckfilename = files_prfx'.'qmgr_name'.'currddname;
      /* Allocate backup file                                    */
      alloc_ok =,
         alloc_file(bckfilename,currddname,NEWF_PARM);
      /* Create files dd list                                    */
      ddlist.currq = currddname;
      jclddlist.currq =,
         '//'currddname' DD DISP=OLD,DSN='bckfilename;
      copyqlist.currq =,
         'COPY QUEUE('qlist.currq') DDNAME('currddname')';
      loadqlist.currq =,
      LEFT('LOAD QUEUE('qlist.currq')',40,' ')||'DDNAME('currddname')';
      /* Notify user                                             */
      say LEFT(qlist.currq,40,' ')||bckfilename;
   end;
   /* Table Bottom                                               */
   say LEFT('',80,'-');
   /* Write copyqlist to CSQUTIL input file                      */
   "EXECIO * DISKW "CSQUTIL_INPDD" (STEM copyqlist. FINIS";
```

```
   copyqlist_RC = RC;
   /* Execute CSQUTIL cmd                                          */
   "CALL 'LNKLST.MQSERIES.V1R2MØ.SCSQAUTH(CSQUTIL)' '"qmgr_name"'";
   CSQUTIL_RC = RC;
   /* Write output file                                            */
   write_outfile_ok = write_outfile(qmgr_name);
   /* FREE all allocated files                                     */
   do currdd = 1 to ddlist.Ø;
      "FREE F("ddlist.currdd")";
   end;
   /* Free CSQUTIL input file                                      */
   "FREE F("CSQUTIL_INPDD")";
return (CSQUTIL_RC);
/*                       write_outfile                           */
/* DESCRIPTION:                                                   */
/*  This proc writes output file. The output file is a job that   */
/*  can be used to restore queues from their back-up files.       */
/* PARAMETERS:                                                    */
/*  qmgr_name  - MQSeries QManager name.                          */
/* RETURN VALUE:                                                  */
/*  TRUE/FALSE - success of writing to outfie.                    */
write_outfile:
PARSE ARG qmgr_name;
   /*                 Constants Definition                       */
   OUTPUT_FILEDD       = 'BKQOUT';
   /*                Variables Initialization                    */
   proc_success   = FALSE;
   /*                        MAIN                                */
   /* Create output stem to be written into output file          */
   next_outline  = 1;
   output_stem.Ø = Ø;
   /* put EXEC dd card to output stem                            */
   output_stem.next_outline =,
      "//COPYQ EXEC PGM=CSQUTIL,PARM='"qmgr_name"'"
   next_outline = next_outline + 1;
   /* STEPLIB statement                                          */
   output_stem.next_outline =,
      "//STEPLIB DD DISP=SHR,DSN=MQSERIES.V1R2MØ.SCSQANLE"
   next_outline = next_outline + 1;
   /* SYSPRINT statement                                         */
   output_stem.next_outline =,
      "//SYSPRINT DD SYSOUT=*";
   next_outline = next_outline + 1;
   /* put Config Queue's DD statements                           */
   do currdd = 1 to jclddlist.Ø;
      output_stem.next_outline = jclddlist.currdd;
      next_outline = next_outline + 1;
   end;
   /* put SYSIN card statement                                   */
   output_stem.next_outline = '//SYSIN DD *'
   next_outline = next_outline + 1;
```

