41

MQ update

November 2002

In this issue

3 Backing up a Unix queue manager that uses linear logging

11 WebSphere MQ V5.3 clustering: hints and tips

21 Stand-alone MQSeries Workflow client/server setup on Windows NT

36 Creating and using generic profiles

48 MQ news

MQ 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

Editor
Madeleine Hudson
E-mail: MadeleineH@xephon.com

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 ($33.75) each including postage.

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/nfc.

***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/mq; you will need to supply a word from the printed issue.

Backing up a Unix queue manager that uses linear logging

In common with all production IT systems one of the housekeeping functions that you will need to do on a regular basis for your Unix queue managers is to back up the file systems they use. If you've followed IBM's recommendations there will be two such file systems: one for the queue manager's data and one for its logs. It is vitally important that these two are completely in sync when a backup is taken, and the mqbackup script and those it calls will ensure this.

In order to prevent any new messages arriving whilst the backup is taken, all channels are stopped and the queue manager is ended. Immediately prior to its being brought down, the system 'record media image' command **rcdmqimg** is run to write an image of all the queues to the logs (for use in media recovery).

The channels are stopped by routing the output of an MQSC DISPLAY CHANNEL command, enhancing it with awk, and writing it to a temporary file that will be piped back into MQSC. In these scripts three such files will be created: one that issues a normal STOP CHANNEL command, one that repeats this with MODE(FORCE) for any that are still up, and one that issues START CHANNEL commands once the backup has completed and the queue manager has been restarted.

Once the queue manager is back up and running, the linear logs can be cleaned up using the *cleanmqlogs* script from the MS62 Support Pack. This highlights any logs that the queue manager no longer needs by compressing them. Any such compressed files that are more than seven days old are automatically erased.

Here is the console output from running mqbackup.

```
MQBACKUP
$ mqbackup MYQMGR
Ø783889, 5765-B75 (C) Copyright IBM Corp. 1994, 2ØØØ.  ALL RIGHTS
RESERVED.
Starting MQSeries Commands.
     1 : stop CHANNEL(BXLAR17_MYQMGR)
AMQ8Ø19: Stop MQSeries channel accepted.
     2 : stop CHANNEL(CREWLINK.SVRCONN)
```

```
AMQ8Ø19: Stop MQSeries channel accepted.
     3 : stop CHANNEL(MQS2_MYQMGR)
AMQ8Ø19: Stop MQSeries channel accepted.
     4 : stop CHANNEL(SMPD9_MYQMGR)
AMQ8Ø19: Stop MQSeries channel accepted.
     5 : stop CHANNEL(MYQMGR_BXLAR17)
AMQ8Ø19: Stop MQSeries channel accepted.
     6 : stop CHANNEL(MYQMGR_MQS2)
AMQ8Ø19: Stop MQSeries channel accepted.
     7 : stop CHANNEL(MYQMGR_SMPD9)
AMQ8Ø19: Stop MQSeries channel accepted.
     8 : stop CHANNEL(MYQMGR_SPLHRLØ6)
AMQ9533: Channel 'MYQMGR_SPLHRLØ6' is not currently active.
     9 : stop CHANNEL(SUNSMPD2_MYQMGR)
AMQ8Ø19: Stop MQSeries channel accepted.
    1Ø : stop CHANNEL(TO.MYQMGR)
AMQ8Ø19: Stop MQSeries channel accepted.
1Ø MQSC commands read.
No commands have a syntax error.
1 valid MQSC commands could not be processed.
Channels stopped with errors
Ø783889, 5765-B75 (C) Copyright IBM Corp. 1994, 2ØØØ.  ALL RIGHTS
RESERVED.
Starting MQSeries Commands.
     1 : stop CHANNEL(BXLAR17_MYQMGR) mode(force)
AMQ9533: Channel 'BXLAR17_MYQMGR' is not currently active.
     2 : stop CHANNEL(CREWLINK.SVRCONN) mode(force)
AMQ9533: Channel 'CREWLINK.SVRCONN' is not currently active.
     3 : stop CHANNEL(MQS2_MYQMGR) mode(force)
AMQ9533: Channel 'MQS2_MYQMGR' is not currently active.
     4 : stop CHANNEL(SMPD9_MYQMGR) mode(force)
AMQ9533: Channel 'SMPD9_MYQMGR' is not currently active.
     5 : stop CHANNEL(MYQMGR_BXLAR17) mode(force)
AMQ9533: Channel 'MYQMGR_BXLAR17' is not currently active.
     6 : stop CHANNEL(MYQMGR_MQS2) mode(force)
AMQ9533: Channel 'MYQMGR_MQS2' is not currently active.
     7 : stop CHANNEL(MYQMGR_SMPD9) mode(force)
AMQ9533: Channel 'MYQMGR_SMPD9' is not currently active.
     8 : stop CHANNEL(MYQMGR_SPLHRLØ6) mode(force)
AMQ9533: Channel 'MYQMGR_SPLHRLØ6' is not currently active.
     9 : stop CHANNEL(SUNSMPD2_MYQMGR) mode(force)
AMQ9533: Channel 'SUNSMPD2_MYQMGR' is not currently active.
    1Ø : stop CHANNEL(TO.MYQMGR) mode(force)
AMQ9533: Channel 'TO.MYQMGR' is not currently active.
1Ø MQSC commands read.
No commands have a syntax error.
1Ø valid MQSC commands could not be processed.
Channels force stopped with errors
Media image for object MYQMGR, type catalogue recorded.
Media image for object MYQMGR, type qmgr recorded.
Media image for object SYSTEM.DEFAULT.PROCESS, type process recorded.
Media image for object SYSTEM.DEFAULT.NAMELIST, type namelist recorded.
Media image for object BXLAR17, type queue recorded.
Media image for object CREWLINK.IMS.REPLY, type queue recorded.
Media image for object CREWLINK.MAGELLAN.QUEUE, type queue recorded.
Media image for object CREWLINK.TO.IMS, type queue recorded.
Media image for object FROM_MQS2, type queue recorded.
Media image for object FROM_SMPD9, type queue recorded.
Media image for object IMSATS.REPLY.QUEUE, type queue recorded.
Media image for object IMSDEV.REPLY.QUEUE, type queue recorded.
Media image for object JS.IMS.IN, type queue recorded.
Media image for object JS.IMS.OUT, type queue recorded.
Media image for object JS.IMSATS.IN, type queue recorded.
Media image for object JS.TESTIMSIN, type queue recorded.
Media image for object MAGELLAN.CREWLINK.QUEUE, type queue recorded.
Media image for object MQS2.XMITQ, type queue recorded.
Media image for object OW.CLUSTER.TEST, type queue recorded.
Media image for object OW.TEST, type queue recorded.
Media image for object SMPD9, type queue recorded.
Media image for object MYQMGR.DEADLQ, type queue recorded.
Media image for object MYQMGR.TO.IMSATS, type queue recorded.
Media image for object MYQMGR.TO.IMSDEV, type queue recorded.
Media image for object SPLHRLØ6, type queue recorded.
Media image for object SUNSMPD2, type queue recorded.
Media image for object SYSTEM.ADMIN.CHANNEL.EVENT, type queue recorded.
Media image for object SYSTEM.ADMIN.COMMAND.QUEUE, type queue recorded.
Media image for object SYSTEM.ADMIN.PERFM.EVENT, type queue recorded.
Media image for object SYSTEM.ADMIN.QMGR.EVENT, type queue recorded.
Media image for object SYSTEM.AUTH.DATA.QUEUE, type queue recorded.
Media image for object SYSTEM.CHANNEL.INITQ, type queue recorded.
Media image for object SYSTEM.CHANNEL.SYNCQ, type queue recorded.
Media image for object SYSTEM.CICS.INITIATION.QUEUE, type queue
recorded.
Media image for object SYSTEM.CLUSTER.COMMAND.QUEUE, type queue
recorded.
Media image for object SYSTEM.CLUSTER.REPOSITORY.QUEUE, type queue
recorded.
Media image for object SYSTEM.CLUSTER.TRANSMIT.QUEUE, type queue
recorded.
Media image for object SYSTEM.DEAD.LETTER.QUEUE, type queue recorded.
Media image for object SYSTEM.DEFAULT.ALIAS.QUEUE, type queue recorded.
Media image for object SYSTEM.DEFAULT.INITIATION.QUEUE, type queue
recorded.
Media image for object SYSTEM.DEFAULT.LOCAL.QUEUE, type queue recorded.
Media image for object SYSTEM.DEFAULT.MODEL.QUEUE, type queue recorded.
Media image for object SYSTEM.DEFAULT.REMOTE.QUEUE, type queue
recorded.
Media image for object SYSTEM.MQSC.REPLY.QUEUE, type queue recorded.
AMQ7467: The oldest log file required to start queue manager MYQMGR is
SØØØØØØ8.LOG
AMQ7468: The oldest log file required to perform media recovery of
queue manager is SØØØØØØ8.LOG
Media image recorded OK
```

```
        stop.qmgr started at Fri Jul 19 Ø7:55:24 BST 2ØØ2
Fri Jul 19 Ø7:55:24 BST 2ØØ2 stop.qmgr Performing immediate shutdown on
MYQMGR.BW
 ....  waiting for up to 3Ø seconds ...
MQSeries queue manager ending.
MQSeries queue manager ended.
Fri Jul 19 Ø7:55:34 BST 2ØØ2 stop.qmgr Queue Manager MYQMGR already
stopped
        stop.qmgr finished at Fri Jul 19 Ø7:55:35 BST 2ØØ2
#   The two filesystems were backed up here      #
MQSeries queue manager 'MYQMGR' started.
Ø783889, 5765-B75 (C) Copyright IBM 1994, 2ØØØ.  ALL RIGHTS RESERVED.
Starting MQSeries Commands.
     1 : start CHANNEL(BXLAR17_MYQMGR)
AMQ8Ø18: Start MQSeries channel accepted.
     2 : start CHANNEL(CREWLINK.SVRCONN)
AMQ8Ø18: Start MQSeries channel accepted.
     3 : start CHANNEL(MQS2_MYQMGR)
AMQ8Ø18: Start MQSeries channel accepted.
     4 : start CHANNEL(SMPD9_MYQMGR)
AMQ8Ø18: Start MQSeries channel accepted.
     5 : start CHANNEL(MYQMGR_BXLAR17)
AMQ8Ø18: Start MQSeries channel accepted.
     6 : start CHANNEL(MYQMGR_MQS2)
AMQ8Ø18: Start MQSeries channel accepted.
     7 : start CHANNEL(MYQMGR_SMPD9)
AMQ8Ø18: Start MQSeries channel accepted.
     8 : start CHANNEL(MYQMGR_SPLHRLØ6)
AMQ8Ø18: Start MQSeries channel accepted.
     9 : start CHANNEL(SUNSMPD2_MYQMGR)
AMQ8Ø18: Start MQSeries channel accepted.
    1Ø : start CHANNEL(TO.MYQMGR)
AMQ8Ø18: Start MQSeries channel accepted.
1Ø MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.
Channels started OK
cleanmqlogs: MYQMGR: Needed for Media Recovery: SØØØØØØ8.LOG
cleanmqlogs: MYQMGR: Needed for Qmgr Restart:   SØØØØØØ9.LOG
About to compress MYQMGR's SØØØØØØØ.LOG
About to compress MYQMGR's SØØØØØØ1.LOG
About to compress MYQMGR's SØØØØØØ2.LOG
About to compress MYQMGR's SØØØØØØ3.LOG
About to compress MYQMGR's SØØØØØØ4.LOG
About to compress MYQMGR's SØØØØØØ5.LOG
About to compress MYQMGR's SØØØØØØ6.LOG
About to compress MYQMGR's SØØØØØØ7.LOG
```

