22

MQ

April 2001

In this issue

upgrade

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* is available from Xephon's 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.

Parsing XML messages using the MRM

To enable an MQSI V2 message flow to parse a legacy message, the message format must be defined to the MRM. All the fields contained within the legacy message and their corresponding field lengths defined in the message set must be brought together in the message type, giving a unique message set identifier and message identifier, against which the message itself may be parsed. The choice of which message set identifier and message identifier to use is, of course, provided either in the RFH2 header or in the MQInput node default information in the message flow.

Having created this environment, legacy messages may then be successfully parsed in a message flow and the data manipulated in the required manner.

Legacy messages, by their nature, already exist within organizations. They can be passed into a message flow for processing without modification. But what about messages created by a new application, or a partner organization, which is restricted to creating messages in XML format? Such messages might be processed by an adaptor, which will convert the XML format to the desired legacy format prior to processing by the message flow. However, it is possible to create messages in XML format, which can utilize this MRM definition.

The XML message must contain embedded tags which exactly match the MRM data names; for example, look at the following MRM definition.

MESSAGE SETS

```
ACCT_SET                Identifier:  DFØSDM006SØ01
Messages
     ACCTMSG            Identifier:  ACCTMSGID
                        Type:        ACCTREC
        ACCTNO   STRING
        ACCTYPE  STRING
        CATEGORY STRING
Types:
     ACCTREC
        ACCTNO   STRING
        ACCTYPE  STRING
        CATEGORY STRING
```

The RFH2 header will contain the following information:

```
<mcd>
     <Msd>MRM</msd>
     <Set>DFØSDMØØ6SØØ1</Set>
     <Type>ACCTMSGID</Type>
     <Fmt>XML</Fmt>
     </mcd>
```

The XML data should appear in the input record as follows:

```
<MRM>
     <ACCTMSGID>
          <ACCTNO>123456<\ACCTNO>
          <ACCTYPE>TTR<\ACCTYPE>
          <CATEGORY>M<\CATEGORY>
     </ACCTMSGID>
</MRM>
```

The important detail in the RFH2 header is the <Fmt> field setting of 'XML'. This would normally contain the value 'MRM' with the data following being supplied in legacy format. A value of 'XML' forces the message flow to use a standard XML parser and assign the values to appropriate field names defined in the MRM. These fields are then accessible in message-processing nodes, as if they had been derived from a message in legacy format.

Ken Marshall
MQSeries Consultant, MQSolutions (UK)

MQSeries V5.1 for NT: security for client/server

BACKGROUND

This article looks in depth at MQSeries V5.1 client/server security with specific reference to the NT platform. This documentation will apply equally well to MQ V5.2. There are a number of benefits with MQSeries V5.1 client, including ActiveX controls. However, on MQSeries V5.1 server, Object Authority Management (OAM) is enabled by default. This means that, unless the correct authorizations have been set up, the MQSeries V5.1 client will not be able to access the queue manager or any objects within it.

This is because MQSeries V5.1 Unix and NT clients send the logged-on user-id to be authenticated at the server. If your SVRCONN channel MCAUSER is blank, the client user-ids have to exist at the server and be properly authorized. Authorization definitions appear to be cached, so the queue manager has to be recycled in order to pick up the new definitions.

However, MQSeries V5.0 clients do not send the logged-on user-id. The user-id that is sent is taken from the environment variable **MQ_USER_ID**. If this is not set, the user-id used to authenticate the client's MQI calls is the user-id under which the SVRCONN channel runs. For Unix, this is specified in *at/etc/inetd.conf*. For NT, it is the user-id under which **runmqlsr** is started. Therefore, there is a potential security risk when using the V5.0 client.

Please note that, while you can use MQ V5.0 client with MQ V5.1 server, I have found that you cannot always reliably use the MQ V5.1 client with MQ V5.0 server.

While the work on which this article is based was carried out using MQ V5.1 server, the setting and displaying of authorizations applies equally to V5.0. However, the constraints in terms of using the V5.0 client as pointed out above should be remembered, and it is recommended that the **MQ_USER_ID**, at least, is specified when using the V5.0 client.

This article will:

- Document how the MQSeries OAM works within NT.
- Define the process of setting authorizations.
- Recommend the best approach for providing client/server security within MQSeries V5.1 on NT.

HOW OAM WORKS WITHIN NT

Outline

The OAM works in conjunction with NT's own security mechanisms and provides authorization services for MQSeries. It is enabled by default on MQ V5.1.

Access is granted to MQSeries queue managers and objects within the queue manager using the **setmqaut** command. It is important to remember that, when granting access to a particular object, such as a queue, access must also be granted to the queue manager.

On NT, access can be granted to users or groups. Users belonging to a group will inherit the access rights granted to their group. If the user belongs to more than one group their access rights will be a combination of the those groups. If access rights have been set for a specific user then these rights will take precedence over any group rights the user may inherit. For instance:

- User1 belongs to group MQUsers.
- User2 belongs to group MQUsers.
- MQUsers has been granted connect access to queue manager QM1.
- MQUsers has been granted put/get access to local queue *QM1.QL1* in queue manager QM1.
- Get access has been removed from User1.

As a result of the above:

- User2 can put/get messages to/from *QM1.QL1* in queue manager QM1.
- User1 can put messages to *QM1.QL1* in queue manager QM1.

The specifics of the **setmqaut** command are covered in more detail in the section *Setting object authorizations within MQSeries V5.1.*

MQSeries for Windows NT handles the user-id of the logged-on user in two ways:

- The user-id is stored in the message descriptor of messages when they are created.
- The user-id is used when MQSeries performs authorization checks for access to MQSeries objects, such as queues.

A user-id can be up to 20 characters long and can be domain-qualified, eg *user@domain*. The domain name can be up to 15 characters long.

A group-id can also be specified and this can be up to 64 characters long.

The OAM for V5.1 has been rewritten to allow security identifier and domain information to be specified in addition to a user-id.

The Windows NT clients send the logged-in user-id and the user's SID when they connect to a queue manager. They no longer read the **MQ_USER_ID** and **MQ_PASSWORD** environment variables. However, the Windows '95 or '98 clients do send the **MQ_USER_ID** if set, or the current user-id if logged in.

The Security Identifier (SID)

In addition to the user-id, the Windows NT Security Identifier (SID) records information identifying the full user account details on the Windows NT security account manager (SAM) where the user is defined. Within MQSeries, the SID is stored in the Accounting Token field of the MQ Message Descriptor.

MQSeries directly queries the SAM identified by the SID when performing authorization checks. For this query to succeed the SAM database must be available. On NT, this could be the local database, the primary domain, or any trusted domain, all of which are checked.

If the OAM receives an authority request that does not contain a Windows NT SID, the security policy for the queue manager determines how the OAM behaves.

MQSeries security policies

The default MQ security policy allows the OAM to receive authority requests that do not contain a Windows NT security identifier. This would be the case with multi-platform environments.

In this situation the OAM attempts to resolve the user-id into a Windows NT SID by searching:

- The local security database.
- The security database of the primary domain.
- The security database of the trusted domain.

If the security policy is set to **NTSIDsRequired** both the user-id and NT SID information must be passed to the OAM. A check is made to ensure the two are consistent. It should be noted that the supplied user-id is compared with the first 12 characters of the id associated with the SID.

DEFINING SECURITY POLICIES

Within MQSeries V5.1 the following entries are added automatically to the NT registry for each queue manager created. The Service specifies the **AuthorizationService** service while the **ServiceComponent** specifies the name of the service and module to use, which by default is the OAM. If you are using the OAM supplied you can also specify a **SecurityPolicy** on the Service stanza. The possible values are:

- **Default**. If, for a particular user, a Windows NT security identifier (NT SID) is not passed to the OAM, the relevant security databases are searched in an attempt to find the appropriate SID.
- **NTSIDsRequired**. An NT SID must be passed to the OAM to perform the security checks.

The security policy is added as follows:

- Within MQSeries Explorer, right-click on the queue manager and select the Properties.
- Click on the Services tab.
- Select **AuthorizationService** from the Services drop down box.
- Select **NTSIDsRequired** from the SecurityPolicy drop down box then click OK.

SETTING OBJECT AUTHORIZATIONS WITHIN MQSERIES V5.1

Outline

The **setmqaut** command is used to define access rights on queue managers and queue objects. It is important to remember that access must be granted to the queue manager as well as to the objects within the queue manager.

Access can be granted to individual users or groups of users. When a user is defined to more than one group that user's access is a combination of his/her groups' access. Access rights defined specifically for a user override the user's group access.

The **dspmqaut** command can be used to view the access rights for a particular user or group.

This section will cover the format of the commands and how they may be used in a typical environment. For further information on the use of the commands see the *MQSeries Systems Management Guide*.

In the examples given below, the queue manager is QM1 with local queues *QM1.Q1* and *QM1.Q2*.

The setmqaut command

The format of the **setmqaut** command is: Setmqaut **–m <qmgr> -t <obj type> -n <obj.name>** -s <service component> -p <user> -g <group> <auth>, where **bold** parameters are required:

-m <qmgr> The name of the queue manager: optional if changing default queue manager.

-t <obj type> Object type for which authorizations are to be changed. Possible values are:
- q or queue
- prcs or process
- qmgr
- nl or namelist.