```
   /* put jcl dd list to output stem                              */
   do currload = 1 to loadqlist.Ø;
      output_stem.next_outline = loadqlist.currload;
      next_outline = next_outline + 1;
   end;
   /* put end dd file to output stem                           */
   output_stem.next_outline = '/*'
   next_outline = next_outline + 1;
   /* set lines number in output stem                          */
   output_stem.Ø = next_outline - 1;
   /* Write output stem to output file                         */
   /*                                                          */
   "EXECIO * DISKW "OUTPUT_FILEDD" (STEM output_stem. FINIS";
   execio_RC = RC
   if (execio_RC <= 4) then,
      proc_success = TRUE;
   else,
      proc_success = FALSE;
return (proc_success);
/*                       alloc_file                             */
/* DESCRIPTION:                                                 */
/*  This procedure allocates a file to a DD card, if the file   */
/*  does not exists, it creates it.                             */
/* PARAMETERS:                                                  */
/*  filename  - Dataset name.                                   */
/*  filedd    - DD card name.                                   */
/*  new_parms - New file parameters (SPACE, UNIT etc)           */
//* RETURN VALUE:                                               */
alloc_file:      procedure,
                 expose TRUE,
                 expose FALSE,
                 expose INDEX_STRING_NOT_FOUND;
PARSE ARG filename,filedd,new_parms;
   /*                 Constants Definition                     */
   DSN_FOUND          = 'OK';
   DSN_NOTFOUND       = 'DATASET NOT FOUND';
   /*                Variables Initialization                  */
   proc_success   = FALSE;
   /*                       MAIN                            */
   /* Check if bckfilename exists and act accordingly         */
   sysdsn_res = SYSDSN(filename);
   if (sysdsn_res = DSN_FOUND) then do;
      /* allocate existing file                               */
      alloc_cmd =,
         "ALLOC F("filedd") DA("filename") SHR REUSE";
      alloc_cmd;
      alloc_rc = RC;
   end;
   else if (sysdsn_res = DSN_NOTFOUND) then do;
      /* allocate new file                                    */
```

```
      alloc_cmd =,
         "ALLOC F("filedd") DA("filename") NEW KEEP CAT "new_parms;
      alloc_cmd;
      alloc_rc = RC;
   end;
   else do;
      say 'ERROR alloc file <'filename'> to dd <'filedd'>';
      say sysdsn_res;
   end;
  /* Check alloc RC and return allocation's success           */
   /*                                                         */
   if (alloc_rc <= 4) then,
      proc_success = TRUE;
   else,
      proc_success = FALSE;
return (proc_success);
/*                        get_qlist                          */
/* DESCRIPTION:                                              */
/*  this procedure gets all queues from qlist file.          */
/* PARAMETERS:                                               */
/*  NONE.                                                    */
* RETURN VALUE:                                             */
/*  TRUE/FALSE - success of getting qlist.                   */
get_qlist:       procedure,
                 expose TRUE,
                 expose FALSE,
                 expose INDEX_STRING_NOT_FOUND,
                 expose qlist.;
   /*                Constants Definition                   */
   SYSINDD        = 'QLST'
   /*                Variables Initialization               */
   proc_success   = FALSE;
   /*                       MAIN                            */
   /* Get lines from file                                   */
   "EXECIO * DISKR "SYSINDD"(STEM file_qlist. FINIS";
   /* Take only valid lines from file                       */
   qlist.0 = 0;
   do currq = 1 to file_qlist.0
      if (LENGTH(STRIP(file_qlist.currq)) ^= 0) then do;
         nextline = qlist.0 + 1;
         qlist.nextline = STRIP(file_qlist.currq);
         qlist.0 = qlist.0 + 1;
      end;
   end;
return (proc_success);
```

MQSeries message rate driver

PURPOSE OF THE PROGRAM

The *mqsdriver* program is used to measure the maximum message rates of a process which is driven by messages arriving on a queue. The standard method of measuring such a process is by loading up a queue with many thousands of messages, starting the process, and then watching for the current depth to reach zero. It's simple and effective, but a little crude. A better method would be keep a steady supply of messages on the input queue and report the rate at which messages are supplied.

METHOD

mqsdriver needs just one model message on the target queue to start the process off. At its simplest, this could be created using MQ Explorer functions. Complex messages will require more complex solutions. As in all the best science books, I leave this problem to the reader.

mqsdriver reads the model message on the target queue. Every ten milliseconds it attempts to put the lowest number of copies of the model message on the queue so that the current depth does not reach zero. The actual message rate is reported every ten seconds.

Execution of *mqsdriver*

mqsdriver is a C program with the following parameters:

- Qname (mandatory).
- QMgrname (optional).
- Report interval in seconds (optional).

Operation of *mqsdriver*

- Start *mqsdriver* and name the input queue (this waits on the input queue).

- Place the message to be replicated on the input queue.
- Start the process to be measured (this reads the input queue).
- Observe the message rate displayed every ten seconds.

PLATFORMS

The program has been used on Windows NT, RS/6000, and Sun platforms. There are minor differences in operation on these platforms so attention should be paid to the *#define* setting at the beginning of the program.

CODE