The four scripts that make up this utility are show below:

```
#!/bin/ksh
# Script: mqbackup
# Purpose: This is the controlling script to backup up a Queue
```

```
Manager's   #
#          file systems
#          It take the name of the Queue Manager as its only parameter
if  [[ $# -ne 1 ]]
then
     print "Usage is: mqbackup qmgrname"
     exit 1
fi
pre.backup $1
# In here you should place the commands that will backup the Queue
Manager's #
# two file systems:
# These are usually /var/mqm and /var/mqm/log
print " "
print " "
print "##################################################"
print "##################################################"
print "##################################################"
print "##################################################"
print " "
print " "
post.backup $1
#!/bin/ksh
# Script: pre.backup
# Purpose: This script does a number of things before the queue manager
#          is run:
# 1  creates the file with the commands to stop all the channels
# 2  creates the file with the commands to start all the channels
# 3  stops all the channels
# 4  runs 'record media image' for the queuemanager ready for backup
# 5  stops the queue manager   if  [[ $# -ne 1 ]]
then
     print "Usage is: pre.backup qmgrname"
     exit 1
fi
integer big_rc=Ø
# Create a file with the command in
echo 'display channel(*)' > display.chan
# Create a file to stop the channels nicely
/usr/bin/runmqsc $1 < display.chan | grep CHLTYPE | grep -v SYSTEM |
awk '{print "stop "$1}' > stop.channel
# Create a file to stop the channels for sure using FORCE
/usr/bin/runmqsc $1 < display.chan | grep CHLTYPE | grep -v SYSTEM |
awk '{print "stop "$1" mode(force)"}' > force.channel
# Create a file to start the channels
#       We need to do this now while the queue manager is still up!
/usr/bin/runmqsc $1 < display.chan | grep CHLTYPE | grep -v SYSTEM |
awk '{print "start "$1}' > start.channel
# Stop the channels nicely
/usr/bin/runmqsc $1 < stop.channel
case $? in
```

```
Ø)      print 'Channels stopped OK' ;;
1Ø)     print 'Channels stopped with errors' ;;
2Ø)     print 'runmqsc for channel stop failed' ;;
*)      print 'Error'
esac
# Give them a chance
sleep 1Ø
# Stop the channels, no question
/usr/bin/runmqsc $1 < force.channel
case $? in
Ø)      print 'Channels force stopped OK' ;;
1Ø)     print 'Channels force stopped with errors' ;;
2Ø)     print 'runmqsc for channel force stop failed'; big_rc=big_rc+1Ø
;;
*)      print 'Error'
esac
# Give them a chance
sleep 1Ø
# Record the media image of the queues and their contents
# ready for the backup
/usr/bin/rcdmqimg -l -m $1 -t all \*
case $? in
Ø)      print 'Media image recorded OK' ;;
132)    print 'Damaged object found' ;;
135)    ;;
*)      print 'Error recording media image'; big_rc=big_rc+1Ø ;;
esac
# Stop the queue manager
#
stop.qmgr $1
exit $big_rc
#!/bin/ksh
# Script: stop.qmgr
# Purpose: This script will shutdown the Queue Manager ensuring all
#          processes have terminated before the backups can be run...
#          It takes just one parameter - the name of the Queue Manager
# Check if we are in debug mode
[[ ${VERBOSE_LOGGING} = "true" ]] && set -x
# explicitly set our path to pick up our commands
PATH="/usr/bin:/bin:/usr/sbin:/etc:/usr/local/bin"
export PATH
PROGNAME="${Ø##*/}"   # The name of this program
# We expect a single parameter of a queue manager nam
if [ $# != 1 ]
then
   echo "\nusage: ${PROGNAME} <queue manager>\n"
   exit 255
fi
echo echo "\t${PROGNAME} started at $(date)"
echo # Set any common variables here
# As per MQSeries System Management guide:
```

```
PROGRAM_KILL_ORDER="amqhasmx amqharmx amqzllpØ amqzlaaØ amqzfuma
amqzxmaØ amqrrmfa"
QMGRS=$1
# Function to check is a Queue Manager is still running #
check_qmgr()
{
if [ `ps -ef | egrep -v
"grep|amqcrsta|runmqsc|stop.qmgr|mqbackup|pre.backup" | grep -c $QMGR`
-ne Ø ]
then return 1  # Still running
else return Ø  # Stopped
fi
}
# Function to stop Queue Manager #
stop_qmgr()
{
# immediate shutdown
check_qmgr
if [ $? -ne Ø ] ; then
  echo `date` $PROGNAME Performing immediate shutdown on $QMGR
  endmqm -i $QMGR &
  echo " ....  waiting for up to 3Ø seconds ..."
  for i in 1 2 3
  do
    sleep 1Ø
    check_qmgr
    if [ $? -eq Ø ] ; then
      break
    fi
  done
else
  echo `date` $PROGNAME Queue Manager $QMGR already stopped
  return
fi
# pre-emptive shutdown
check_qmgr
if [ $? -ne Ø ] ; then
  echo `date` $PROGNAME Performing pre-emptive shutdown on $QMGR
  endmqm -p $QMGR &
  echo " ....  waiting for up to 6Ø seconds ..."
  for i in 1 2 3 4 5 6
  do
    sleep 1Ø
    check_qmgr
    if [ $? -eq Ø ] ; then
      break
    fi
  done
else
  echo `date` $PROGNAME Queue Manager $QMGR already stopped
  return
fi
# Kill off any remaining tasks
check_qmgr
if [ $? -ne Ø ] ; then
  for PROGRAM in $PROGRAM_KILL_ORDER
  do
    PID=`ps -ef|grep $QMGR|grep $PROGRAM|awk -F" " '{print $2}'`
    if [ "."$PID != "." ] ; then
      echo `date` $PROGNAME "Killing (-9) PID $PID $PROGRAM"
      kill -9 $PID
      echo " ....  waiting for 1Ø seconds ..."
      sleep 1Ø
    fi
  done
fi
}
# Function to clean up Processes
function cleanup_processes
{
for PROC in DCSQMain runmqsc amqcrsta
do
    ps -ef|grep -i $PROC|egrep -v grep|awk '{print $2}'|while read PID
    do
      if [[ $PROC = runmqsc ]]
      then
        ps -fp $PID|grep -q $QMGR && kill -9 $PID
      else
        kill -9 $PID
      fi
    done
done
}  # end of cleanup_processes
# ascertain the OS and set any specific variables here
OS=$(uname -s)
case $OS in
  "AIX")
   # Set any IBM AIX specific parameters
   ;;
  "SunOS")
   # Set any SUN Solaris specific parameters
   ;;
  "*")
   # Not a recognised OS
   echo "ERROR: the operating system is not known"
   exit 255
   ;;
esac
# Main code #
for QMGR in $QMGRS
do
          stop_qmgr
      cleanup_processes
done
echo echo "\t${PROGNAME} finished at $(date)"
#!/bin/ksh
# Script: post.backup
# Purpose: This script does a number of things after the queue manager
#          back-up is run:
# 1  restarts the queue manager
# 2  starts all the channels
# 3  runs cleanmqlogs to compress old log files
# 4  deletes compressed log files over 7 days old
# 5  deletes temporary files
if  [[ $# -ne 1 ]]
then
     print "Usage is: post.backup qmgrname"
     exit 1
fi
# Start the queue manager
/usr/bin/strmqm $1
# Start the channels
/usr/bin/runmqsc $1 < start.channel
case $? in
Ø)      print 'Channels started OK' ;;
1Ø)     print 'Channels started with errors' ;;
2Ø)     print 'Channels start failed ' ;;
*)      print 'Error' ;;
esac
# Run IBM support pack MS62 (cleanmqlogs) to compress unneeded log
files cleanmqlogs -Z $1
# Get rid of compressed logs over 7 days old
cd /var/mqm
find . -mtime +7 -name '*.LOG.Z' -exec rm -f {}  \;
# Get rid of the temporary files
rm display.chan
rm stop.channel
rm force.channel
rm start.channel
```

Chris Bell, Systems Consultant
British Airways (UK)

WebSphere MQ V5.3 clustering: hints and tips

INTRODUCTION

A number of changes to the MQSC commands have been made for the

WebSphere MQ (WSMQ) clustering support in V5.3 to make it easier to recover a cluster after a problem. This articles provides some hints and tips on how to use these commands effectively. Beginning with a look at the key points that should be adhered to when creating a cluster, we go on to focus on the channels that make up the backbone of the cluster and on which everything else relies. Continuing with an examination of some problems that have been encountered when using clustering, we shall also discuss ways of fixing them. The aim is that by combining a better understanding of how clusters work with the additional functions provided in V5.3 users should find it simpler to maintain clusters.