-n <obj name> The name of the object for which authorizations are to be changed. It must not be generic.

-s <service component>
This only applies if you are using installable authorization services. It can be ignored for our purposes.

-p <user> This specifies the name of the principal, ie user whose authorizations are to be changed. On NT this can also specify a domain, ie *userid@domain.*

-g <group> Specifies the name of the group whose authorizations

are to be changed. At least one principal or group must be specified. More than one principal and/or group can be specified but each must be prefixed '–p' or '–g' respectively.

<auth> Specifies the authorizations to be given (prefixed '+') or removed (prefixed '-'). A number of authorizations can be specified in one **setmqaut** command. The authorizations that can be given are categorized as follows:

- authorizations for issuing MQI calls
- authorizations for MQI context
- authorizations for issuing commands for administration tasks
- generic authorizations.

In Appendix A, Table 1 lists all the authorizations.

Granting access to the queue manager

In order for a user to access queues or other MQ objects they must first be granted access to the queue manager itself. For application developers it is sufficient to grant connect access to the queue manager, for example:

```
Setmqaut -m QM1 -t qmgr +connect -p mousem
```

This command would grant connect access to queue manager QM1 for user mousem.

```
Setmqaut -m QM1 -t qmgr +connect -g mqusers
```

This command would grant connect access to queue manager QM1 for group mqusers.

Granting access to queues

Once access has been granted to the queue manager, appropriate access to the queue objects can be specified. For application developers it is sufficient to grant connect access to the queue manager, ie:

```
Setmqaut -m QM1 -t q -n QM1.Q1 +put -get -p mousem
```

This command would enable user mousem to put messages to *QM1.Q1* but prevent the user from retrieving messages.

```
Setmqaut -m QM1 -t q -n QM1.Q2 +put +get -g mqusers -p duckd

Setmqaut -m QM1 -t q -n QM1.Q2 -put -p mousem
```

The first command grants **put/get** access to queue *QM1.Q2* for group mqusers and user duckd (who, in this instance, is not a member of the mquser group).

The second command prevents user mousem from putting messages to *QM1.Q2*. If user mousem is also a member of group mqusers then he would be able to get messages from *QM1.Q2* by virtue of the first command.

Note that, in addition to the above commands, a further **setmqaut** would need to be issued to give **connect** access to queue manager QM1 for group mqusers and users duckd and mousem.

The dspmqaut command

The format of the **dspmqaut** command is: Dspmqaut –m <qmgr> **-t <obj type> -n <obj.name>** -s <service component> -p <user> -g <group> , where **bold** parameters are required.

-m <qmgr>	Specifies the name of the queue manager on which the enquiry is to be made.
-t <obj type>	Object type on which the enquiry is to be made. Possible values are: – q or queue – prcs or process – qmgr – nl or namelist.
-n <obj name>	Name of object on which the enquiry is to be made. It must not be generic. It should not be included if enquiring on the queue manager.
-s <service component>	This only applies if you are using installable authorization services. It can be ignored for our purposes.
-p <user>	Specifies the name of the principal, ie user whose authorizations are to be changed.

	On NT this can also specify a domain, eg *userid@domain*
-g <group>	Specifies the name of the group whose authorizations are to be changed. Only one principal or group must be specified.

OPTIONS AND RECOMMENDATIONS

Detailed below is a brief summary of the options available when configuring MQSeries V5.1 client/server security.

(1) By-passing MQSeries security

Set the **mqsnoaut** variable to 'yes' prior to creating the queue manager, or remove the authorization service component from the queue manager. The advantages include:

- No security checking so no problems with client access. Any client with the appropriate **MQSERVER** variable or channel table will be OK.

The disadvantages include:

- There is no way to restrict access on the queue manager.

(2) Take the user-id from the server connection channel

Set the **MCA User-id** on the server connection channel to a valid user defined on the server. Ensure this user has access to the MQ objects. The advantages include:

- Any client with the appropriate **MQSERVER** variable or channel table will be OK.

The disadvantages include:

- This does not provide real security – any client using the channel will get on.

(3) Take user-id passed by client, define MQ access on principals

Each user accessing the queue manager objects will have to be defined as a principal using the **setmqaut** command. The advantages include:

- This option gives tight security control at the user level.

The disadvantages include:

- You need to define every user using **setmqaut**. New users will also need to be defined.

(4) Take user-id passed by client, define MQ access on principals, NTSIDsRequired

Each user accessing the queue manager objects will have to be defined as a principal using the **setmqaut** command. The advantages include:

- This option gives tight security control at the user level.
- It ensures the user is coming from a valid client – without **NTSIDsRequired,** if a matching user-id is defined on the server, it will be used instead.

The disadvantages include:

- You need to define every user using **setmqaut**. New users will also need to be defined.

(5) Take user-id passed by client, define MQ access on groups

Each user accessing the queue manager objects will have to be defined as a member of a group. The group is then given access to the queue manager objects. (Note: do not put the users in the mqm group as they will get administrative functions.) The advantages include:

- This option gives tight security control at the user level.
- You only need to set access options once to the group, regardless of how many new clients access the server.

The disadvantages include:

- You need to define every user to the group via the user manager.

Recommended approach

Option five, detailed above, is the one recommended. Options one and two do not provide any security. The third and fourth options provide tight security, but at the expense of heavy maintenance and potential disruption to the client base.

From an MQ perspective, the last option provides simpler maintenance. Different groups can be defined to represent different types of access. Users can be added to more than one group to get combined access. While this does involve user manager maintenance it is probably more straightforward than using the **setmqaut** command. This can also be performed by the server maintenance teams and does not require specialist MQ knowledge.

You can add domain groups into the local group defined on the MQ Server. For instance, assume you have defined local group mqusers on the MQ server. If you have a number of users in a domain group called 'Ipswich', then this group can be added to the mqusers group, thus removing the need to define all users individually.

Please note that the group mqm should not be used – this group has administration rights to all the queue managers that users defined to 'group inherit'. Adding users to this group effectively by-passes any security measures.

SUMMARY

If you are planning to upgrade an MQSeries V5.0 server to V5.1, your existing clients will continue to function as before and no additional measures in terms of granting access to MQ objects will be required to those you already had in place for V5.0. If you do upgrade your clients to V5.1 then you will also need to upgrade your MQ server to V5.1 and grant the necessary access rights. Remember that access needs to be granted to the queue manager as well as the queues.

To facilitate maintenance and recovery it is advisable to put your authorization requests in a command file. It is also recommended that you identify different types of MQ users and create local groups in the NT security database for each of these. Access to MQ objects should then be based on these security groups.

Users can be added into the appropriate MQ group in order to gain access. For ease of maintenance, domain groups can be added to the MQ groups, removing the need to define individual users. Please note that the mqm group should only contain users who need to have administration rights to all the queue managers.

APPENDIX A: MQ AUTHORIZATIONS

Table 1 lists all possible authorization values that can be specified on the **setmqaut** command. For further information on each of these you should refer to the following sections in the IBM *MQSeries Systems Administration* guide:

- 17.27.1 Authorizations for MQI calls.
- 17.27.2 Authorizations for context.
- 17.27.3 Authorizations for commands.
- 17.27.4 Authorizations for generic operations.

Authority	Type	Qmgr	Queue	Process	Namelist
All	Generic	Yes	Yes	Yes	Yes
Alladm	Generic	Yes	Yes	Yes	Yes
Allmqi	Generic	Yes	Yes	Yes	Yes
Altusr	MQI calls	Yes	No	No	No
Browse	MQI calls	No	Yes	No	No
Chg	Commands	Yes	Yes	Yes	Yes
Clr	Commands	No	Yes	No	No
Connect	MQI calls	Yes	No	No	No
Crt	Commands	Yes	Yes	Yes	Yes
Dlt	Commands	Yes	Yes	Yes	Yes
Dsp	Commands	Yes	Yes	Yes	Yes
Put	MQI calls	No	Yes	No	No
Inq	MQI calls	Yes	Yes	Yes	Yes
Get	MQI calls	No	Yes	No	No
Passall	Context	No	Yes	No	No
Passid	Context	No	Yes	No	No
Set	MQI calls	Yes	Yes	Yes	No
Setall	Context	Yes	Yes	No	No
Setid	Context	Yes	Yes	No	No

Table 1: Authorization values

APPENDIX B: TESTS

The following tests were carried out using an NT workstation connected to a server on which MQ V5.1 was installed. The workstation had MQSeries V5.0 installed initially, and the first test ensured that this client worked with an MQSeries V5.1 server. MQSeries V5.1 client

was then installed on this machine. The **MQSERVER** variable was used to connect the client to the server. The user-id for the tests initiated on my machine was mousem.

Queue manager QM2 was created specifically for the tests. The listener for this queue manager was on port 1415. The following user-ids were used on the server:

Mqboss	This id is in the administrator's group, and was used to create queue manager QM2 and set object authorizations.
mousem	This id was used for testing access to the queue manager.
mqtest	This id was used for testing access to the queue manager.

Test cases

1 Put/get messages to/from local queue on MQ V5.1 server using MQ V5.0 client.
- user on workstation: mousem
- user on server: Mqboss
- result: puts/gets worked successfully.