```
 /* Program name: MQSDRIVER                                          */
 /* Function:                                                        */
 /*   MQSDRIVER is a C program to put messages on a                  */
 /*   message queue                                                  */
 /*      --  reads with browse 1 message from the                    */
 /*          queue named in the  1st parameter                       */
 /*      --  Q manager is the optional second parameter              */
 /*      --  report interval in seconds is the optional 3rd parameter*/
 /*      --  enough messages are placed on  the queue to drive a     */
 /*          workload                                                */
/*      --  stops if there is a MQI completion code                 */
 /*         of MQCC_FAILED                                          */
 #include <stdio.h>
 #include <stdlib.h>
 #include <string.h>
     /* includes for MQI  */
 #include <cmqc.h>
 #include <time.h>
/*#define AIXCOMP *** set this for UNIX*******************/
/* This test is on an NT machine */
 #define NTCOMP
 #ifdef NTCOMP
 #include <windows.h>
 #endif
 int main(int argc, char **argv)
 {
   /*   Declare MQI structures needed                              */
   MQOD     od = {MQOD_DEFAULT};    /* Object Descriptor           */
   MQMD     md = {MQMD_DEFAULT};    /* Message Descriptor          */
   MQGMO   gmo = {MQGMO_DEFAULT};   /* get message options         */
   MQPMO   pmo = {MQPMO_DEFAULT};   /* put message options         */
   MQCNO    Connect_options = {MQCNO_DEFAULT}; /* MQCONNX options   */
```

```
MQHCONN  Hcon;                  /* connection handle            */
MQHOBJ   Hobji;                  /*input object handle          */
MQLONG   O_optionsi;             /*input MQOPEN options         */
MQHOBJ   Hobjo;                  /*output object handle         */
MQLONG   O_optionso;             /*output MQOPEN options        */
MQHOBJ   Hinq;                  /*inquire object handle         */
MQLONG   O_optionsq;             /*inquire MQOPEN options       */
MQLONG   C_options;             /* MQCLOSE options             */
MQLONG   CompCode;              /* completion code             */
MQLONG   Reason;                /* reason code                 */
MQLONG   CReason;               /* reason code for MQCONN      */
MQBYTE   buffer[66000];         /* message buffer              */
MQLONG   buflen;                /* buffer length               */
MQLONG   messlen;               /* message length received     */
char     QMName[50];            /* queue manager name          */
MQLONG   Select[1];             /* attribute selectors         */
MQLONG   IAV[1];                /* integer attribute values    */
MQLONG   numOnQueue;            /* Number of messages on Queue */
MQLONG   numOfPuts;             /* Number of messages to Put   */
MQLONG   i;                     /* count of puts               */
MQLONG   initialNumOnQueue;     /* initial Number of messages  */
MQLONG   multiply;              /* multiplication factor       */
MQLONG   totalMsgsPut;          /* Number of messages read     */
double   msgRate;
MQLONG   msgsInSlot;
MQLONG   countAtLastSlot;
time_t   current_time;
time_t   last_report_time;
MQLONG   elapsedTime;
MQLONG   reportDuration;
printf("MQSDRIVER start\n");
if (argc < 2)
{
  printf("Required parameter missing - queue name\n");
  exit(99);
}
/*   Create object descriptor for subject queue                 */
strcpy(od.ObjectName, argv[1]);
QMName[0] = 0;   /* default */
if (argc > 2)
  strcpy(QMName, argv[2]);
/*   Connect to queue manager                                   */
Connect_options.Options = MQCNO_FASTPATH_BINDING;
MQCONNX(QMName,                  /* queue manager               */
       &Connect_options,        /* connect options             */
       &Hcon,                   /* connection handle           */
       &CompCode,               /* completion code             */
       &CReason);               /* reason code                 */