I'm assuming that readers have a basic understanding of what MQ clustering is. If not, I would recommend reading the first few chapters of the *WSMQ Queue Manager Clusters* manual before proceeding.

TERMINOLOGY

From here on we talk about partial repository and full repository queue managers. A full repository queue manager holds a complete view of a cluster, ie it knows about all the queue managers in the cluster, the cluster queues they host, and which other queue managers in the cluster are interested in the queues. It is recommended that there are two full repositories for a cluster in order to avoid a single point of failure.

All other queue managers within a cluster hold a partial view of the cluster. This means that they know only about the cluster queue managers and cluster queues that are required by applications on their queue managers. If a partial repository needs to find out about a queue or queue manager within the cluster, it will subscribe for the information from two full repository queue managers and then add the information to its own partial repository.

DEFINING CLUSTER CHANNELS

The key to using MQ clustering successfully is the correct definition and maintenance of the manually defined cluster channels on queue managers within the cluster. Let's start by looking at the two different types of cluster channel and what they are used for.

Manually defined cluster sender channels

A manually defined cluster sender channel has a slightly different meaning depending on whether it is defined on a partial repository queue manager or a full repository queue manager.

On a partial repository queue manager it can be thought of as a bootstrap to get a queue manager up and running in the cluster. The channel must point at a full repository for the cluster. This is the full repository that the new queue manager will use initially to subscribe for information about other full repositories in the cluster.

Once the new queue manager has the information about all the full repositories in the cluster it no longer requires the manually defined cluster sender definition because it can automatically create channels to the full repositories. However, when making a choice between which two full repositories to use when publishing or subscribing for queues and queue managers within the cluster, those full repositories that have manually defined cluster sender channels to them are picked ahead of automatically defined cluster sender channels. So if you want some control over which full repositories a partial repository uses you should define manual cluster sender channels to them.

(Note that this does not guarantee that subscriptions will only be made on those full repositories to which you have defined the manual cluster senders. The queue manager makes its choice of repository based on a number of different criteria, such as the channel state, ie a running channel is better than a retrying one. If the other criteria are all equal then manually defined channels take precedence.)

On a full repository the manually defined cluster sender channels are used to tell the queue manager which other full repositories for the cluster it should send information to. This means that rather than have every full repository talking to every other full repository for the cluster you can decide on the best route for propagation of the information. It is vital that the manually defined cluster sender channels on the full repositories form a fully connected set, otherwise some of the full repositories will not receive all of the updates about the cluster.

In the example shown in Figure 1 the arrows represent manually defined cluster sender channels. P1 is a partial repository and F1, F2, and F3 are full repositories.

If P1 needs to subscribe for a queue it will preferentially choose F1 for one of the subscriptions as it has a manually defined cluster sender channel to F1. It will also choose between F2 and F3 to make its second subscription.

If F1 received information about a new queue in the cluster it would publish that information to F2; F2 would publish it to F3 and F3 would publish it back to F1, which would notice that it had already seen the information and discard the second copy. Note that this example is only meant to show how the manually defined cluster sender channels control the flow of information between the full repositories. In practice it would be useful to have some redundant connections as in this example a failure of F2 would mean that when F1 receives new information it will not reach F3.

Cluster receiver channels

By defining a cluster receiver channel we are stating how other queue managers in the cluster can talk to our queue manager. The cluster receiver channel also contains information such as a CONNAME. This is because the information provided in the cluster receiver channel definition is used by other queue managers within the cluster to create automatic sender channels to our queue manager. When the cluster receiver is defined the information within it is sent to the full repositories for the cluster and the full repositories will pass the information on to any other queue managers within the cluster that want to send data to our queue manager.

We can see that defining a cluster receiver channel has much further-reaching effects than most MQSC commands, which only affect the local queue manager. For this reason it is worth being especially careful when

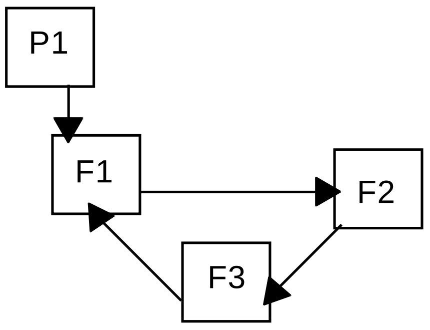

Figure 1: Manually defined cluster sender channels

defining or modifying cluster receiver channel definitions. I would recommend avoiding, for example, cutting and pasting of cluster receiver definitions from one queue manager in the cluster to another because if you forget to change some of the attributes this can adversely affect the cluster.

Example problem

Let's take a look at what could go wrong when a cluster receiver is incorrectly defined and how to fix the problem. For our example let's say that when defining/altering the cluster receiver channel definition the wrong CONNAME is used.

- What happens?

  The new channel information is sent to the full repositories and they will pass it on to other queue managers that need to send data to our queue manager. The new information takes effect the next time an attempt is made to start the automatic sender channels to our queue manager. At this time they will go into retry state because they will be unable to connect. Note that if the channels are already running they will continue to run with the old information until the next time they restart so you may not notice the problem immediately.

- How do you fix the problem?

  Alter the cluster receiver channel to contain the correct CONNAME. Once again, the information will be sent to the full repositories and they will forward it on to interested queue managers. If the channels are in retry the next time they do retry they will pick up the new CONNAME and use that. Alternatively the channels could be started manually if the retry interval is quite long.

Channel definition summary

A lot of problems that we see in MQ clustering arise because the channels have not been defined correctly. It is worth taking time to understand how the two different types of channel are used. Let's now look at what happens when a partial repository queue manager is added to the cluster and how to determine whether or not it has been added successfully.

ADDING A PARTIAL REPOSITORY QUEUE MANAGER TO THE CLUSTER

The initial communication between a partial repository and a full repository takes place when both a cluster sender channel and a cluster receiver channel are defined on the partial repository for the same cluster. The partial repository takes the information from its cluster receiver channel and sends it to the full repository down the cluster sender channel. The full repository then stores the information as an automatically defined cluster sender channel, which it then immediately uses to send the information from its cluster receiver channel back to the partial repository.

If you want to check that this has taken place the easiest way is to issue the command **DISPLAY CLUSQMGR(*) ALL** on the partial repository queue manager and look at the cluster sender definitions as explained below.

When you issue a **DISPLAY CLUSQMGR** command on a cluster queue manager and look at the DEFTYPE attribute of the objects displayed there are three different types of cluster sender channel that you may see:

- CLUSSDR. This is a manually defined cluster sender channel with which we have not yet successfully completed our initial communications with the cluster queue manager at the receiving end of the channel. The CLUSQMGR attribute will show the name of the queue manager as being SYSTEM.TEMPQMGR... This is because it doesn’t yet know the real name of the queue manager at the receiving end of the channel.
- CLUSSDRB. This is a manually defined cluster sender channel which has successfully completed its initial communications with the cluster queue manager at the receiving end of the channel. It will now have a real queue manager name in the CLUSQMGR attribute and the other attributes of the channel will reflect the attributes defined in the CLUSRCVR at the receiving end of the channel. Note the QMTYPE attribute should be REPOS. If it is not then the manually defined channel does not point at a full repository for the cluster.
- CLUSSDRA. This is an automatically defined cluster sender channel.

So by displaying the CLUSQMGR information on the partial repository what we should see is a cluster sender channel with

DEFTYPE(CLUSSDRB). If we see one with DEFTYPE(CLUSSDR) we know that the initial communications have not completed. Here is a list of things to check if this occurs.

1 Is a CLUSRCVR channel defined? Are the CLUSRCVR and CLUSSDR channels defined in the same cluster? The initial communications will not take place until both a cluster receiver and a cluster sender exist for the same cluster.

2 Is the CLUSSDR channel started? If the channel is stopped then start it. If it's in retry check that the listener on the full repository is started and that the CONNAME is correct in the CLUSSDR definition.

3 If you issue DISPLAY CLUSQMGR on the full repository do you see your queue manager displayed? If no cluster sender channel for your queue manager is displayed, check for any error messages issued on the full repository. Check the CURDEPTH of the SYSTEM.CLUSTER.COMMAND.QUEUE on the full repository. This is the queue where the work for the repository manager is queued. If the CURDEPTH is high but dropping, it could be that the request from your queue manager has not been processed yet.

4 Is the automatically defined sender channel from the full repository to the partial repository started? If the channel is stopped, start it. If it's in retry, check that the listener on the partial repository is started and that the CONNAME is correct in the partial repositories CLUSRCVR definition.

The remainder of this article looks at additions to the MQSC commands in WSMQ V5.3.

HANDLING DUPLICATE QUEUE MANAGERS IN A CLUSTER

When a change occurs to an object in the cluster the change always originates at the queue manager in the cluster on which the MQSC command was issued that caused the change. That queue manager will then inform two of the full repository queue managers about the change and they in turn will inform any other full repositories and also any partial repositories.