2 Put messages to local queue on MQ V5.1 server using MQ V5.1 client. User mousem is not defined as a local user on the MQ V5.1 server.
- user on workstation: mousem
- user on server: Mqboss
- result: put fails with RC 2035 – user not authorized.
  Following error message in *amqerr01.log* on server:
  05/11/00 08:52:21
  AMQ8075: authorization failed because the SID for entity mousem cannot be obtained
- explanation: the Object Authority Manager was unable to obtain a SID for the specified entity
- action: ensure that the entity is valid, and that all necessary domain controllers are available.

3 Put messages to local queue on MQ V5.1 server using MQ V5.1 client. User mousem is defined as a local user on the MQ V5.1 server. The password for mousem on the server is different from the password for mousem on the client.
- user on workstation: mousem
- user on server: Mqboss
- result: put fails with RC 2035 – user not authorized.
  Following error message in *amqerr01.log* on server:
  05/11/00 09:03:52
  AMQ8077: entity mousem has insufficient authority to

access object QM2

- explanation: the specified entity is not authorized to access the required object. The following requested permissions are unauthorized: connect
- action: ensure that the correct level of authority has been set for this entity against the required object, or ensure that the entity is a member of a privileged group.

4 User mousem on MQ V5.1 server is added to the mqm group.
Put messages to local queue on MQ V5.1 server using MQ V5.1 client.

- user on workstation: mousem
- user on server: Mqboss
- result: put fails with RC 2035 – user not authorized
  Following error message in *amqerr01.log* on server:
  05/11/00 09:06:58
  AMQ8077: Entity mousem has insufficient authority to access object QM2
- explanation: the specified entity is not authorized to access the required object. The following requested permissions are unauthorized: connect
- action: ensure that the correct level of authority has been set for this entity against the required object, or ensure that the entity is a member of a privileged group.

Note

Tests five to eight were carried out on the MQ V5.1 server. No client workstation was involved.

5 Issue **runmqsc** command on MQ V5.1 server against queue manager QM2.

- user mousem is in the mqm group
- user on workstation: not applicable
- user on server: mousem
- result: command fails with RC 2035 – user not authorized. Following error message in *amqerr01.log* on server:
  05/11/00 09:12:01
  AMQ8077: entity mousem has insufficient authority to access object QM2
- explanation and action: as above.

6 Issue **crtmqm** command on MQ V5.1 server.

- user mousem is in the mqm group
- user on workstation: not applicable
- user on server: mousem
- result: queue manager created successfully.

7 Removed mousem from mqm group.

– issue **crtmqm** command on MQ V5.1 server
– user on workstation: not applicable
– user on server: mousem
– result: command fails with RC 2035 – user not authorized.
  Following error message in *amqerr01.log* on server:
  05/11/00 09:18:01
  AMQ8077: entity mousem has insufficient authority to access object QM2
– explanation and action: as above.

8a Access granted to the following MQ objects:
– connection to queue manager QM2:
  `setmqaut -m QM2 -t qmgr +connect -p mousem.`
– **put/get** access to local queue *QM2.Q1*:
  `setmqaut -m QM2 -n QM2.Q1 -t q +put +get -p mousem.`

8b Put/get messages to/from local queue *QM2.Q1* on MQ V5.1 under user mousem.
– user on workstation: not applicable
– user on server: a) Mqboss, b) mousem
– result: puts/gets worked successfully.

9 Put/get messages to/from local queue *QM2.Q1* on MQ V5.1 server using MQ V5.1 client.
– user on workstation: mousem
– user on server: mousem
– result: puts/gets worked successfully.

10a Create local queue *QM2.Q2* in queue manager QM2.

10b Put/get messages to/from local queue *QM2.Q1* and *QM2.Q2* on MQ V5.1 server using MQ V5.1 client.
– user on workstation: mousem
– user on server: a) Mqboss, b) Mqboss
– result: puts/gets worked successfully on *QM2.Q*
  put/get on *QM2.Q2* fails with RC 2035 – user not authorized
  Following error message in *amqerr01.log* on server:
  05/11/00 15:21:33
  AMQ8077: entity mousem has insufficient authority to access object *QM2.Q2*
– explanation: the specified entity is not authorized to access the required object. The following requested permissions are unauthorized: **put**
– action: ensure that the correct level of authority has been set for this entity against the required object, or ensure that the entity is a member of a privileged group.

11a Access removed to local queue *QM2.Q1* on queue manager QM2:
`setmqaut -m QM2 -n QM2.Q1 -t q -put -get -p mousem.`

11b Put/get messages to/from local queue *QM2.Q1* on MQ V5.1 under user mousem
- user on workstation: mousem
- user on server: Mqboss
- result: put fails with RC 2035 – user not authorized
  following error message in *amqerr01.log* on server:
  05/11/00 15:43:20
  AMQ8077: entity mousem has insufficient authority to access object *QM2.Q1*
- explanation: the specified entity is not authorized to access the required object. The following requested permissions are unauthorized: put.
- action: ensure that the correct level of authority has been set for this entity against the required object, or ensure that the entity is a member of a privileged group.

12a Created group mqtest on MQ V5.1 server and add user mousem to this group. Access granted to the following MQ objects for group mqtest:
- connection to queue manager QM2:
  ```
  setmqaut -m QM2 -t qmgr +connect -g mqtest
  ```
- put/get access to local queue *QM2.Q1*:
  ```
  setmqaut -m QM2 -n QM2.Q1 -t q +put +get -g mqtest.
  ```

12b Put/get messages to/from local queue *QM2.Q1* on MQ V5.1 under user mousem
- user on workstation: mousem
- user on server: Mqboss
- result: put fails with RC 2035 – user not authorized.
- following error message in *amqerr01.log* on server:
  05/11/00 15:49:46
  AMQ8077: Entity mousem has insufficient authority to access object *QM2.Q1*
- explanation: the specified entity is not authorized to access the required object. The following requested permissions are unauthorized: put
- action: ensure that the correct level of authority has been set for this entity against the required object, or ensure that the entity is a member of a privileged group.

Note

Access to queue *QM2.Q1* for user mousem was specifically restricted. This takes precedence over the group access granted to mqtest.

13 User bunnyb is defined as a local user on the MQ V5.1 server and is added to group mqtest. Put/get messages to/from local queues *QM2.Q1* and *QM2.Q2* under user bunnyb.
- user on workstation: not applicable
- user on server: bunnyb
- result: put/get worked successfully to *QM2.Q1* but returned 2 0 3 5

against *QM2.Q2*

– expected as group mqtest was granted access to *QM2.Q1* but not *QM2.Q2*.

14 User mousem has no put/get access to *QM2.Q1*. Amended server connection channel *C1.CLIENT.QM2* to include fudde in MCA user field. Put messages to local queues *QM2.Q1* under user mousem.

– user on workstation: mousem
– MCA user: fudde
– user on server: Mqboss
– result: put fails with RC 2035 – user not authorized.
  Following error message in *amqerr01.log* on server:
  05/11/00 08:52:21
  AMQ8075: authorization failed because the SID for entity fudde cannot be obtained.
– explanation: the Object Authority Manager was unable to obtain a SID for the specified entity
– action: ensure that the entity is valid, and that all necessary domain controllers are available.

15 User mousem has no put/get access to *QM2.Q1*.
Defined fudde as a local user on MQ V5.1 server and added to group mqtest. Put messages to local queues *QM2.Q1* under user mousem.

– user on workstation: mousem
– MCA user: fudde
– user on server: Mqboss
– result: put works; group mqtest has access to put/get messages to/from *QM2.Q1*.

Note

Tests 16 to 22 test the use of security policies and methods of bypassing security.

16 Open command prompt.
Set **mqsnoaut=yes**, then issue **crtmqm** command to create queue manager QM3. Define queue *QM3.Q1* in queue manager QM3. Put messages to local queue *QM3.Q1* under user mousem.

– user on workstation: mousem
– user on server: Mqboss
– result: put works.

17 Delete queue manager QM3 then recreate using MQSeries explorer. Define queue *QM3.Q1* in queue manager QM3. Put messages to local queue *QM3.Q1* under user mousem.

– user on workstation: mousem
– user on server: Mqboss
– result: put fails with RC 2035 – user not authorized.

18 Delete AuthorizationService from queue manager QM3 via the MQSeries Services panels. Stop and restart QM3. Put messages to local queue *QM3.Q1* under user mousem.
   – user on workstation: mousem
   – user on server: Mqboss
   – result: put works.

19 Delete queue manager QM3 then recreate using MQSeries explorer. Define queue *QM3.Q1* in queue manager QM3. Put messages to local queue *QM3.Q1* under user mousem.
   – user on workstation: mousem
   – user on server: Mqboss
   – result: put fails with RC 2035 – user not authorized.

20 Use the **setmqaut** command to grant access to QM3 and local queue *QM3.Q1* for user mousem. Put messages to local queue *QM3.Q1* under user mousem.
   – user on workstation: mousem
   – user on server: Mqboss
   – result: put works.

21 Amend the security policy to **NTSIDsRequired**. Stop and restart QM3. Put messages to local queue *QM3.Q1* under user mousem.
   – user on workstation: mousem
   – user on server: Mqboss
   – result: put fails with RC 2035 – user not authorized.