/* report reason and stop if it failed     */
```

```
   if (CompCode == MQCC_FAILED)
   {
     printf("MQCONN ended with reason code %ld\n", CReason);
     exit( (int)CReason );
   }
   /*   Open the named message queue for input; shared               */
   O_optionsi = MQOO_INPUT_SHARED    /* open queue for input         */
         + MQOO_FAIL_IF_QUIESCING
         + MQOO_BROWSE;             /* but not if MQM stopping      */
   MQOPEN(Hcon,                      /* connection handle            */
          &od,                       /* object descriptor for queue  */
          O_optionsi,                 /* open options                */
          &Hobji,                     /* object handle               */
          &CompCode,                 /* completion code              */
          &Reason);                  /* reason code                  */
   /* report reason, if any; stop if failed      */
   if (CompCode == MQCC_FAILED)
   {
     printf("MQOPEN 1 ended with reason code %ld\n", Reason);
     exit( (int)Reason );
   }
   /*   Open the target message queue for output                     */
   O_optionso = MQOO_OUTPUT           /* open queue for output       */
           + MQOO_FAIL_IF_QUIESCING ;
     /*      + MQOO_SAVE_ALL_CONTEXT; */ /* but not if MQM stopping
*/
   /* + MQOO_PASS_ALL_CONTEXT */
   MQOPEN(Hcon,                      /* connection handle            */
          &od,                       /* object descriptor for queue  */
          O_optionso,                 /* open options                */
          &Hobjo,                     /* object handle               */
          &CompCode,                 /* MQOPEN completion code       */
          &Reason);                  /* reason code                  */
   /* report reason, if any; stop if failed      */
   if (CompCode == MQCC_FAILED)
   {
     printf("MQOPEN 2 ended with reason code %ld\n", Reason);
     exit( (int)Reason );
   }
   /*   Open named queue for INQUIRE                                 */
   O_optionsq = MQOO_INQUIRE    /* open to inquire attributes       */
              + MQOO_FAIL_IF_QUIESCING;
   MQOPEN(Hcon,                /* connection handle                */
          &od,                /* object descriptor for queue      */
          O_optionsq,           /* open options                    */
          &Hinq,               /* object handle for MQINQ          */
          &CompCode,           /* MQOPEN completion code           */
          &Reason);            /* reason code                      */
   /* report reason, if any; stop if failed      */
   if (CompCode == MQCC_FAILED)
```

```
{
  printf("MQOPEN 3 ended with reason code %ld\n", Reason);
  exit( (int)Reason );
}
/*   Get a seed message from the message queue                    */
gmo.Version = MQGMO_VERSION_2;     /* Avoid need to reset Message */
gmo.MatchOptions = MQMO_NONE;      /* ID and Correlation ID after */
                                   /* every MQGET                 */
gmo.Options = MQGMO_WAIT;       /* wait for new messages          */
gmo.Options += MQGMO_BROWSE_NEXT ;
gmo.WaitInterval = MQWI_UNLIMITED;      /* no limit for waiting   */
if (argc > 3)
  reportDuration = atoi(argv[3]);
else
   reportDuration = 10;
totalMsgsPut = 0;
time(&last_report_time);
countAtLastSlot= 0;
buflen = sizeof(buffer) - 1; /* buffer size available for GET   */
md.Encoding       = MQENC_NATIVE;
md.CodedCharSetId = MQCCSI_Q_MGR;
MQGET(Hcon,                /* connection handle                */
      Hobji,               /* object handle                    */
      &md,                 /* message descriptor               */
      &gmo,                /* get message options              */
      buflen,              /* buffer length                    */
      buffer,              /* message buffer                   */
      &messlen,            /* message length                   */
      &CompCode,           /* completion code                  */
      &Reason);            /* reason code                      */
/* report reason, if any; stop if failed      */
if (CompCode == MQCC_FAILED)
{
  printf("MQGET ended with reason code %ld\n", Reason);
  exit( (int)Reason );
}
numOfPuts = 16;
multiply = 1;
initialNumOnQueue = 16;
while (CompCode != MQCC_FAILED)
{
  Select[0] = MQIA_CURRENT_Q_DEPTH;
  MQINQ(Hcon,        /* connection handle                */
    Hinq,            /* object handle                    */
    1L,              /* Selector count                   */
    Select,          /* Selector array                   */
    1L,              /* integer attribute count          */
    IAV,             /* integer attribute array          */
    0L,              /* character attribute count        */
    NULL,            /* character attribute array        */
```

```
       /*  note - can use NULL because count is zero         */
       &CompCode,       /* completion code                   */
       &Reason);        /* reason code                       */
     if (CompCode == MQCC_OK)
     {
       numOnQueue= IAV[Ø];   /* currdepth */
/*      printf("curdepth is now %ld\n", numOnQueue); */
       if (numOnQueue <= multiply * initialNumOnQueue )
       {
         if (numOnQueue == Ø)
         {
           multiply = multiply*2;
           printf("changing number of puts to %ld\n", multiply *
initialNumOnQueue);
         }
         numOfPuts = multiply * initialNumOnQueue;
       }
       else {
         numOfPuts =Ø;
       }
     }
     else  {
       printf("MQINQ ended with reason code %ld\n", Reason);
       exit( (int)Reason );
     }
      /* pmo.Options |= MQPMO_PASS_ALL_CONTEXT; */
     pmo.Options |= MQPMO_NEW_MSG_ID;
     pmo.Options |= MQPMO_NEW_CORREL_ID;