The same applies for deletes. If you want to remove a queue manager from a cluster the correct way to do this is to delete any cluster resources that the queue manager owns, ie issue **ALTER QL(<queuename>) CLUSTER(' ')** on each cluster queue that the queue manager owns. This will inform two of the full repositories that the queues are no longer a part of the cluster. Issue **ALTER CLUSRCVR(<channelname>) CLUSTER(' ')** on each cluster receiver that the queue manager owns. This will inform two of the full repositories that the automatically defined sender channels to the queue manager should be deleted. This information will then get propagated around the cluster via the full repositories so that all the queue managers are aware of the deletion.

(See the *WSMQ Queue Manager Clusters* manual for the full procedure for removing a queue manager from the cluster.)

A common problem that has been encountered in clustering occurs when a queue manager is deleted without first being removed from the cluster. This makes other queue managers in the cluster think it still exists and the queue manager can no longer inform the cluster that it doesn’t. This can be solved by issuing the reset cluster command on a full repository in the cluster to remove the deleted queue manager’s definitions.

```
RESET CLUSTER(<clusname>) ACTION(FORCEREMOVE) QMNAME(<qmgrname>)
QUEUES(YES)
```

Note that the reset cluster command is available in all releases of MQ clustering; however, the QUEUES keyword is new in 5.3 and allows any cluster queues associated with the deleted queue manager to be removed also.

The problem shown above is often made more complicated by deleting the queue manager and then recreating it from a script, which has the effect of adding it back into the cluster with the same queue manager name and the same channel definitions. Because the queue manager was recreated it will have a new unique identifier associated with it, called the QMID, and other queue managers in the cluster will now see two instances of the channels, one instance for the old QMID and one for the new QMID.

We cannot use the reset cluster command shown above as the queue manager name is no longer unique so we must use a slightly different version of the command.

```
RESET CLUSTER(<clusname>) ACTION(FORCEREMOVE) QMID(<oldqmid>)
QUEUES(YES)
```

Note that the QMID keyword is new for 5.3.

This will force the full repository to delete any cluster queue and cluster queue manager information for the QMID specified and propagate the deletes around the cluster.

REFRESHING A QUEUE MANAGER'S VIEW OF THE CLUSTER

There may be times when a queue manager's view of the cluster becomes out of sync with the rest of the cluster. There is a command called **REFRESH CLUSTER,** which causes the queue manager on which it is run to discard the information it holds in its local repository and then communicate with the full repositories again to get back in sync.

The version of this command that exists in the pre-5.3 releases will never completely remove all of the information from the local repository as it tries to leave enough information around to enable the queue manager to join the cluster successfully again. For example, on a partial repository the information regarding full repository queue managers is not removed so that as part of the refresh the queue manager can talk to the full repositories and request new information.

Situations may arise, however, when it is useful to be able to say "I want to get my queue manager to start again in the cluster from the initial manually defined channels", so in V5.3 this has been added as an option.

If you specify **REFRESH CLUSTER(<clusname>) REPOS(YES)** all of the information about the cluster in the queue manager's local repository is removed. In order to achieve this the sender channels for the cluster should be stopped as the queue manager cannot remove information about them if they are in use. If the sender channels are not stopped when the command is run it will force them to stop so that the refresh can be completed. There are several points to consider when using this command.

First, the REPOS(YES) option cannot be run on a full repository. This is because it will delete all the channels and, therefore, cannot ensure that other queue managers in the cluster know it is being refreshed and that they should take some action. Therefore you must first alter the full repository to be a partial repository. This will make the full repository inform the other queue managers in the cluster that it is not a full repository any more and they can then request information from another full repository if there is one.

Having made the queue manager a partial repository it’s worth waiting for the change to be propagated around the cluster before issuing the refresh command, ie if there are a lot of channels that the full repository needs to start to inform the partial repositories that it is no longer a full repository, wait for them to start first. Once the partial repositories have been informed about the change the channels can then be stopped and the refresh command issued.

Secondly, if you have only one full repository for the cluster (which is not recommended), then after the refresh has occurred and the queue manager is altered to be a full repository again it cannot tell all the other queue managers that it is a full repository because it’s just deleted all of the information about them. Therefore, it will be necessary to refresh all the other queue managers in the cluster to get them communicating correctly with the full repository again. If you have two full repositories for the cluster, temporarily making one of them a partial repository is fine because once the refresh is complete and the queue manager is altered to be a full repository again it will tell the other full repository, which will inform all the partial repositories.

Thirdly, if your queue manager has channels that are in multiple clusters, refreshing a single cluster will not completely remove all of the local repository information because some of it is relevant to other clusters as well. Another option that has been added to the **REFRESH** command is the ability to say:

```
REFRESH CLUSTER(*)
```

This will cause all the clusters to be refreshed on the local queue manager at the same time.

The most important point to remember about the refresh cluster command is that all it is doing is clearing the local repository state, which will then get added again later when it is required. In a working cluster it should not be necessary to run this command. If a queue manager appears to be out of sync with the full repositories it would be more useful to check that the channels are running and the definitions are correct, rather than hope that a refresh will fix the problem.

TO CONCLUDE...

Clustering can be a very powerful tool, providing an easy way to connect and share resources between a large number of queue managers. When we look at implementing a cluster it is important to get right the initial manually defined channels that provide the information the queue managers require to cluster themselves. Therefore it's worth having a good understanding of what functions these channels perform and the effects they have on the cluster.

The changes to the MQSC commands in WSMQ V5.3 focus on the two main problems seen by users of clustering. First, being able to remove a duplicate queue manager from the cluster when that actual queue manager no longer exists and, second, to be able to cold start a queue manager in the cluster from its original manually defined channels. For more information on clustering I would strongly recommend reading the *WSMQ Queue Manager Clusters* manual, which contains numerous examples of setting up clusters.

My thanks go to Mike Horan and Andrew Banks of IBM for their contributions to this article.

Dan Millwood, WSMQ Development
IBM Hursley (UK)

Stand-alone MQSeries Workflow client/server setup on Windows NT

The aim of this article is to assist developers who would like to set up MQSeries workflow on Windows NT in stand-alone mode. It is based on existing and informative guidelines by IBM but I have attempted to make it more practical so that programmers jumping into MQSeries with little or no background knowledge will benefit most.

TO BEGIN

Check that Windows NT Workstation Version 4.0 at Service Pack 6a is installed on your workstation. The version and service level installed on your system are displayed on the screen at boot-up. See the Microsoft Network or the Microsoft Web site *http://support.microsoft.com/support/ntserver/content/servicepacks/* for Windows NT service packs.

- Click on Service Pack 6a, then on Download, then on High Encryption Version.
- Click on Start SP6a Express Download (for single computer installation only).
- Save the program *sp6patch_i386.exe* to disk.
- Open Windows Explorer, locate the downloaded file, and double-click on it. This will prompt you with a licence agreement, which you should accept. Click on Install to install the Service Pack. Once the installation is successful you will be prompted to restart the machine to make the software updates effective.

The next step is to create a new Windows user-ID with administration rights. This user-ID is used later during the MQ Workflow configuration. Note that this user-ID must conform to the naming rules for DB2. The password must be no longer than eight characters and must not contain accent characters.

- On the task bar click on the Windows Start menu and select Programs->Administrative Tools (Common)->User Manager. The User Manager window appears.
- Within the User Manager window from the menu bar select User->New User.... A New User dialog box appears.
- Within the New User dialog box:
  - enter a user name that is a maximum of eight characters long within the Username field. Make a note of your new user-ID, ie ChandraU.
  - enter a password within the password field and confirm the password within the Confirm Password field. Make a note of the password ‘Password’ and keep it in a safe place.
  - Deselect the User Must Change Password at Next Logon check box.

  - Click on the Groups button. A Group Memberships dialog box appears.
  - Within the Group Memberships dialog box make your new user-ID a member of the administrators' group by selecting Administrators and clicking on <-Add. Select OK. Control passes back to the New User dialog box.

- Select OK and close the User Manager window.

Log off from Windows and log on again specifying the new user-ID and password.

INSTALLING PREREQUISITE SOFTWARE

DB2 and MQSeries are prerequisites for MQ Workflow and must be installed before running the MQ Workflow installation on your workstation. The following steps are the minimum needed to install DB2 and MQSeries for an MQ Workflow stand-alone system.

Installing DB2 on Windows

The following instructions describe how to install DB2 Universal Database Version 7.1 on Windows NT for an MQ Workflow stand-alone system.

- You must be logged on with a user-ID that has administrative rights.
- Insert the CD-ROM labelled *DB2 Universal Database Enterprise Edition Version 7.1 for Windows Operating Environments* into your CD-ROM drive.
- If the installation does not start automatically, start it by clicking on the Start menu on the task bar, selecting Run..., and entering **x:\Setup.exe** in the Open field and click on OK, where *x* is the drive letter for your CD-ROM drive.
- The installation window with the welcome text appears. Click on Install to start the installation procedure. The Select Products window is displayed.
- Select DB2 Enterprise Edition and then click on Next. The Select Installation Type window is displayed.

- Select Custom and then click on Next. The Select Components window is displayed.
- Select the following components:
  - Application Development Interfaces
  - Documentation
  - Base DB2 UDB support
  - Administration and Configuration Tools.
- Click on Next. If you get a message about installing Microsoft MDAC click on OK. MQ Workflow will install this if it is required.
- When you see the Create DB2 Instance window click on Next.
- When you see the Configure DB2 Services window click on Next.
- Enter the user-ID and password that you created earlier and click on Next.
- When you see the Start Copying Files window click on Next. The DB2 program files will now be copied to your workstation, DB2 will be configured, and the DB2 services will be started.
- When you see the Setup Complete window, click on Finish.