22 Reset the security policy to default. Stop and restart QM3. Put messages to local queue *QM3.Q1* under user mousem.
   – user on workstation: mousem
   – user on server: Mqboss
   – result: put works.

Mark Richards (UK)

E-mail alerts

If you'd like to be notified when new issues of *MQ Update* have been placed on our Web site, you can sign up for our e-mail alert service, which notifies you when new issues (including new free issues) have been placed on our Web site. To sign up, go to http://www.xephon.com/mqupdate.html and click the 'Receive an e-mail alert' link.

Freeing a hung Unix queue manager

INTRODUCTION

From time to time, most people using MQSeries on a Unix system will have had experience of a queue manager that does not respond to any valid MQ command at all. Using **runmqsc** against it just results in a hung system, as does any attempt to shut it down (including **endmqm -p**). Typically, this happens on heavily overloaded development or test systems, and often in an environment where application programming teams have the freedom to do as they please. MQ's problems in these circumstances are often the result of system resource shortages or contentions, but the end result is the same – the queue manager is well and truly broken!

Given this situation, the only remedial action you can take is to end all of the queue manager's processes manually, using the Unix **kill** command.

THE CORRECT SEQUENCE

When using this method to stop a queue manager it is important that the various processes involved are terminated in the correct order. Table 1 details the sequence required.

On systems with inbound channels from other remote queue managers it is probable that there will be a number of **amqcrsta** processes running. These are spawned by the inet daemon detecting connections on the port assigned to the queue manager in question. These can also be terminated, although MQ is quite tolerant of them being there when you eventually try to restart the queue manager.

These processes can be identified with the Unix **ps** command. For example, to stop the command server for a queue manager called TESTQM1, you can first issue the following command to identify its process-id:

```
ps -ef | grep amqpcsea | grep TESTQM1,
```

which results in the following output:

```
mqm  7407  1  0 09:58:40 ?  0:00 amqpcsea TESTQM1.
```

Order	Function	MQ process name
1	Command server	amqpcsea
2	Logger	amqhasmx
3	Linear log formatter	amqharmx
4	Checkpoint processor	amqzllpØ
5	Queue manager agents	amqzlaaØ
6	Processing controller	amqzxmaØ
7	Cluster repository process	amqrrmfa

Table 1: Correct order for terminating processes

To terminate the process you will need to issue a **kill** command for it. Frequently, **kill** used on its own will not succeed in removing the process, and in these cases, you will need to add a terminate signal to the command, with **-9** being the most powerful. So, to be one hundred per cent certain of terminating the process shown above, you would need to issue:

```
kill -9 74Ø7
```

Note that, in order to issue the Unix **kill** command for any MQ process, you will need to be logged on as either mqm or a superuser.

MQ'S IPCS USAGE

Often, where a queue manager has hung on a system that is very heavily overloaded, killing the queue manager's processes will still not allow it to be restarted. When this happens, the **strmqm** command that has been issued also appears to hang, even though there are no processes running.

Under these circumstances, the problem will almost certainly be with MQ's use of semaphores and shared memory sets – and these too will need to be freed before the queue manager can be restarted.

A problem here is that, on systems where multiple queue managers are being run, MQ does not differentiate between them in its usage of such IPCS resources: ie it is impossible to tell which semaphores are being used by which queue manager, they are all shown as simply owned by mqm. As a result, you will need to shut down any other active queue managers before using the system **ipcrm** command to terminate them.

MQ's IPCS resources can be identified by issuing the following command:

```
ipcs | grep mqm
```

This will result in output that looks like the following:

```
m      12323   0x00231b2e --rw-rw----      mqm      mqm
m      12324   0x00231b2f --rw-rw----      mqm      mqm
m      12325   0x00231b56 --rw-rw----      mqm      mqm
m      12326   0x002112f2 --rw-rw-rw-      mqm      mqm
m      12327   0x002112f3 --rw-rw-rw-      mqm      mqm
m      12328   0x002112f4 --rw-rw-rw-      mqm      mqm
s      65548   0x80217cc3 --ra-ra----      mqm      mqm
s      65549   0x802261e9 --ra-ra----      mqm      mqm
s     196622   0x0022b145 --ra-ra----      mqm      mqm
s     196623   0x00208e71 --ra-ra-ra-      mqm      mqm
s      65552   0x00201ac0 --ra-ra-ra-      mqm      mqm
```

To terminate an IPCS resource you need to use the **ipcrm** command together with its type and id, so to remove the first one shown above you could issue:

```
ipcrm -m 12323
```

Clearly, having to go through each one like this can be very tedious and time-consuming, particularly on heavily-used queue managers. Fortunately, thanks to the power of Unix facilities, there is a single command that will remove them all. You can issue:

```
ipcrm `ipcs | grep "mqm" | cut -c1-12 | sed 's/[m^s]/\-&/g'`
```

Again, note that you will need to be logged on either as mqm or a superuser to do this.

THE **KILLQMGR.KSH** UTILITY

This utility carries out all of the actions detailed above in a single shell script, making it possible to free a hung queue manager with a single one-line command. Below, I have listed some points to note about the utility and its code.

- The utility takes one parameter – the name of the queue manager to be stopped. Some help text can be displayed by entering the command **killqmgr.ksh -h**.
- First, the queue manager's processes are counted by using **grep -c**, with **grep -v** used before this to eliminate the **grep** process

itself. If no processes are found for the specified queue manager, the utility is exited.

- There are certain processes which may not be running, for example the command server, or the linear log formatter if you are using circular logging. For these particular 'optional' processes a check is first made to see if they are actually there before issuing the appropriate **kill** command.
- Based on past experience, I've used **kill -9** here to ensure that the process is terminated.
- Once the seven processes listed in Table 1 have been killed, I've used a 'catch-all' to terminate any other process running that can be identified as being associated with the specified queue manager.
- Once all processes have been terminated, the utility prompts the user as to whether they want to kill off MQ's IPCS resources as well. If the answer is 'y', the system is checked for any other queue manager that might be running.
- If other queue managers are found, their names are obtained by searching for their processing controllers, again using **ps -ef**. One problem here is that different versions of Unix will show the queue manager's name within this output at different offsets: AIX will show it starting in column 62 whereas Solaris will have it in column 60. If you are running on other versions of Unix you will need to tailor these lines appropriately.
- In shutting down these other queue managers I've made the assumption that they will respond to an **endmqm -i** command to stop them. This may need to be changed to **endmqm -p** if they still refuse to come down, as the shell script could hang at this point if they don't.

KILLQMGR.KSH

```
# Script for killing Queue Manager
# and optionally IPCS resources for MQ
#      Help Text Requested ?                                  #
if  [[ $1 = "-h" ]]
then
     print "*********************************************"
     print "* killqmgr.ksh                                *"
```

```
     print "*                                                *"
     print "* This is a utility for freeing a hung           *"
     print "* Queue Manager - by killing all of its        *"
     print "* processes (in the correct sequence), and       *"
     print "* optionally all of its IPCS resources.          *"
     print "*                                                *"
     print "* Usage is: killqmgr.ksh qmgrname                *"
     print "************************************************"
     exit 1
fi
#      Queue Manager entered as parameter ?                  #
if  [[ $# -ne 1 ]]
then
     print "Usage is: killqmgr.ksh qmgrname"
     exit 1
fi
#      Are any processes actually running for this QM ?      #
prcs=`ps -ef | grep -v "grep" | grep -c "$1"`
if  [[ $prcs -eq 0 ]]
then
     print "No processes running for $1 ..."
     exit 1
fi
#      Kill all QM's processes in correct sequence           #
print "Shutting Down $1 Command Server if running ..."
print " "
ps -ef | grep "$1" | grep "amqpcsea" >/dev/null
if  [[ $? -eq 0 ]]
then
 kill -9 2>/dev/null `ps -ef | grep "$1" | grep "amqpcsea" | cut c10-15`
fi
 print "Shutting Down $1 Logger ..."
print " "
kill -9 2>/dev/null `ps -ef | grep "$1" | grep "amqhasmx" | cut -c10-15`
print "Shutting Down $1 Log Formatter if running ..."
print " "
ps -ef | grep "$1" | grep "amqharmx" >/dev/null
if  [[ $? -eq 0 ]]
then
kill -9 2>/dev/null `ps -ef | grep "$1" | grep "amqharmx" | cut -c10-15`
fi
print "Shutting Down $1 Checkpoint Processor ..."
print " "
kill -9 2>/dev/null `ps -ef | grep "$1" | grep "amqzllp0" | cut -c10-15`
print "Shutting Down $1 Queue Manager Agents ..."
print " "
kill -9 2>/dev/null `ps -ef | grep "$1" | grep "amqzlaa0" | cut -c10-15`
print "Shutting Down $1 Processing Controller ..."
print " "
kill -9 2>/dev/null `ps -ef | grep "$1" | grep "amqzxma0" | cut -c10-15`
```

```
print "Shutting Down $1 Repository Process if running ..."
print " "
ps -ef | grep "$1" | grep "amqrrmfa" >/dev/null
if  [[ $? -eq Ø ]]
then
 kill -9 2>/dev/null `ps -ef | grep "$1" | grep "amqrrmfa" | cut-c1Ø-15`
fi
print "Killing off any other processes for $1 ..."
print " "
ps -ef | grep -v grep | grep "$1" >/dev/null
if  [[ $? -eq Ø ]]
then
    kill -9 2>/dev/null `ps -ef | grep "$1" | cut -c1Ø-15`
fi
#      Kill IPCS Resources for mqm ?                        #
print "Do you want to kill of MQ's IPCS resources (y/n) ?:"
read ans
if  [[ $ans != "y" ]]
then
     print "Exiting ..."
     exit 1
fi
 qmgrs=`ps -ef | grep -v "grep" | grep -c "amqzxmaØ"`
if  [[ $qmgrs -ne Ø ]]
then
     print "There is at least one other Queue Manager running ..."
     print "Killing MQ's IPCS resources will affect these too ..."
     print "Shut down other Queue Manager(s) (y/n) ?:"
     read ans2
     if  [[ $ans2 != "y" ]]
     then
          print "Exiting ..."
          exit 1
     fi
fi
#      Are we on AIX or Solaris here ?                      #
os=`uname`
if  [[ $os = "AIX" ]]
then scol="62"
elif  [[ $os = "SunOS" ]]
then scol="6Ø"
else scol="6Ø"
fi
ps -ef | grep -v "grep" | grep "amqzxmaØ" | cut -c$scol-9Ø > @tmp1
awk '{
      tfile = "@qmgr."NR
      print $Ø > tfile
      }' @tmp1
#      Shut down other active QMs                           #
i=1
while [ $i -le $qmgrs ]
```

```
do
  qm=`cat @qmgr.$i`
  print "Shutting Down $qm ..."
  endmqm -i $qm
  i="`expr $i + 1`"
done
#      Kill all mqm's IPCS resources                          #
print "Killing off all MQ's IPCS resources .."
ipcrm `ipcs | grep "mqm" | cut -c1-12 | sed 's/[m^s]/\-&/g'`
print "Ready for Queue Manager Restart ..."
rm @*
```