     for (i = Ø; i < numOfPuts; i++)
     {
       MQPUT(Hcon,              /* connection handle              */
             Hobjo,             /* object handle                  */
             &md,               /* message descriptor             */
             &pmo,              /* default options (datagram)     */
             messlen,           /* message length                 */
             buffer,            /* message buffer                 */
             &CompCode,         /* completion code                */
             &Reason);          /* reason code                    */
       totalMsgsPut++;   /* Increment the count */
       if (CompCode == MQCC_FAILED)
       {
         printf("MQPUT ended with reason code %ld\n", Reason);
         exit( (int)Reason );
       }
     }
     time(&current_time);
     if ( (elapsedTime = (current_time - last_report_time) ) >=
reportDuration )
     {
```

```
        msgsInSlot = totalMsgsPut - countAtLastSlot;
        countAtLastSlot = totalMsgsPut;
        msgRate = (double) msgsInSlot / (double) elapsedTime;
        last_report_time = current_time;
        printf("Slot duration is %d seconds, msg rate is %3.3f, total
msgs=%d\n",elapsedTime,msgRate, totalMsgsPut);
      }
      #ifdef NTCOMP
       Sleep(1Ø); /*sleep in millisecs for NT */
      #endif
      #ifdef AIXCOMP
        usleep(1ØØØØ); /*sleep in microsecs*/
      #endif
     }
    /*   Close the source queue (if it was opened)                    */
    C_options = Ø;                  /* no close options             */
    MQCLOSE(Hcon,                   /* connection handle            */
            &Hobji,                  /* object handle               */
            C_options,
            &CompCode,              /* completion code              */
            &Reason);               /* reason code                  */
    /* report reason, if any     */
    if (Reason != MQRC_NONE)
    {
      printf("MQCLOSE ended with reason code %ld\n", Reason);
    }
    MQCLOSE(Hcon,                   /* connection handle            */
            &Hobjo,                  /* object handle               */
            C_options,
            &CompCode,              /* completion code              */
            &Reason);               /* reason code                  */
    /* report reason, if any     */
    if (Reason != MQRC_NONE)
    {
      printf("MQCLOSE ended with reason code %ld\n", Reason);
    }
    MQCLOSE(Hcon,                   /* connection handle          */
            &Hinq,                  /* object handle              */
            C_options,
            &CompCode,              /* completion code            */
            &Reason);               /* reason code                */
     /* report reason, if any     */
    if (Reason != MQRC_NONE)
    {
      printf("MQCLOSE ended with reason code %ld\n", Reason);
    }
    /*   Disconnect from MQM if not already connected                 */
    if (CReason != MQRC_ALREADY_CONNECTED )
    {
      MQDISC(&Hcon,                    /* connection handle          */
```

```
            &CompCode,                  /* completion code            */
            &Reason);                   /* reason code                */
      /* report reason, if any     */
      if (Reason != MQRC_NONE)
      {
        printf("MQDISC ended with reason code %ld\n", Reason);
      }
    }
    /* END OF MQSDRIVER                                                */
    printf("MQSDRIVER end\n");
    return(0);
}
```

R E Branagan
Programmer, IBM Hursley (UK)

Need help with an *MQSeries* problem or project?

Maybe we can help:

- If it's on a topic of interest to other subscribers, we'll commission an article on the subject, which we'll publish in *MQ Update*, and which we'll pay for – it won't cost you anything.
- If it's a more specialized, or more complex, problem, you can advertise your requirements (including one-off projects, freelance contracts, permanent jobs, etc) to the hundreds of MQSeries professionals who visit *MQ Update*'s home page every week. This service is also free of charge.

Visit the *MQ Update* Web site, http://www.xephon.com/mqupdate.html, and follow the link to Suggest a topic or Opportunities for MQSeries specialists.

Finding the maximum message length of a queue

INTRODUCTION

This article provides an illustration of a C function that returns the maximum message length a queue can accept. It works for all usable types of queue, including alias, cluster, and remote.

When putting large messages to a queue the maximum message length a queue can accept must be known. If the target queue is local, only a single MQINQ() call is required. But if the queue is of any other type, such as a remote queue, a cluster queue, or an alias, several calls need to be performed.

For example, let's say a remote queue – *RQ1* – points to the transmission queue *XQ1*, which has a MAXMSGLEN attribute of 64K. *RQ1* can only accept messages up to 64K in length. Let's say an alias – *ARQ1* – points to this remote queue.

In order to find out the maximum message length *ARQ1* can accept, four MQINQ() calls need to be performed:

- Find out the target queue name that *ARQ1* points to (*RQ1*).
- Get the type of *RQ1* (remote).
- Find out which transmission queue *RQ1* points to (*XQ1*).
- Get the maximum message length of *XQ1* (64K).

Add to that the MQOPEN() calls required to get the MQHOBJ of each queue to inquire, and you'll get a lot of code.

The following handy C function uses recursion to get the FINAL maximum message length any queue can accept. It was compiled and tested on several Unix flavours, on Windows NT, and on OS/390. It supports every queue type that exists in MQSeries 5.1 except model queues, which are not usable for obvious reasons.