Installing MQSeries on Windows

Now you need to install MQSeries for Windows NT and Windows 2000 V5.2 for an MQ Workflow stand-alone system.

- Insert the CD-ROM labelled *MQSeries for Windows NT and Windows 2000, V5.2* into your CD-ROM drive.
- If the installation does not start automatically, start it by clicking on the Start menu on the task bar, selecting Run..., and entering **x:\Setup.exe** in the Open field, where *x* is the drive letter for your CD-ROM drive, and click on OK.
- The Select Setup Language window appears. Select the language English and click on OK. The Setup dialog appears as MQSeries prepares the install shield, after which the Welcome window is displayed.
- Click on Next. The Read Licence Conditions window is displayed. Click on Yes to accept the terms of the licence agreement.

Certain prerequisite software must exist on your workstation for MQSeries to install correctly. If anything is missing, the Software Requirements window appears displaying a list of items you will need to install. If you have everything you need, the Choose Installation Folders window is displayed.

- To accept the default MQSeries installation folders, click on Next. If you do not want to use the defaults given, change them and then click on Next. The Setup Type window is displayed.
- Select the Typical radio button and then click on Next.
- The Select Program Folder window is displayed.
- Click on Next. This causes a folder called IBM MQSeries to be added to the Windows Start menu under Programs. The Ready to Copy Files window is displayed.
- Click on Next. Program files are copied to the MQSeries installation directory. This may take some time, after which the Setup Complete window is displayed.
- MQSeries is now installed and is set to start automatically as a Windows service. If necessary (depending on your current setup) your workstation will be rebooted.
- The Setup Complete window allows you to launch the following:
  - MQSeries Default Configuration Wizard
  - MQSeries Information Centre
  - MQSeries First Steps
  - MQSeries Explorer
  - Notepad to view the release notes.
- Uncheck all check boxes.
- Click on Finish. You have completed installing MQSeries. It has been set to start automatically as a Windows service.

Installing MQ Workflow on Windows

Before installing the MQ Workflow stand-alone software several services that may be running on your system must be stopped.

- If IBM Antivirus is installed on your workstation, stop the service AvService.
- If Norton Antivirus is installed on your workstation, stop the service NAV Auto-protect.
- If Microsoft Outlook is installed on your workstation, close the application and log off.

After stopping these services follow the instructions given next, which describe how to install a stand-alone MQ Workflow system on a single Windows NT workstation.

- Insert the CD-ROM labelled *IBM MQSeries Workflow V.3 – Program Code for Windows Platforms* into the CD-ROM drive.
- If the installation program does not start automatically, start it by opening a command prompt window and entering: **x:\WINDOWS\SETUP,** where *x* is the drive letter for the CD-ROM drive.
- Select the language that you want to use. This becomes the default for your MQ Workflow stand-alone installation.
- The Software License Agreement window is displayed. You may either accept or decline the licence. Note that the language of the Software License Agreement dialog depends on the Regional Settings specified in the Windows Control Panel, not on the language you selected in the previous dialog.
- Click on Next. The Welcome window is displayed.
- Click on Next. The Choose Destination Location window is displayed with the default directory *C:\Program Files\MQSeries Workflow* set as the installation directory for MQ Workflow. If you do not want to use this as your installation directory enter a new location.
- Click on Next. The Setup Type window is displayed. Select All Components and click on Next.
- The Select Components window is displayed. If not all the components are selected, click Select All.
- Click on Next. The Select Program Folder window is displayed.

- Click on Next. A folder for MQ Workflow is created and appears on the Windows Start menu under Programs. The Start Copying Files window is displayed.
- Confirm your selection by clicking on Next. Program files are copied to the MQ Workflow installation directory.
- When the installation phase is complete remove the program code disk from the CD-ROM drive.
- Click Finish to restart your workstation and activate the changes made by the installation program. After rebooting, the MQ Workflow configuration utility should start automatically.

CONFIGURING MQ WORKFLOW

Configuration must be performed directly after installing MQ Workflow so that database and communication resources provided by the prerequisite software, DB2 and MQSeries, can be used. This is done using the MQ Workflow configuration utility, which starts automatically after the MQ Workflow installation stage. If necessary you can start it manually by selecting the MQSeries Workflow Configuration Utility icon found in the IBM MQSeries Workflow folder.

It is recommended that you use the default values provided during the configuration stage for a test and first-time MQ Workflow stand-alone installation.

There follow the steps required to configure the Workflow System. The screen shots can be found at www.xephon.com/extras/Appendix_A.doc.

- On the General tab (Figure 1):
  - Click New. You will be prompted with default configuration FMC as the configuration-ID (Figure 2). Click OK to confirm.
  - Check all components you want to configure. It is common to select all components.
- On the Runtime Database tab (Figures 3, 4, and 5):
  - Select DB2 as the DB2 instance. You will also see other instances, but ignore them.

- Click New to the right of the second box to create a new database in the selected DB2 instance.
- In the pop-up window that appears accept all the default values by clicking OK.
- When the pop-up closes click on the button DB2 CONNECT PARAMETERS, which will prompt you to enter a user-ID and password. Enter the user-ID that has workflow admin-ID – in this case it is 'ChandraU' and the password is 'Password'.
- Click Next.

- On the Queue Manager tab (Figure 6):
  - TCP/IP port configuration should be selected under the communication protocol and the adjacent field should contain either your computer's host name or, if you're not connected to network, localhost. Then click Next.
- On the Cluster tab click Next to accept all defaults (Figure 7).
- On the Client Connections tab, click Next (Figure 8).
- On the Buildtime tab (Figure 9) click Next. By default, IBM DB2 Universal Database is selected.
- On the Buildtime Database tab (Figures 10 and 11), follow the same procedure as for the Runtime Database tab above.
- On the Client tab (Figure 12), click Next.
- On the Java CORBA Agent tab (Figure 13), click Done.
- The configuration takes some time (15 to 20 minutes), during which you will be shown its progress (Figure 14). On completion you will be prompted with an acknowledgement that the configuration has been successful (Figure 15). Click OK and Reboot the system.

CHECKING THE CONFIGURATION WITH FMCZCHK.

The MQ Workflow configuration-checking utility can be used to check the configuration of an MQ Workflow server, client, or buildtime running on any of the supported platforms in a standard MQ Workflow client/server network or stand-alone system. To help you configure MQ Workflow the configuration-checking utility can find and correct installation errors and inconsistencies, eg you can check that:

- Environment variables are set correctly.
- Network drivers are installed properly.
- Network configuration files have been updated.
- The MQ Workflow profile contains consistent settings.

Start the utility immediately after each change to the MQ Workflow configuration.

Starting the configuration checking utility

The configuration-checking utility is started and used in the same way for all MQ Workflow components. It is a command-line utility in US English only and is designed to be platform-independent. No additional installation or configuration steps are required. It is a self-contained tool that is copied to the MQ Workflow BIN directory during installation. To start the utility type **fmczchk** at a command prompt.

Configuration checking is done in several phases. During each phase one particular MQ Workflow component is checked. Even if the checks for a component do not complete successfully the configuration-checking utility proceeds with the checks for all other components. Problems are displayed as soon as they are detected.

In addition, the configuration-checking utility creates a log file called *fmczchk.log,* which contains all error or warning messages and other important information.

Using command-line options

Although you can start the utility without any options, several are available that can be specified directly after the **fmczchk** command.

The command-line options explained here are valid for MQ Workflow V3.2 and later versions only. For earlier versions see the relevant *MQ Workflow Installation Guide* or refer to the online documentation. Not all options are described here; for a full description of the **fmczchk** command see the *MQSeries Workflow Administration Guide*.

Command-line options start with a slash (/) or minus (-) and can be followed by an argument. Arguments to options can be separated from the option letter by an empty string (....), a blank (' '), a colon (:), or an equals sign (=). The options are not case-sensitive. Use the options listed below.

- **-330** – specifies that you want to perform checks on MQ Workflow Version 3.3.0. The default is the version with which the utility was created.

  Which checks are available depends on the MQ Workflow version that you specify. When you start the utility only those checks which apply to the defined version are displayed.

- **-b** – selects batch mode. Messages are not written to the console.

- **-d** – show debug messages. Debug messages are needed by support personnel to help analyse problems. Since these messages are intended for support personnel only they are not documented here.

- **-e** – show error messages only. The default is to show error and warning messages and to suppress information messages.

- **-i** show – all messages, ie error, warning, and information messages.

- **-htm,-html** – write messages to the *fmczchk0.htm* file instead of the *fmczchk.log* file. The *fmczchk0.htm* file provides links to online documentation that contains further information describing the configuration-checking utility. Error, warning, and information messages written to the *fmczchk0.htm* file are written as links. Clicking on each message takes you to an online description for that message, which gives you information regarding the severity and user action required for the message.

  Note that the hyperlinks in the generated *fmczchk0.htm* file only work in the *InstallationDirectory/bin* directory, because that is where the message explanation file (.......htm) is located.

- **-l filename** – name of the log file. If this file already exists, the messages will be appended to the file.

- **-y configurationidentifier** – allows you to specify a configuration-ID other than the default. Specifying a different configuration-ID allows you to perform checks on different systems. If this option is not used, the value of the DefaultConfigurationID variable set in the general configuration profile is used.

- **-c command [;...]** – performs a task specified by the command. The following are valid commands:
  - **sca [:filemask;...]** – specify this command to scan all MQ Workflow executables for the version string. You can restrict the scanning by specifying your own file mask as an optional argument, ie

    ```
    fmczchk -c sca:dll \fmck*.dll;bin \fmce*.exe
    ```

  - **tcp:service,port** – adds a port to the TCP/IP services file. For example, to add port definitions for MQSeries to your services file:

    ```
    fmczchk -c tcp:fmclFMCQMA5Ø1Ø,5Ø1Ø
    ```

    Port definitions are automatically added to the services file during configuration.