Chris Bell
Systems Consultant, British Airways (UK)

MQSI V2 performance trace analysis

THE PROBLEM

The problem we had to address, and which this article describes, was to search for performance bottlenecks in an MQSI V2 user trace, which could have many message flows, each with up to 500 nodes in the flow. Assistance in identifying which nodes took the most time was also desired.

THE SOLUTION

The **mqsichoptrace.pl** program is used to reformat the MQSI V2 user trace formatted output to enable the user to diagnose performance bottlenecks. A 'debug' level trace is recommended, as it shows the user ESQL statements. It does this by:

- Keeping track of the elapsed time for each node.
- Simplifying the presentation of the data.
- Attempting to mark the start and end of flows and nodes.

Any **MQPUT** statements are also marked. Marking the end of a node isn't as clear-cut as one would like, but the eye should be drawn to the right area of the trace.

The latest stable version of Perl is 5.6.0 and it can be downloaded from

www.perl.com. MQSI V2 runs on Windows NT, SUN, and RS/6000 platforms, as does **mqsichoptrace.pl**.

EXECUTION OF **MQSICHOPTRACE.PL**

Take the output from *mqsiformatlog* and supply it as input to the **mqsichoptrace.pl** Perl program. **mqsichoptrace.pl** is a Perl (V5) program with two parameters:

- *--f=input_file_name* (mandatory).
- *--l= -1* or *0* or *1* (optional, 0 is default).

Example command line

```
perl mqsichoptrace.pl  --f=trace.fil  --l=-1 >summary.txt
```

EXAMPLE OUTPUT

```
14:18:13.937   136   The   Execution   Group   'sherwin'  has  processed
a configuration message successfully.
14:18:13.937  136  MQPUT  to  queue 'SYSTEM.BROKER.EXECUTIONGROUP.REPLY'
on queue manager 'CSIM': MQCC=0, MQRC=0; node
'ConfigurationMessageFlow.outputNode'.
14:18:13.937  136  Message successfully output by output node
'ConfigurationMessageFlow.outputNode' to queue
'SYSTEM.BROKER.EXECUTIONGROUP.REPLY' on queue manager 'CSIM'.
****** message put   - Elapsed Time from previous 0.000  *****
******************* New Flow starting  ******************************
14:18:21.750  473 Parser type 'Properties' created on behalf of node
'Input XML' to  handle portion of incoming message of length 0 bytes
beginning at offset '0'.
14:18:21.750  473  Parser type 'MQMD' created on behalf of node 'Input
XML' to  handle  portion  of incoming message of length '364' bytes
beginning at offset  '0'.  Parser  type  selected  based  on value
'MQHMD' from previous parser.
14:18:21.750  473  Message  being  propagated  to the output terminal;
node 'Input XML'.
****** Leaving Node   - Elapsed Time from previous 0.000  *****
14:18:21.750 473 Parser type 'XML' created on behalf of node 'Input XML'
to handle  portion  of  incoming  message  of length '1681' bytes
beginning at offset  '364'.  Parser  type  selected  based  on value
'XML' from previous parser.
14:18:21.764  473 Node 'LoopControl': Executing statement 'SET
OutputRoot = InputRoot;' at (4, 1).
14:18:21.764  473  Node 'LoopControl': Evaluating expression 'InputRoot'
at (4, 18).
14:18:21.764 473 Node 'LoopControl': (4, 5) : Navigating path element.
```

```
14:18:21.764 473 Node 'LoopControl': Performing tree copy of 'InputRoot'
to 'OutputRoot'.
14:18:21.764  473 Node 'LoopControl': Executing  statement  'SET  dLoc =
CARDINALITY(OutputDestinationList.Destination.RouterList.DestinationData[])
+ 1;' at (6, 1).
14:18:21.764  473 Node 'LoopControl': Evaluating expression
'CARDINALITY(OutputDestinationList.Destination.RouterList.DestinationData[])
+ 1' at (6, 88).
14:18:21.764  473  Node 'LoopControl': Evaluating  expression
'CARDINALITY(OutputDestinationList.Destination.RouterList.DestinationData[])'
at (6, 12).
14:18:21.764  473  Node 'LoopControl': (6, 46) : Failed to navigate to
path element.
14:18:21.781  473  Node 'LoopControl': Assigning value ''Propogate'' to
'OutputDestinationList.Destination.RouterList.DestinationData[dLoc].labelname'.
14:18:21.781  473  Message  propagated  to  out  terminal from compute
node 'LoopControl'.
****** Leaving Node   - Elapsed Time for previous Ø.Ø31  *****
14:18:21.781  473  Message  propagated  to target label node by router
node 'PropogateOrEnd'.
****** Leaving Node - Elapsed Time for previous Ø.ØØØ  *****
14:18:21.781 473 Message propagated to out terminal from node
'Propogate'.
****** Leaving Node - Elapsed Time for previous Ø.ØØØ  *****
14:18:21.781 473 Message propagated to match terminal by check node
'FlowOrder1'.
****** Leaving Node - Elapsed Time for previous Ø.ØØØ  *****
14:18:21.781  473  Node  'MakeNewDoc':  Executing statement 'SET I = 1;'
at (1Ø, 1).
14:18:21.812 473 Node 'MakeNewDoc': Performing tree copy of
'InputRoot.XML.SW_CUSTOMER.BILLTO' to
'OutputRoot.XML.SW_CUSTOMER.BILLTO'.
14:18:21.812 473 Node 'MakeNewDoc': Executing statement 'SET
OutputRoot.XML.SW_CUSTOMER.SHIPTO  =
InputRoot.XML.SW_CUSTOMER.SHIPTO[I];'at (19, 1).
14:18:21.812  473  Node  'MakeNewDoc': Evaluating expression
'InputRoot.XML.SW_CUSTOMER.SHIPTO[I]' at (19, 41).
14:18:21.812 473 Node 'MakeNewDoc': Evaluating expression 'I' at (19,
74).
14:18:21.812 473 Node 'MakeNewDoc': (19, 5) : Navigating path element.
14:18:21.812 473 Node 'MakeNewDoc': Performing tree copy of
'InputRoot.XML.SW_CUSTOMER.SHIPTO[I]' to
'OutputRoot.XML.SW_CUSTOMER.SHIPTO'.
14:18:21.812  473  Message  propagated  to  out  terminal from compute
node 'MakeNewDoc'.
****** Leaving Node   - Elapsed Time from previous Ø.Ø31  *****
14:18:21.812  473  MQPUT  to  queue 'TESTOUT1' on queue manager '':
MQCC=Ø,
MQRC=Ø; node 'Qoutput'.
14:18:21.812  473  Message  successfully output by output node 'Qoutput'
to queue 'TESTOUT1' on queue manager ''.
```

```
****** message put   - Elapsed Time from previous 0.000  *****
14:18:21.827  473  Message  propagated  to  match  terminal  by  check
node 'FlowOrder1'.
****** Leaving Node   - Elapsed Time from previous 0.015  *****
******************** New Flow starting  ******************************
14:18:30.406 136 Parser type 'Properties' created on behalf of node
'ConfigurationMessageFlow.InputNode'  to handle portion of incoming
message of length 0 bytes beginning at offset '0'.
14:18:30.406   136   Parser   type   'MQMD'   created   on   behalf   of
node 'ConfigurationMessageFlow.InputNode' to handle portion of incoming
message of  length  '364' bytes beginning at offset '0'. Parser type
selected based on value 'MQHMD' from previous parser.
14:18:30.406  136  Message  being  propagated  to the output terminal;
node 'ConfigurationMessageFlow.InputNode'.
****** Leaving Node   - Elapsed Time from previous 0.000  *****
Finished:
```

RECOMMENDATION ON USE

Given a large user trace file with lots of nodes being executed, run with level *-1*. This gives just two lines for every node and it should be simple to spot the longest elapsed times. Re-run with the default of 0 (or just leave it off) to give more detail. It is usually best to pipe this output to a new file. For completeness, a parameter value *--l=1* will show all trace data in the original, albeit in a better format.