```
int mqc_q_get_maxmsglen(
        MQHCONN  Hconn,
```

```
        MQHOBJ   Hq,
        MQLONG   *maxmsglen_,
        MQLONG   *mq_cc_,
        MQLONG   *mq_rc_)
```

PARAMETER DESCRIPTION

- *Hconn*: Input; queue manager connection handle as returned from MQCONN().
- *Hq*: Input; queue handle of the target queue, opened for inquiry as returned from MQOPEN().
- *maxmsglen_*: Output; returns the maximum message length the queue can accept, or zero in case of a failure.
- *mq_cc_*, *mq_rc_*: Output; values returned from MQSeries in case of a failure.

USAGE NOTES

- As stated, the target queue handle must be pre-opened for inquiry. We do it this way (instead of accepting a queue name, for example) because the user will probably re-use the same handle subsequently, in order to put a message to the queue.
- The maximum size of the queue is influenced by two outside factors: the MAXMSGLEN attribute of the queue manager (shown by the MQSC command **DIS QMGR**) and, in the case of client connections that use SVRCONN and CLNTCONN channels, by the values of MAXMSGLEN in the channels. This function (and MQINQ() in general) returns the MINIMUM of all these values.

  For example, if a queue has a 100MB maximum length, a program connecting through a client connection channel that has a 100MB maximum length and a SVRCONN channel that has 4MB maximum length, will 'see' the minimal value – 4MB.

```
/**
 ** mqc_q_get_maxmsglen(): get maxmsglen attribute of a specified queue.
 ** deals with local, alias, remote and cluster queues.
 ** model queues are not supported.
int mqc_q_get_maxmsglen(
```

```
        MQHCONN  Hconn,                  /* connection handle
*/
         MQHOBJ   Hq,                   /* queue handle MQOPENed for inq
*/
        MQLONG   *maxmsglen_,
        MQLONG   *mq_cc_,
        MQLONG   *mq_rc_)
{
     MQLONG   Selectors[1];
     MQLONG   qtype;
    MQOD     od = {MQOD_DEFAULT};
    MQLONG   open_options;
    MQLONG   close_options;
     MQHOBJ   target_Hq;
     MQCHAR   next_qname[MQ_Q_NAME_LENGTH];
     *maxmsglen_ = 0;
     /* find out q type */
     Selectors[0] = MQIA_Q_TYPE;
     MQINQ(Hconn, Hq, 1, Selectors, 1, &qtype, 0, NULL, mq_cc_, mq_rc_);
     if (*mq_cc_ != MQCC_OK)
          return 0;
     switch(qtype)
     {
     case MQQT_LOCAL:
          /* get maxmsglen */
          Selectors[0] = MQIA_MAX_MSG_LENGTH;
          MQINQ(Hconn, Hq, 1, Selectors, 1, maxmsglen_, 0, NULL, mq_cc_,
mq_rc_);
          if (*mq_cc_ != MQCC_OK)
               return 0;
          break;
          case MQQT_ALIAS:
          /* get base q name (targq) */
          Selectors[0] = MQCA_BASE_Q_NAME;
          MQINQ(Hconn, Hq, 1, Selectors, 0, NULL, MQ_Q_NAME_LENGTH,
next_qname, mq_cc_, mq_rc_);
          if (*mq_cc_ != MQCC_OK)
               return 0;
          break;
     case MQQT_CLUSTER:
          /* point to cluster xmitq */
          strncpy(
               next_qname,
               “SYSTEM.CLUSTER.TRANSMIT.QUEUE”,
               (size_t)MQ_Q_NAME_LENGTH);
          break;
     case MQQT_REMOTE:
          /* get xmitq name */
          Selectors[0] = MQCA_XMIT_Q_NAME;
          MQINQ(Hconn, Hq, 1, Selectors, 0, NULL, MQ_Q_NAME_LENGTH,
```