  - **trc:level [,filename][,split][,flipflop][,filesize]** – can be used to enable and disable tracing. The trace level can range from zero for minimum information to three for maximum information. You can optionally specify the name of the trace file. For example, to enable a full trace of system Config001, using split tracing and 5000 kilobyte flip-flop tracing files, you can enter:

    ```
    fmczchk -y ConfigØØ1 -c trc:3,/tmp/traces/my_trace,1,1,5ØØØ
    ```

  - **@cmdfile** – alternatively, you can create a file containing several commands that you wish to run. For example, if you create a file *fmczchk.cmd* with the following lines in it:

    ```
    -c tcp:fmclFMCQM5Ø1Ø,5Ø1Ø
    -c sca:dll \fmck*.dll;bin \fmce*.exe
    ```

    you can start the utility as follows:

    ```
     fmczchk @fmczchk.cmd
    ```

Note that you must prefix commands with the **-c** option. This allows you to include other command options that previously could not be used in the response file, for example: **-y FMC1**.

The configuration checking log file and online documentation

The configuration checking utility writes a log file with the name *fmczchk.log* in the current directory. This log file is intended to be used by your support personnel. The message options you specify after the **fmczchk** command determine what messages are displayed on the screen during the configuration checking routine. These options are ignored when writing to the log file, that is, all messages are recorded in the log file.

A list of all the messages that can be written in the log file is available in the file *fmczchk.htm*. By specifying the html option after **fmczchk** an html file is created instead of the log file. This html file contains links that give you access to online documentation. The online documentation gives an explanation, user response, and severity for each message. Each message is made up of a message identifier code and message text. The last character of the message identifier code denotes the message type or severity of the message.

Note that the hyperlinks in the generated *fmczchk0.htm* file only work in the *InstallationDirectory/bin* directory because that is where the message explanation file (.......htm) is located.

The following shows the format for each type of message identifier code, where *nnn* is a number used to identify each message.

- FMC34nnnI – informational message. No action is required.
- FMC34nnnW – warning message. Action may be required. Check the user response in the on-line documentation.
- FMC34nnnE – error message. Action is required. Check the user response in the on-line documentation.

The log file is not created if it cannot be opened, eg because of missing write permission in the current directory. However, the configuration utility continues to display important error and warning messages on the screen. To display all messages on the screen use option **i**.

VERIFYING THE MQ WORKFLOW CONFIGURATION

To verify that components are communicating correctly you should check your MQ Workflow installation. This is done by verifying that the MQ Workflow Server on your workstation is running and that the MQ

Workflow Client on your workstation can connect to it.

To verify that the MQ Workflow Server is running:

- Run the batch job **START_MQ**, which will generate:
  - strmqm FMCQM
  - start runmqlsr /t tcp /p 5010 /M FMCQM
  - start runmqtrn /m FMCQM /q FMCTRIGGER

  and start:
  - queue manager for configuration FMC1
  - MQSeries trigger monitor for configuration FMC1
  - MQSeries command server for configuration FMC1: this will start two DOS windows; do not close them.
- Select services.
- On Windows NT:
  - on the task bar, click on the Windows Start menu and select Settings
  - select Control Panel
  - select the Services icon. A dialog box appears.
- Within the Service window of the dialog box locate the line that reads MQSeries Workflow Version 3.3.0 - FMC.
- If the status of this service is Started the MQ Workflow Server is running.
- To verify that the client can connect to the server, start the following:
  - the Administration Server
  - the Administration Utility
  - Standard Client
  - Build Time.

Starting the administration utility

Start the Administration Server before starting the Administration Utility, the procedure for which is as follows:

- On the Windows task bar click on the Start menu. Select Programs.
- Select the IBM MQSeries Workflow program folder.

- Select the MQSeries Workflow Administration Utility – Configuration-ID, where Configuration-ID is the configuration identifier that identifies the MQ Workflow configuration for the Administration Utility. A command prompt will be opened, displaying the following:

```
-FMC16ØØ6I Administration Utility started.
System group name :[FMCGRP ]FMCGRP
System name :[FMCSYS ]FMCSYS
Userid :[ADMIN ]ADMIN
Password :[]
```

- Enter the password for the MQ Workflow user-ID ‘ADMIN’. The password is initially set to ‘password’. For more details about the administration utility refer to the *IBM MQSeries Workflow: Administration Guide*.

Starting and stopping other MQ Workflow servers

If MQ Workflow server components are not started with the Administration server you must use the MQ Workflow administration utility to start each server component individually.

Starting the standard Client

Before starting the standard MQ Workflow Client, the Administration server and all other MQ Workflow server components must already be running. To start a standard MQ Workflow Client:

- On the Windows task bar click on the Start menu and select Programs.
- Select the IBM MQSeries Workflow program folder.
- Select the MQSeries Workflow Client - <ConfigID> icon, where <ConfigID> is replaced by the configuration identifier that identifies the MQ Workflow configuration for the standard MQ Workflow Client.

In the window that appears enter the Client’s user-ID (ADMIN) and password (password) and the name of the MQ Workflow system and system group to which the Client should connect. If unified logon has been set in the runtime database you are automatically logged on to the MQ Workflow system without the need to specify the Client’s user-ID and password. Unified logon means that when users have logged on to

Windows 2000 or NT with their password there is no need to further log on to individual applications. MQ Workflow supports unified logon when it is specified in the system properties table during MQ Workflow Buildtime.

Starting Buildtime

To start Buildtime installed on any of the supported Windows-based operating platforms, perform the steps listed below.

- On the Windows task bar, click on the Start menu and select Programs.
- Select the IBM MQSeries Workflow program folder.
- Select the MQSeries Workflow Buildtime - <ConfigID> icon, where <ConfigID> is replaced by the configuration identifier that identifies the MQ Workflow configuration for MQ Workflow Buildtime.

In the window that appears enter the Buildtime's user-ID (ADMIN) and password (password). If unified logon has been set in the Buildtime database you are automatically logged on without the need to specify the Buildtime's user-ID and password. Your stand-alone system is now fully functional.

COMMON ERRORS

Here are some of the errors I have encountered while configuring workflow in stand-alone mode.

- FMC34808W: Java could not be found in the defined PATH.
- FMC34023E: error 126 loading javai.dll.
- FMC34023E: error errorcode loading libname. The explanation given was that the configuration checker could not load the library libname. In response, you must make sure that the library is installed and that the path is contained in the LIBPATH (AIX and OS/2), LD_LIBRARY_PATH (Sun Solaris), SHLIB_PATH (HP-UX) or PATH (Windows) statement.
- FMC34811E: the Java Version 1.1.8 is not supported by the MQWF Java API.

- FMC34811E The Java Version version is not supported by the MQWF Java API. The explanation given was that the Java version installed is currently not supported by MQSeries Workflow. In response, make sure you are using Java Version 1.1.*x* where *x* is greater than six. For MQSeries Workflow Version 3.3.0 and subsequent release levels you must use Java Version 1.2.2 or later.

Chandra Upadhyayula
Programmer Analyst (USA)

Creating and using generic profiles

WebSphere MQ (WSMQ) V5.3 introduces the concept of generic profiles with the aim of enabling easier administration of WSMQ object security. Essentially, generic profiles allow the specification of several wildcard types within object names when setting authorities. This allows an administrator to set the individual or group access rights to many different objects using just one command.

This article will examine the wildcard types and their use and explain how profile management using the **setmqaut** and new **dmpmqaut** commands works. It will assume some basic familiarity with WSMQ security, such as setting and displaying authorities.

CREATING PROFILES

A profile is a set of information consisting of a profile name, an object type, an entity name, and an authority. The entity name may be either a specific user-ID or a group name. Each set of information specifies the authority that the entity has to all objects of the specified type with names that match the profile name. In this context 'match' means a wildcard character match in accordance with a set of rules that will be outlined later in this article.

Profiles are created using the existing **setmqaut** command. This command remains largely unaltered from the previous version of MQ; the only difference is that the *-n* parameter is now used to indicate a profile name instead of an object name. An outline syntax of the WSMQ 5.3 version of the command is as follows:

```
   setmqaut -m <queue manager name> -n <profile name> -t <object type>
-
   p/-g<principal/group> <authority specification>
```

A more detailed syntax example can be found in WSMQ publications.

The profile name may take one of two forms: fully specified, in which case the profile name contains no wildcard characters, or generic, when wildcard characters are used.

If the profile name is fully specified an object of the type indicated by the *-t* parameter must already exist with an object name identical to the profile name. In this way the administrator can specify the access rights that a user has to a particular object and the behaviour of the **setmqaut** command is unaltered from previous versions of WSMQ. If the profile name is not the same as an existing object a relevant error message will be displayed when issuing the **setmqaut** command.

If the profile name is generic, ie if it contains wildcard characters, clearly it will not be identical to the names of any existing objects. Additionally, it need not wildcard match any existing objects at the time of creation of the profile.

All generic profiles are persistent. They apply to all existing objects of the specified type that they wildcard match at the time of creation; additionally, they apply to all objects of the specified type created in the future with object names which wildcard match the profile name.

Once a profile has been created it remains in effect until explicitly removed. It is possible for the authority that a user has to an object to be set to null by giving an authority specification of *-all*. There will be, however, times when it will be useful to remove the profile specifically. This is achieved by giving an authority specification of *+remove* or if you prefer, *-remove*.

Having covered the creation of profiles using the **setmqaut** command it will be useful to take a look at some examples. Before doing this there are two further important concepts to consider – naming objects and the wildcard matching rules.