SOURCE CODE

```
#!/usr/bin/perl
use Getopt::Long;
$secstor=0;
$secsold=0;
$nodemsg=" ";
$outline="";
if ( !GetOptions( "f=s","l:i" ) ) {
   usage();
}
# l=-1 gives summary detail and l=1 gives most detail; l=0 is default
if ( $opt_f ne undef ) {
   $file  = lc $opt_f;
} else {
   usage();
}
if ( $opt_l eq undef ) {
   $detail  = 0;
} else {
   $detail = $opt_l;
}
```

```
unless (open (FILE, "$file")) {
die "couldn't open the input trace file $file: $!";}
while  (<FILE>) {  # read the formatted MQSI V2 trace file
   chomp($_);
   if  (/UserTrace/)  {  #  trace  line  with most information
   if  (($secsold  !=  Ø)  and  (($nodemsg  ne " " and $detail >=-1 ) or
($nodemsg eq " " and $detail > -1 ))) {
         chomp($outline);
         print "$outline\n";
      }
      if  ($nodemsg ne " " ) {  # detected what we think is node end
         $secsdiff = sprintf("%.3f",$secsdiff);
         $msgline  =  "******  $nodemsg - Elapsed  Time  from  previous
         $secsdiff  *****";
         $nodemsg = " ";
         print "$msgline\n";
      }
     ($dat,$tim,$thread,$ut,$bip,$body) = split(' ',$_,6);
     ($hr,$min,$sec) = split(':',$tim);
     $secs = substr($sec,Ø,6);  # assumes nn.nnn format
     $secsnew=($hr*36ØØ)+($min*6Ø)+$secs; # total seconds from midnight
     if  ((/Parser  type/)  and  (/Properties/)) { # the trigger to
     start a flow
        $secstor = $secsnew;
        $msgline = " ";
        print "$msgline\n";
        $msgline  =  "********* New Flow starting ******************";
        print "$msgline\n";
     }
     $secsdiff=$secsnew-$secsold;
     if ($secsold == Ø) {  # 1st time through processing only
        $secstor = $secsnew;
     }
     $secsold = $secsnew;
     $timt = join(':',$hr,$min,$secs);
     $outline=  join('  ',$timt,$thread,$body);  # put the interesting
     bits back together
     if (/propagated/)  {                  # triggers to exit a node
        $secsdiff=$secsnew-$secstor;
        $secstor = $secsnew;
        $nodemsg = "Leaving Node";
     }
     elsif (/Message successfully output/) { # final put node
         $secsdiff=$secsnew-$secstor;
         $secstor = $secsnew;
         $nodemsg = "Message Put";
     }
}
   elsif ($detail > Ø ) {                    # print all trace lines?
     s/\s+/ /;
     $outline = $outline . $_;
```

```
   }
}
print  "Finished:\n";
exit 0;
sub usage {
   print "usage mqsichoptrace (options)
   options:
   --f=filename [--l= -1 or 0 or 1] \n";
   exit 1;
}
```

R E Branagan
Programmer, IBM Hursley (UK)

What's new in MQSeries for AS/400 V5.1 and V5.2

INTRODUCTION

This article looks at the changes that have been made in MQSeries versions V5.1 and V5.2 for the AS/400, and considers why it was necessary to make those changes.

The AS/400 was one of the first MQSeries platforms and its design concentrated more on performance than portability, taking advantage of a number of unique OS/400 features. For an example of this, consider the way that MQSeries queues resolve to user spaces (**USRSPC*) in the *QMQMDATA* library, or the implementation of shared memory, using user spaces and user queues (**USRQ*).

Since the first AS/400 release, MQSeries has been implemented on a number of Unix and PC operating systems and these platforms have tended to receive the bulk of Hursley's development focus. New features in MQSeries distributed platforms were not always easy to implement in MQ/400 due to differences in architecture, and, because MQ/400 was tied into the Rochester OS/400 release schedule, new function was often released later, if at all.

By the time MQSeries for AS/400 had reached V4R2M1, the gap in functionality between MQ/400 and the other distributed platforms had grown to the point where it was obvious that something needed to be done.

The decision was taken to close the function gap with the next release of MQSeries for AS/400, and that future releases of MQSeries on the AS/400 should be tied into the MQSeries distributed family release cycle. The objectives for the V5.1 release were:

- To achieve parity of function with the MQSeries Unix V5.1 platforms.
- To satisfy additional customer requirements – most significantly, allowing better configuration of MQSeries' job run attributes.
- To retain the best of the current product – for example, it was important to retain the usability of earlier releases of MQ/400.
- To provide a migration path for existing users.
- To avoid any detrimental impact on performance.

FUNCTIONAL CHANGES

The following functions have been added into MQ/400 V5.1:

- Multiple queue manager support.
- Queue manager cluster support.
- Support for two-phase commit and local and global units of work.
- Support for threaded applications.
- Enhanced work management configuration.
- Increased maximum message size (100 MB) and maximum queue size (2 GB).
- Support added for **MQCONNX**, **MQCMIT**, **MQBACK**, and (to some extent) **MQBEGIN**.
- Enhanced MQSeries classes for Java and bindings for COBOL.

The following functions are not supported in MQ/400 V5.1:

- The *Admin.* application. This was a product option in V4R2M1 and earlier releases, which has now been dropped. The new **WRKMQM** command now provides some of the same function

in the base product, but remote administration of queue managers is better handled by the NT MQSeries explorer.

- Publish/subscribe. There is no publish/subscribe broker for MQ/400, but MQ/400 queue managers can participate in a publish/subscribe network.
- XA transaction coordination. At the time of writing, MQ/400 does not support participation in XA transactions.

Discussion of the new function is divided into the following sections:

- Multiple queue managers and API changes.
- Changes to commands.
- Changes to the data storage model.
- Changes to the process model.
- Clustering and channel changes.
- Logging and recovery changes.
- Work management configuration.
- Security.

MULTIPLE QUEUE MANAGERS AND API CHANGES

One of the objectives of this release was to bring the standard V5 MQSeries API to the AS/400 while allowing applications written against earlier releases of MQ/400 to run on V5, without the need for changes or recompilation.

This was complicated because earlier versions of MQSeries for AS/400 allowed users to make implicit connections to the (only) queue manager. It was not necessary to issue an **MQCONN** to get a connection handle because a default connection handle could be passed to MQSeries functions. This feature was unique to the AS/400 and so was not supported by the V5.x API.

The delivery of new function, combined with previous release compatibility, is accomplished by supplying the V5 API in the new service programs **LIBMQM** (for non-threaded applications) and **LIBMQM_R** (for threaded applications).

For compatibility with V4R2M1, separate service programs **AMQZSTUB** and **AMQVSTUB** are supplied. These service programs do not have access to new functions like **MQCONNX**, **MQBEGIN**, **MQCMIT**, and **MQBACK**, but they do support the default connection handle, which allows implicit connections to the default queue manager.

COMMAND CHANGES

To support multiple queue managers a new parameter **MQMNAME** has been added to the commands that did not have it before. The parameter is optional and is placed after the positional parameters so that CL programs from the earlier releases will not need recompilation.

If the MQMNAME parameter is not specified it defaults to the default queue manager. This will allow sites with a single queue manager to continue using their MQSeries applications without any disruption. In addition to the new **MQMNAME** parameter, the following commands have been added or modified in other ways.

WRKMQM

The new 'work with queue managers' command (**WRKMQM**) allows top-level administration of queue managers on a machine. From this command you can see the status of all queue managers; start, stop, and change queue managers; and work with queues, channels, processes, namelists, etc.

WRKMQMTRN

The new 'work with transactions' command is needed for the two-phase commit support, which has been introduced in V5. It lists transactions which are in doubt because they have been prepared but not committed, and allows you to choose whether to commit or back them out.

ENDMQM

Over the past few releases, MQ/400 has had a number of utility programs to fully quiesce a queue manager and all jobs connected to it. In V5.1 these were combined into a single program (**AMQSTOP4**). In V5.2 this program was incorporated into the **ENDMQM** command.

The option **ENDCCTJOB** (end-connected jobs) and a time-out value have been added to facilitate this. If **ENDCCTJOB(*YES)** is specified on **ENDMQM**, then ***ALL** can be specified as the queue manager name, allowing all queue managers to be ended. The **ENDCCTJOB(*YES)** option does the following:

- Starts the command server (if necessary) and stops all channels.
- Ends the command server.
- Records all media images.
- Ends the queue manager.
- Ends any listeners.
- Ends any jobs with dangling connections to the queue manager.
- Reclaims the QMQM activation group.
- Cleans up shared memory and semaphores.
- Issues message **AMQ6154** if successful.
- Issues message **AMQ6153** if unsuccessful.