```
next_qname, mq_cc_, mq_rc_);
          if (*mq_cc_ != MQCC_OK)
               return 0;
          break;
     /* unsupported q type */
     default:
          *mq_cc_ = MQCC_OK;
          *mq_rc_ = MQRC_NONE;
          return 0;
     }
     /* if still didn't get maxmsglen, another q needs to be inquired */
     if(*maxmsglen_ == 0)
     {
          /* open target q */
          strncpy(od.ObjectName, next_qname, (size_t)MQ_Q_NAME_LENGTH);
          open_options = MQOO_INQUIRE + MQOO_FAIL_IF_QUIESCING;
          MQOPEN(Hconn,              /* connection handle            */
                 &od,                /* object descriptor for queue  */
                 open_options,       /* open options                 */
                 &target_Hq,         /* object handle                */
                 mq_cc_,             /* MQOPEN completion code       */
                 mq_rc_);            /* reason code                  */
          if(*mq_cc_ != MQCC_OK)
               return 0;
          /* recurse to get maxmsglen of target q */
  if(!mqc_q_get_maxmsglen(Hconn, target_Hq, maxmsglen_, mq_cc_, mq_rc_))
               return 0;
          /* close target q */
          close_options = MQCO_NONE;
          MQCLOSE(Hconn,
                   &target_Hq,
                    close_options,
                    mq_cc_,
                    mq_rc_);
          if(*mq_cc_ != MQCC_OK)
               return 0;
     }
     return 1;
}
```

Roy Razon
MQSeries Consultant (Israel)

MQ news

IBM has announced a raft of enhancements to the MQSeries family, including an updated version of MQSeries for NT and Windows 2000 and a new release of the Application Messaging Interface (AMI). AMI Version 1.2 now has an open policy-handler framework, which allows the functionality of the AMI to be extended.

MQSeries Java Message Service Version 5.2 includes new features including two-phase commit and additional performance improvements. Internet pass-thru Version 1.1 can be used to implement messaging solutions between remote sites across the Internet. It also has new security features and administration.

Meanwhile, High Availability SupportPacs provide examples and functions to utilize high availability support on AIX, Compaq Tru64, HP-UX, Solaris, and Windows NT and 2000 platforms. Support for MQSeries V5.2 is extended to OS/400 V5R1 and AIX V5R1.

IBM has previewed availability of MQSeries JMS support on OS/390 and iSeries and of AMI V1.2 (which will replace V1.1) on iSeries.

IBM has also announced MQSeries Integrator V2.0.2, which extends support to the HP-UX and iSeries 400 platforms in addition to AIX, Solaris, and Windows NT. It also provides national language support for ten languages, has MQSeries Everyplace and SCADA protocol support, includes integration of NEON V5.2 Rules and Formatter, and a new visual message flow debugger. There's also improved performance of the Message Repository Manager (MRM).

For further information, contact your local IBM representative.

* * *

BMC Software has enhanced its support for IBM's MQSeries Integrator (MQSI) with the release of Patrol for MQSI 1.3. The new version allows customers to collect and report on message flow statistics via the Message Flow Statistics Collection module, a customizable node that provides information such as average and last duration, average and last size, and other statistics.

Patrol for MQSI 1.3 also includes a new feature that enables customers to activate rules to automatically restart components such as message brokers, the configuration manager, the user name server, and listeners when the system detects that they have stopped. Other features include user and service trace formatting enhancements, improved discovery performance, and filtering of events by event ID.

For further information, contact:
BMC Software, 2101 City West Blvd, Houston, Texas, 77042-2827, USA
Tel: +1 800 841 2031
Fax: +1 713 918 8000
Web: http://www.bmc.com

BMC Software, Vicarage Road, Assurance House, Egham, Surrey, TW20 9JY, UK
Tel: +44 1784 478000
Fax: +44 1784 430581