OBJECT NAMES

It has long been the practice to use the period or full stop (.) to act as a separator in object names, eg SYSTEM.AUTH.DATA.QUEUE. The separators have previously been treated simply as part of the character string forming the object name. The introduction of generic profiles, however, brings functional significance to the way in which object names are structured using these separators.

In the example queue name SYSTEM.AUTH.DATA.QUEUE the separators split the object name into four parts: SYSTEM, AUTH, DATA, and QUEUE. These parts are known as qualifiers. A qualifier can be defined as a contiguous sequence of characters delimited by either the period or the string boundary.

Generic profiles bring functional significance to qualifiers because as well as the wildcards being capable of matching various combinations of characters they may also be used to match qualifiers. This will become clearer by examining the following wildcard matching rules and examples.

WILDCARD MATCHING RULES OVERVIEW

Wildcards are special characters that can be included within profile names to match zero, one, or more characters or qualifiers within object names. There are three types of qualifier:

- Specify a question mark (?) in a profile name to match any single character in an object name.
- Specify an asterisk (*) as either:
  - a qualifier in a profile name to match any one qualifier in an object name
  - a character within a qualifier in a profile name to match zero or more characters within a qualifier in an object name.
- Specify a double asterisk (**) once in a profile name as either:
  - the entire profile name to match all object names
  - a beginning, middle, or ending qualifier to match zero or more qualifiers in an object name.

These rules will be familiar to users with experience of RACF. They allow an administrator to specify a variety of generic profiles that will match various objects at the character level or, if preferred, at the higher structural level of qualifiers.

Having examined the various structures and rules which make up the functional behaviour of generic profiles let's look at the creation of a generic profile and how it controls access to various objects.

EXAMPLE PROFILE CREATION

For the purpose of illustration let's assume that there exists a queue manager called QMGR on which queues with the following names have been defined:

- QUEUE.ADMIN.A.
- QUEUE.ADMIN.B.
- QUEUE.DEFAULT.A.
- QUEUE.DEFAULT.B.

Similarly, there exist operating system users with user-IDs USER1 and USER2.

Consider the case where the system administrator wants to grant authority to USER1 to be able to put to the two administration queues. In previous versions of MQ it would have been necessary to use two **setmqaut** commands but with generic profiles only one command is necessary:

```
   setmqaut -m QMGR -n "QUEUE.ADMIN.*" -t q -p USER1 +put
```

This command creates a generic profile allowing USER1 to put to all queues with an object name that matches the profile name QUEUE.ADMIN.*. Clearly, using the wildcard rules outlined previously, the first two queues of our example match the profile name and, hence, after issuing the **setmqaut** command USER1 is able to put to the two administration queues as required. Note that profiles with the names QUEUE.ADMIN.? or QUEUE.ADMIN.** would also have matched here but in a more complicated example it may be necessary to choose carefully which of the three is most appropriate.

In a similar way consider the case where the administrator wants to grant full authority to USER2 to both the administration queues and the default queues. Again, whereas four **setmqaut** commands would previously have been necessary, this can now be accomplished by creating one generic profile:

```
   setmqaut -m QMGR -n "QUEUE.**" -t q -p USER2 +all
```

Further points worth noting here include the fact that it is advisable to enclose profile names in quotation marks (",") on Unix platforms, otherwise the wildcard characters may be affected by the command line parser before being passed to the administrative commands, such as **setmqaut** and **dmpmqaut**, for processing. Also note that all profiles are case-sensitive and that although QUEUE.** matches the four queues queue.** does not.

From these examples hopefully it is apparent how the naming of objects, and the use of qualifiers in particular, takes on extra significance. It is important when considering naming schemes for objects to anticipate ways in which the administrator may wish to allocate access rights. It may, for example, be useful to ensure that groups of objects that will require similar access rights share a particular qualifier in their names.

WILDCARD MATCHING RULES IN DETAIL

Although the matching rules are quite simply stated it is worth considering a few further example generic profiles to emphasize how the rules work in practice. The following examples give a generic profile name along with lists of object names, some of which match the profile and some which do not.

Profile name	Matching objects		Nonmatching objects	
QUEUE.?.DEFAULT	QUEUE.A.DEFAULT	(1)	QUEUE.DEFAULT	(3)
	QUEUE.B.DEFAULT	(2)	QUEUE.AB.DEFAULT	(4)
			QUEUEX.A.DEFAULT	(5)

In the above example (1) and (2) match because the A and B characters match '?'. (3) does not match as '?' must match precisely one character – it is not permitted to match no characters nor, as in example (4), more than one character. (5) does not match as the rest of the profile name, excluding the '?' character, must match the object name exactly.

Profile name	Matching objects	Nonmatching objects

```
  QUEUE.*.DEFAULT   QUEUE.A.DEFAULT      (1)   QUEUEX.ADMIN.DEFAULTY
(4)
                    QUEUE.ADMIN.DEFAULT  (2)   QUEUE.ADMIN.X.DEFAULT
(5)
                    QUEUE..DEFAULT       (3)   QUEUE.DEFAULT
(6)
```

The '*' character may be specified, as in the above example, as a qualifier to represent one or more qualifiers in the object name. Hence (1) and (2) match; the length of the qualifier is irrelevant so (1) matches even though the qualifier is only a single character.

Note that a qualifier of null length is also permitted and so (3) also matches. As with the previous example (4) does not match because the remaining qualifiers in the profile name, excluding that represented by the '*' character, must match the qualifiers in the object name exactly. (5) does not match because '*' is only permitted to match precisely one qualifier and ADMIN.X consists of two qualifiers. In a similar way (6) does not match because '*' is not permitted to match no qualifiers.

```
  Profile name      Matching objects           Nonmatching objects
  QUEUE.AD*N        QUEUE.ADMIN          (1)   QUEUE.AD.N
(3)
                    QUEUE.ADN            (2)   QUEUE.ADNX
(4)
```

The '*' character is specified in this example as part of a qualifier, which implies that it will match zero or more characters in an object name. So in this example (1) matches because the '*' matches the characters 'MI'. (2) also matches as * may match zero characters. (3) does not match as the object name contains three qualifiers, whereas the profile name contains only two. (4) does not match because the part of the qualifier following the '*' character in the profile name does not match the end of the last qualifier in the object name.

```
  Profile name      Matching objects             Nonmatching objects
  QUEUE.**.DEFAULT  QUEUE.ADMIN.DEFAULT   (1)    QUEUE.DEFAULT
(3)
                    QUEUE.ADMIN.X.DEFAULT (2)    QUEUEX.DEFAULTY
(4)
```

Specifying '**' in a profile name as a qualifier matches any number of qualifiers in the object name, hence (1) and (2) clearly match. (3) also matches as it is permitted for '**' to match zero qualifiers. Finally, (4) does not match as qualifiers in the profile name other than ‘**’ must match qualifiers in the object name exactly.

PRIORITY

The flexibility provided by the various wildcard characters means that it is possible to create generic profiles such that more than one profile name matches a particular object. Given that access rights are based also on users and their group membership, the possibility of several profiles matching an object name for each of the various users and groups involved in an access check would lead to a degree of complexity. In order to prevent this, a concept called profile priority is introduced.

Profile priority states that, in the case where two profiles match an object name and type for a particular user or group, only the profile with the most specific profile name applies. The most specific profile is determined by comparing the two profile names from left to right. The profile with the most specific character at the point where the two differ is deemed the most specific. In comparing characters a non-generic character is deemed more specific than a generic character. In comparing two generic characters a question mark is more specific than an asterisk, which in turn is more specific than a double asterisk.

To illustrate the concept of profile priority consider the following example. Let’s assume we have a queue manager called QMGR on a queue called QUEUE.ADMIN.A. Now let’s construct two generic profiles using the following commands:

```
setmqaut -m QMGR -n "QUEUE.ADMIN.?" -t q -p USER1 +put
setmqaut -m QMGR -n "QUEUE.ADMIN.*" -t q -p USER1 +get
```

The first command creates a generic profile granting operating system user-ID USER1 put access to all queues that match the profile QUEUE.ADMIN.?. The second creates a generic profile granting the same user-ID get access to all queues that match the profile QUEUE.ADMIN.*.

Now consider the case where an authority check is performed for USER1 to determine what access rights the user has to the queue QUEUE.ADMIN.A. The key point to note in this example is that both of the generic profiles match the queue name. In this case the concept of profile priority is relevant and so only the most specific profile will apply. Comparing the two profile names from left to right: the profile names differ at the point of the last character. Both characters are generic, but following the definition outlined earlier a question mark is deemed more specific than an asterisk. The profile QUEUE.ADMIN.? is, therefore, deemed the most specific and will be the only one that applies when performing the authority check. This means that USER1 has put access only to the queue QUEUE.ADMIN.A.

DISPLAYING PROFILES

Profiles are displayed using the new **dmpmqaut** command. An outline syntax diagram of the command is as follows:

```
dmpmqaut -m <queue manager name> [-n <profile name>/ -l] -t <object
type> -p/-g <principal/group>
```

Again, a more detailed syntax example can be found in WSMQ publications. The parameters are similar to those used by the **dspmqaut** and **setmqaut** commands, with notable differences being that all parameters are optional and that a *-l* parameter may be specified in place of a profile name to produce a terse list.

The **dmpmqaut** command dumps all authority profiles that match the specified parameters. In this way the parameters are, in essence, acting as a filter: only profiles with fields that match those of the specified parameters are displayed. The use of the **dmpmqaut** command is, therefore, varied. By specifying no parameters at all a complete list of all profiles may be obtained. Conversely, by specifying all parameters, details of one particular profile may be examined.