WRKMQMCHST

On V5.1 and earlier releases, the 'work with channel status' command only showed **SAVED** channel status, which was the cause of some confusion. In V5.2 we have added a new option – **CHLSTS() –** to the command to allow users to specify ***CURRENT**, ***SAVED**, or ***ALL** types of status. The panel has been modified to show what type of status is displayed.

RCDMQMIMG

A common requirement has been for an easy way to find out which journal receivers are required by MQSeries to restart the queue manager or perform media recovery. In the past, this information has been made available via the **AMQ7460** and **AMQ7462** messages, issued when a queue manager starts and when a new journal receiver is attached. In V5.2 these messages can be forced to the job log using a new option on the **RCDMQMIMG** command.

RFRMQMAUT

A new command has been added in V5.2 called **RFRMQMAUT** (Refresh MQSeries Authorities). This command performs the same function as the new MQSC command **REFRESH SECURITY**, ie it forces a refresh of the OAM's cache so that it is no longer necessary to stop and restart the queue manager for changes to group memberships to take effect.

Miscellaneous

In addition to the commands detailed, a number of new commands have been added to allow management of queue manager clusters and namelists.

DATA STORAGE CHANGES

To support multiple queue managers, the data libraries *QMQMDATA* and *QMQMPROC*, which were used in V4R2M1 and earlier releases, have been replaced with a data library for each queue manager. These libraries are named *QMxxxxxxxxxx* where *xxxxxx* is the queue manager name, if less than eight characters. If the name is more than eight characters it is truncated to eight characters. If there is already a library with that name then a two-digit counter is appended to the name.

The queue manager library contains the queue manager's local journal and receivers; the channel definition file; a queue manager-specific SNA receiver program; and the *QMQMMSG* message queue. The *QMQMMSG* message queue is used as the message queue when the queue manager starts jobs, so messages about the queue manager jobs are written to this queue. Messages that used to be sent to *QSYSOPR* are also written to this queue (for example the **AMQ7460** and **AMQ7462** messages about which receivers are required at start-up and for media recovery).

Unlike *QMQMDATA*, the new queue manager libraries do not contain any queue data or definitions – we now use stream files (***STMF**) in the IFS file system to store MQSeries object definitions and data. In this respect, MQ/400 is now very similar to the PC and Unix platforms.

When MQSeries for the AS/400 is installed, two top level IFS directories are created:

- */QIBM/ProdData/mqm*. Subdirectories below this directory contain MQSeries include files, samples, and the trace formatting file.
- */QIBM/UserData/mqm*. Subdirectories below this directory contain error logs, trace data, and a directory for each queue manager holding the information that used to be in *QMQMDATA*.

In order to perform a full backup of MQSeries you must back up all of the queue manager specific libraries, and all IFS directories below the */QIBM/UserData/mqm* directory hierarchy.

THE PROCESS MODEL

In previous releases, a user who called an MQSeries application program did not need to be authorized to access the data in *QMQMDATA*. The **AMQZSTUB** service program was owned by QMQM and ran with the QMQM adopted authority using the **USRPRF(*OWNER) USEADPAUT(*YES)** technique. This meant that, when the user's application called the MQSeries API, the QMQM profile's authorities would be adopted and the application would have access to MQSeries data in the *QMQMDATA* library.

This method for adoption of authorities does not work for data in the IFS, so from V5.1 onwards it is necessary for application jobs to communicate with separate MQSeries agent jobs to access and change MQSeries data. The agent jobs run under the QMQM user profile and so have authority to MQSeries shared resources. The application jobs communicate with the agents so the application has no direct access to MQSeries resources, eliminating the risk of an application accidentally or maliciously corrupting MQSeries data.

MQCONN is different from most calls in that the application communicates directly with the Execution Controller (EC). The Execution Controller owns the agent jobs. When an application tries to make a connection, the EC decides whether to start a new agent, to start a thread in an existing agent, or to re-use an existing agent that has just been released by another application.

Figure 1 shows the jobs started by an MQSeries queue manager and the way they relate to the user's application (shown on the left). A greater number of jobs is started for every queue manager, compared

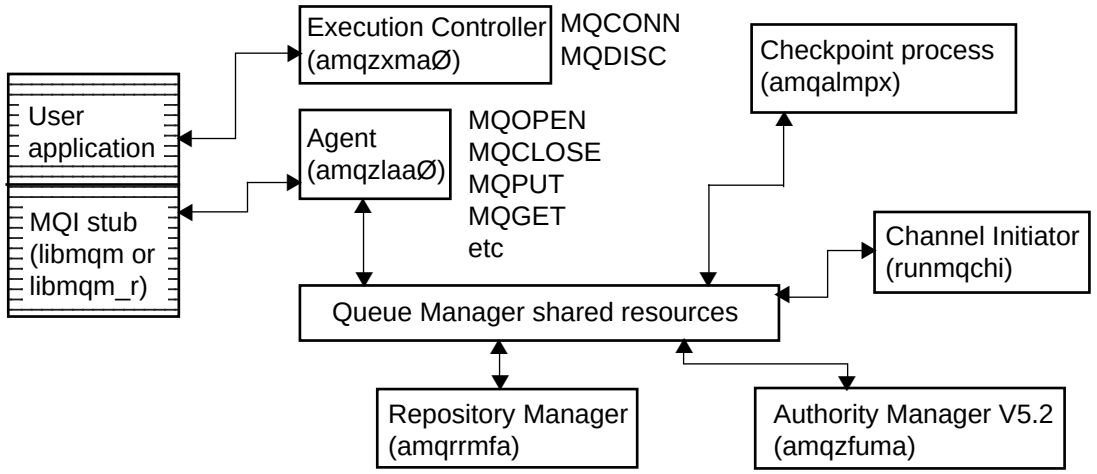

Figure 1: The process model

with earlier releases. Configuring these jobs is discussed in the section on work management. The queue manager that a job is servicing can be identified by message *AMQ7163* in the job log.

Fastpath binding

The **MQCONNX API** call (extended connect) includes an **MQCNO_FASTPATH_BINDING** option, which runs all MQSeries code in the application's job rather than using an agent job. It is effectively the old process model, but MQ/400 V5 cannot adopt authorities so the following restrictions apply:

- The application must be running in multi-threaded job mode.
- The application must be running under the QMQM user profile.
- The application must not terminate abnormally – this could lead to corruption of data.

For this reason, fastpath bindings should be used with extreme caution, if at all, and applications using fastpath bindings should be tested in-depth using the standard bindings before switching to fastpath binding.

CHANNEL CHANGES

Clustering

Clustering is a major addition to MQSeries V5. Linking queue

managers in clusters results in reduced system administration, increased availability, and the potential for workload balancing.

To simplify administration, a queue manager can send a message to any other queue manager in the same cluster without the need for explicit channel definitions, remote-queue definitions, or transmission queues for each destination – channels are defined dynamically when needed.

Because you can define instances of the same queue on more than one queue manager the workload can be distributed throughout the queue managers in a cluster. Networks become more scalable because new instances of the queue can be added by introducing new queue managers into the network.

Clustering offers increased availability because multiple instances of the same queue can be on more than one server. If one server fails then messages will continue to be routed to the other servers with instances of the queue.

TCP/IP channels

In V4R2M1 and earlier releases of MQ/400, all receiver channels ran as separate non-threaded jobs (**AMQCRSTA**).

In V5.1 this was changed so that receiver channels ran as threads within the listener job (**RUNMQLSR**). Channels that run as threads in a listener start more quickly than channels that run as jobs, and, because resources are shared within the listener, resource utilization is lower than for channels that run as jobs. However, there is a maximum of 1024 channel threads in a single listener, so they are not as scalable as **AMQCRSTA** channels, and channels that run as threads need thread-safe channel exits.

In V5.2 the decision was taken to allow channels to be configurable - they can run as threads or jobs. The configuration is done in the *qm.ini* file, and therefore, works at the queue manager level. There is a new 'ThreadedListener' value in the Channels stanza which is set to 'Yes' or 'No'.

The default is to run channels as processes (like V4) to make migration easy for customers who are still using V4R2M1, or earlier. Customers who upgrade from V5.1 and want to continue using threaded channels

must set 'ThreadedListener=Yes' in their *qm.ini* Channels stanza.

The **STRMQMLSR** command in V5 allows you to specify the TCP/IP port number to listen on. This makes it easier to run multiple listeners on different ports for a queue manager (in earlier releases the port number could only be specified in the *QMINI* file in *QMQMDATA*).

SNA channels

The configuration of SNA channels has not been changed, except that the **AMQCRC6A** program now connects to the default queue manager, rather than the only queue manager. Each queue manager library contains a copy of program **AMQCRC6B**. For a queue manager-specific SNA channel connection, use routing data that will call the appropriate **QMGRLIB/AMQCRC6B** program.

LOGGING AND RECOVERY CHANGES

Prior to MQ/400 V5.1, the only unit of work that was supported was a single-phase global unit of work. In V5 of MQSeries for the AS/400 we now support local and global units of work. The global units of work in V5 use a two-phase commit process.

Global units of work allow MQSeries and other resources, such as databases or CICS, to work in conjunction with each other. Other platforms support two types of global unit of work:

- Internal global units of work, where MQSeries acts as the coordinating transaction manager, controlling other participants (resource managers) via XA.
- External global units of work, where an external transaction manager coordinates participants, controlling the units of work.