It is important to note the difference between the two administrative commands **dmpmqaut** and **dspmqaut**. **Dspmqaut** is unchanged for this release of WSMQ; it still performs the clearly defined function of displaying the authority that a particular entity has to a specified object. In doing so it takes account of all appropriate profiles and, in the case of the entity being a user-ID, it also takes account of the user’s group membership and any profiles that may be applicable to those groups.

Conversely, the **dmpmqaut** command displays the actual profiles, including generic and non-generic profiles, which have been explicitly created by the administrator, along with profiles that the system creates. The latter include those which give the creator and the mqm group full access to objects. Profiles displayed by the **dmpmqaut** command are used by the **dspmqaut** command and the OAM when calculating the authority that an entity has to a particular object.

When dumping authority records using the **dmpmqaut** command, the profiles that are displayed match all specified parameters exactly, with the one exception being that of the *-n* profile name parameter. When this parameter is non-generic, profiles with generic names that match the object name are also displayed. This allows the administrator to determine which profiles have been defined on the system that match a particular object.

Note that if the *-n* parameter specifies a generic profile the profiles that are displayed must have a profile name that matches the parameter exactly.

Dmpmqaut examples

A few examples of the **dmpmqaut** command along with typical output which the command may produce in each case show the different ways in which the command may be used. The examples are based on a system where the queues and profiles have been defined as in the previous section on example profile creation.

Example

```
dmpmqaut -m QMGR -n "QUEUE.ADMIN.*" -t q -p USER1

profile:     QUEUE.ADMIN.*
object type: queue
entity:      USER1
entity type: principal
authority:   get
```

In this example the *-n* parameter is generic and so only profiles with a profile name that matches this parameter exactly are displayed. Displayed profiles must also match the other parameters and so only profiles for queues and user-ID USER1 are included.

Example

```
dmpmqaut -m QMGR -n "QUEUE.ADMIN.A" -t q

profile:     QUEUE.ADMIN.*
object type: queue
entity:      USER1
entity type: principal
authority:   get
```

```
profile:     QUEUE.**
object type: queue
entity:      USER2
entity type: principal
authority:   allmqi dlt chg dsp clr
profile:     QUEUE.ADMIN.A
object type: queue
entity:      administrator
entity type: principal
authority:   allmqi dlt chg dsp clr
profile:     QUEUE.ADMIN.A
object type: queue
entity:      mqm
entity type: principal
authority:   allmqi dlt chg dsp clr
```

In this example the *-n* parameter is non-generic and so all profiles for queues with names which the object name matches are displayed. This includes the generic profiles with profile names QUEUE.ADMIN.* and QUEUE.ADMIN.** along with the non-generic profiles with profile names matching the object name QUEUE.ADMIN.A exactly, which were created when the object was created. Note that in this example no user-ID or group name parameters are specified and so no filtering based on the entity name occurs.

Example

```
dmpmqaut -m QMGR -l

profile: @CLASS, object type: authinfo
profile: SYSTEM.MQSC.REPLY.QUEUE, object type: queue
profile: SYSTEM.CLUSTER.COMMAND.QUEUE, object type: queue
profile: SYSTEM.ADMIN.QMGR.EVENT, object type: queue
profile: SYSTEM.DEFAULT.AUTHINFO.CRLLDAP, object type: authinfo
profile: SYSTEM.DEFAULT.PROCESS, object type: process
profile: SYSTEM.ADMIN.COMMAND.QUEUE, object type: queue
profile: SYSTEM.CICS.INITIATION.QUEUE, object type: queue
profile: SYSTEM.CHANNEL.INITQ, object type: queue
profile: SYSTEM.DEFAULT.NAMELIST, object type: namelist
profile: SYSTEM.DEFAULT.INITIATION.QUEUE, object type: queue
profile: SYSTEM.CLUSTER.REPOSITORY.QUEUE, object type: queue
profile: SYSTEM.DEFAULT.MODEL.QUEUE, object type: queue
profile: SYSTEM.DEAD.LETTER.QUEUE, object type: queue
profile: SYSTEM.PENDING.DATA.QUEUE, object type: queue
profile: SYSTEM.DEFAULT.LOCAL.QUEUE, object type: queue
profile: SYSTEM.CHANNEL.SYNCQ, object type: queue
profile: SELF, object type: qmgr
profile: SYSTEM.DEFAULT.REMOTE.QUEUE, object type: queue
profile: SYSTEM.DEFAULT.ALIAS.QUEUE, object type: queue
```

```
profile: SYSTEM.CLUSTER.TRANSMIT.QUEUE, object type: queue
profile: SYSTEM.ADMIN.PERFM.EVENT, object type: queue
profile: @CLASS, object type: queue
profile: @CLASS, object type: namelist
profile: @CLASS, object type: process
profile: @CLASS, object type: qmgr
profile: SYSTEM.ADMIN.CHANNEL.EVENT, object type: queue
```

The *-l* parameter instructs the command to display a terse list in which only the profile name and type of object to which the profile applies is displayed. Note that in this example the only other parameter is the queue manager name. In this case no filtering of the authority records occurs and so the command generates a list of all defined profiles for the specified queue manager.

The above output is a typical indication of the profiles that are generated when a queue manager is created with the profiles listed being those associated with the default queue manager objects. These profiles typically give the creator and mqm group full access to the default objects.

In addition there are two special types of profile. Authorities for the queue manager itself are stored in profiles with the name SELF. Authorities for object classes are stored in profiles with the name @CLASS.

RECOVERY

The introduction of the **dmpmqaut** command enables users to take a record of all authority profiles for a particular queue manager. WSMQ V5.3 also stores entity type information within an authority profile and the combination of these two new features enables users to store and then later recreate all authorities associated with a queue manager.

Once a system administrator has defined all the profiles which they will later wish to recover, running the **dmpmqaut** command and specifying only the queue manager name will display all profiles for that queue manager in verbose format. The output from the command may be piped to a text file and stored until recovery is required.

To recover the authorities consider a typical profile entry in the stored text file of the form:

```
profile:     <profile name>
object type: <object type>
entity:      <user name>
```

```
  entity type: <principal/group>
  authority:   <authority>
```

To reconstruct a profile, issue a **setmqaut** command of the form:

```
  setmqaut -m <queue manager> -n "<profile name>" -t <object type> -p/-
g
  <user name> <authority>
```

The *-p* flag should be specified in the case where the entity type is a principal or, alternatively, the *-g* flag if it is a group. So for example, if the profile entry is:

```
  profile:     QUEUE.ADMIN.*
  object type: queue
  entity:      USER1
  entity type: principal
  authority:   get
```

the **setmqaut** command required to reconstruct the profile on a queue manager named QMGR is:

```
  setmqaut -m QMGR -n "QUEUE.ADMIN.*" -t queue -p USER1 +get
```

Note that in the special cases where the profile name is @CLASS or SELF the **setmqaut** command must be issued against an object of the appropriate type or the queue manager respectively.

Performing this process of issuing **setmqaut** commands for all profiles listed in the previously saved **dmpmqaut** output will recreate exactly all the authorities that existed at the point when the **dmpmqaut** command was run.

David Postlethwaite, MQSeries Development
IBM Hursley (UK)

Editor's note:

In last month's issue of *MQ Update* a Web URL for two files was omitted from Stefan Raabe's article entitled *Multiple CKTI trigger monitor transactions in CICS*. The files (STCKTI1 and STCKTI2, mentioned on page 36) can be found at www.xephon.com/extras/stckti.txt.

MQ news

CommerceQuest has introduced its CommerceQuest Suite for IBM WebSphere, which is said to act as an enabling engine to create additional connectivity to enhance WebSphere software capabilities, enable faster implementations, and complement existing systems such as WebSphere and WebSphere MQ.

This suite helps IBM and CommerceQuest sites minimize the integration effort needed to integrate existing data and applications as XML Web services interfaces to support new WebSphere-powered applications. It will enable functional access and visibility to distributed disparate data and applications.

The suite comprises applications, tools, and expertise to support mainframe applications, MQ Integration middleware, and WebSphere. The specific elements include Rapid CICS Enabler for WebSphere, Rapid Web Services Enabler for WebSphere, Rapid Application Enabler for WebSphere, and Rapid Database Enabler for WebSphere.

Other tools and applications include Rapid File Transfer Enabler for WebSphere, Rapid MQ Enabler for WebSphere, Rapid MQ Integrator Enabler for WebSphere, and Rapid POS Enabler for WebSphere.

For more information contact:
CommerceQuest, 2202 N Westshore Blvd, Tampa, FL, 33607, USA.
Tel: +1 813 639 6300.
Fax: +1 813 639 6900.
Web: http://www.commercequest.com

CommerceQuest (UK), Doncastle House, Doncastle Road, Bracknell, Berkshire, RG12 8PE, UK.
Tel: +44 1344 861010.
Fax: +44 1344 861011.

* * *

MQSoftware has announced that its Q Pasa! middleware management solution now supports IBM WebSphere Business Integration.

Q Pasa! Version 3.0 provides end-to-end monitoring and administration capabilities for WebSphere Business Integration, as well as support for WebSphere Application Server 4.0, WebSphere MQ Everyplace, and z/OS.

The new version of Q Pasa! also offers a new architecture designed to support larger configurations, an enhanced Java-based Management Console, and additional security authorization and administration functionality.

With the new release, MQSoftware is also rolling out a training course on WebSphere Business Integration Implementation.

For more information contact:
MQSoftware,1660 South Highway 100, Suite 400, Minneapolis, Minnesota 55416, USA.
Tel: +1 952 345 8720.
Fax: +1 952 345 8721.

MQSoftware, Surrey Technology Centre, 40 Occam Road, Surrey Research Park, Guildford, Surrey, GU2 7YG, UK.
Tel: +44 1483 295400.
Fax: +44 1483 573704.

* * *