Because XA is not supported by MQ/400, only external global units of work are allowed (with OS/400 acting as the coordinator). Global units of work are only allowed in single-threaded jobs. This is because an OS/400 unit of work has the scope of a job but an MQSeries connection has the scope of a thread. This could allow a thread which had not connected to MQSeries to attempt to commit or back-out changes in MQSeries.

A local unit of work contains only updates to MQSeries resources and

uses the API calls **MQCMIT** and **MQBACK** to commit or rollback the unit of work. MQSeries for the AS/400 identifies the unit of work type by checking whether commitment control is running with ***JOB** scope when the first **MQPUT/MQPUT1/MQGET** call is made under syncpoint.

- If commitment control is running, an external global unit of work will be started and OS/400 will be the coordinator.
- If commitment control is not running, the unit of work will be local and MQSeries will coordinate.

Once a local unit of work has started, beginning OS/400 commitment control will not change the unit of work type. Disconnecting from MQSeries will reset the type of unit of work.

MQBEGIN

XA is not supported by MQSeries for AS/400, so the **MQBEGIN API** call can't be used to register participants in a unit of work. It is provided to aid compatibility when applications are ported from other platforms. The **MQBEGIN** call will return an error:

```
(MQCC_FAILED - MQRC_ENVIRONMENT_ERROR)
```

if commitment control is running for the job and a warning:

```
(MQCC_WARNING - MQRC_NO_EXTERNAL_PARTICIPANTS)
```

if commitment control is not running.

Pending disconnect

When an external global unit of work is started, MQSeries registers a special exit program with OS/400. This exit is called by OS/400 whenever a **COMMIT** or **ROLLBACK** command is issued. The exit performs the work needed by MQSeries to carry out the commit or rollback request and so it needs a connection to the queue manager. It is, therefore, necessary to keep connections alive to a queue manager if a disconnect is issued while a global unit of work is in progress. The disconnect appears to complete normally but it is really only pending.

Once in this state, certain restrictions apply. The job can reconnect to the same queue manager and even do more work in the same unit of work, but the job cannot connect to a different queue manager, and the

user cannot end commitment control. A commit or rollback request completes the pending disconnect and lifts the restrictions.

WORK MANAGEMENT

The Work Management features of V5 allow customization of all MQSeries job attributes. There is a tiered method for configuring jobs, allowing you to change an individual job for a specific queue manager, all jobs for a queue manager, or all jobs on a system.

Work Management Objects

A number of objects are supplied in the QMQM library to assist MQSeries work management. These include:

- A QMQM subsystem where all MQSeries jobs run by default. The subsystem description is supplied preconfigured with a job queue entry for the QMQM job queue and routing entries for the supplied class descriptions.
- QMQM job queue: linked to QMQM subsystem with maximum active jobs set to ***NOMAX**.
- Class descriptions:
  - **QMQMRUN20**: high priority class, to start jobs with run priority 20
  - **QMQMRUN35**: medium priority class, to start jobs with run priority 35
  - **QMQMRUN50**: default class, to start jobs with run priority 50
  - **QMQMMSG**: default message queue.
- Job descriptions:
  - **AMQZLAA0**: queue manager agent job description
  - **AMQZXMA0**: execution controller job description.

MQSeries work management configuration

Job descriptions are the most important element when customizing MQSeries jobs.

When the job is submitted the **SBMJOB** command takes the job queue, library list, routing data, and output queue from the job description. When the job starts, the routing data from the job description is matched with routing entries in the subsystem which are used to find a class. The class description defines the job run priority and timeslice as well as less commonly used items, such as maximum number of threads, maximum temperature for storage, etc.

When MQSeries submits jobs it follows this pattern:

1 Searches for a job description matching the job name in the queue manager library.

2 If that is not found, it looks for the **QMQMJOBD** in the queue manager library.

3 If that is not found, it looks for a job description matching the job name in the QMQM library.

4 If that is not found, it looks for the **QMQMJOBD** in the QMQM library.

Work management examples

The easiest way to understand how to configure MQSeries by changing job descriptions is to look at some examples.

Example one

Configure all non-threaded receiver channels (**AMQCRSTA** jobs) for all queue managers to run at priority 35.

- Create a new job description in library QMQM called **AMQCRSTA**. Set its routing data parameter to **RTGDTA(QMQMRUN35**).
- When a receiver channel is started, MQSeries will check the queue manager library for job descriptions **AMQCRSTA** and **QMQMJOBD**. As long as these don't exist MQSeries will use the newly created **AMQCRSTA** job description in library QMQM. When the job starts, it will match its routing data with the **QMQMRUN35** routing entry in the subsystem routing table, which will force it to use the job class with run priority 35.

Example two

Configure all jobs for a specific queue manager to use a different subsystem.

- Create the new subsystem and associated objects including at least one job queue. Make sure that the subsystem's job queue entry has the maximum number of active jobs set to ***NOMAX** so that multiple MQSeries jobs can be run at the same time from a single job queue.
- Copy the QMQMJOBD job description into the queue manager library, and change its JOBQ() parameter to point to the new subsystem's job queue.

When MQSeries starts any job for the queue manager it will check for a job description matching the job name in the queue manager library. As long as that does not exist it will use the new default **QMQMJOBD** job description, and put jobs on the job queue specified by the modified JOBQ() parameter.

SECURITY

In earlier releases of MQSeries all MQSeries objects resolved to AS/400 objects (usually *USRSPCs) in the *QMQMDATA* library. When authorities were granted or revoked from MQSeries objects, those authorities were mapped to equivalent OS/400 authorities and granted or revoked from the underlying ***USRSPC** object.

In V5, MQSeries objects are represented by stream files in the IFS and it is no longer possible to map MQSeries authorities directly to their underlying objects. MQ/400 now uses the same security system as MQSeries on the PC and Unix platforms, ie the Object Authority Manager (OAM). The OAM is an MQSeries-pluggable service, which means that third parties can write their own OAM to replace the one supplied with MQSeries.

The default OAM in V5.1 uses a system whereby each MQSeries object has an associated Access Control List file. These ACL files are text files containing a list of users and a hex value for each user indicating which authorities have been granted. The ACL files can be browsed with the DSPF command to find out those users who have

authority to an object, and the **DSPMQMAUT** command will report which authorities have been granted.

The default OAM in V5.2 has been changed to store authorities in an MQSeries system queue called *SYSTEM.AUTH.DATA.QUEUE*. This provides much better performance than the ACL files, but without a text file it is not so easy to determine who is authorized to an object. To solve this, a utility program called **amqoamd** is supplied. This program can list which users are authorized to one or all objects for a queue manager.

The OAM caches authorizations the first time an object is referenced. If changes are made to a user's group authorities the caches will not automatically be updated. In V5.1 it was necessary to stop and restart the queue manager to refresh the cache. In V5.2 the new command – **RFRMQMAUT** – carries out this function.

Special authorities

The AS/400 has a number of special authorities which can be granted to user profiles. One of these is ***ALLOBJ** authority and, when this is granted to a user profile, that user can access all objects on the system.

In the initial release of V5.1, MQSeries did not honour the ***ALLOBJ** special authority, so, for example, the QSECOFR profile (all-powerful security officer which has ***ALLOBJ** authorities) did not automatically have authority to run MQSeries commands or access MQSeries data. This was fixed in V5.2 (and by PTF in V5.1) so that a user with ***ALLOBJ** authority can run all MQSeries commands and access all data.

Command authorities

Two user profiles are shipped with MQSeries V5, QMQM and QMQMADM. The QMQMADM profile is the MQSeries administration profile. Users who need to run MQSeries commands must have the QMQMADM profile as their group profile or supplementary group profile.

Mark Phillips
MQSeries Development, IBM Hursley (UK)

MQ news

IBM has announced MQSeries link for R/3, providing connection between SAP R/3 business applications and back-end systems running MQSeries.

MQSeries link for R/3 products participate with other MQSeries versions, including MQSeries Integrator and MQSeries Workflow.

It connects existing R/3 applications with other R/3, R/2, and non-SAP legacy systems. It also connects SAP R/3 with other programs across IBM and compatible platforms.

It works with the Application Link Enabling (ALE) layer of the R/3 system to transmit Intermediate Documents (IDOCs) into and out of R/3, using MQSeries to carry the information.

Out now, prices weren't announced.

For further information, contact your local IBM representative.

* * *

Fiorano software, a provider of standards-based Java messaging solutions, has shipped version 5.0 of its JMS 1.0.2-compliant Java messaging server.

New features of FioranoMQ 5 include a pluggable, scalable connection management module, Java REALMS security support, and message routing across geographically distributed servers using the FioranoMQ Repeater.

Fiorano has also released the Fiorano Test Suite for JMS. It includes automated conformance tests for each line of the JMS 1.0.2 API and provides complete profiling of all JMS features. Test Suite for JMS is available to evaluators of FioranoMQ 5 free of charge for a limited time.

For further information contact:
Fiorano Software, 718 University Avenue, Suite 212, Los Gatos, CA 95032, USA.
Tel: +1 408 354 3210
Fax: +1 408 354 0846
Web: http://www.Fiorano.com

* * *