36

MQ update

June 2002

In this issue

3 Triggering batch jobs on OS/390

12 Global transactions with MQSeries and Oracle, part 2: set-up and configuration

22 Configuring MQSeries with Microsoft Cluster Server, part 2: installation and configuration

33 Enhancing MQSeries set authorities scripts

37 PUT/GET using Correl-ID

42 MQSI V2 database administration tips

46 July 2000 – June 2002 index

48 CICS 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

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.

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 ($33.75) each including postage.

Triggering batch jobs on OS/390

Triggering is a mechanism supported by MQSeries that allows you to run a program when it is needed, namely when there is a message to be read. The mechanism involves triggering and initiation queues, a process definition, and a trigger monitor. (I will not go into detail here on triggering; please refer to the *MQSeries System Management* book.)

On almost every platform for which MQSeries is available IBM delivers a trigger monitor, *runmqtrm*. On OS/390 there is a trigger monitor only for CICS (CKTI transaction). To use the feature in batch you need to install SupportPac MA12. The use of the batch trigger monitor is not straightforward so I decided to write this article (along with some REXX code) with the aim of assisting other users.

The supplied REXX program will help you to work with the batch trigger monitor. It is meant to be triggered upon arrival of a message to a named local queue. It will then read the message (it should contain a valid job name) and submit the given job. In this way you do not have to define separate processes for every program you want to trigger. It works as a kind of dispatcher.

The program is written using MQSeries SupportPac MA18 (REXX interface to MQSeries).

INSTALLATION

To install the supplied code follow the steps given below:

1. Send the REXX code to your mainframe (ASCII mode in FTP or ASCII and CR/LF in Personal Communications file transfer).
2. Store it as a library member (RECFM=FB, LRECL=80) named TRIG.

To use it, you need to create some MQSeries object definitions along with some batch trigger monitor configuration. I will guide you through these steps.

MQTRIG

```
PARSE EXTERNAL Input /* Read the parameter and
                        split it into QMgr and QName */
QMgr = WORD(Input, 1)
QName = WORD(Input, 2)
SAY 'Messages will be read from 'QName
SAY
/* Initialize the interface */
RXMQVTRACE = ''
rcc= RXMQV(‘INIT’)
SAY 'RC=' rcc
/* Connect to Queue Manager */
RXMQVTRACE = ''
rcc = RXMQV('CONN', QMgr)
SAY 'RC=' rcc
/* Open Queue for Input */
RXMQVTRACE = ''
oo = MQOO_INPUT_SHARED
rcc = RXMQV('OPEN', QName, oo , 'h1', 'ood.' )
SAY 'RC=' rcc
/* Get a Message */
i = Ø
g.Ø      = 54 /* The message text is a job name */
g.1      = ''
igmo.opt = MQGMO_ACCEPT_TRUNCATED_MSG
RXMQVTRACE = ''
rcc = RXMQV('GET',  h1,'g.','igmd.','ogmd.','igmo.','ogmo.’)
DO WHILE WORD(rcc, 1) <> '2Ø33' /* messages stil on queue */
   i = i + 1
   SAY 'RC=' rcc
   comment = '(If the above reason code is "2Ø79 Warning", '
   comment = comment'the job name contained more than 54 characters. '
   comment = comment'Please check.) '
   SAY comment
   SAY
   SAY 'Job 'i': 'g.1
   ADDRESS TSO
   JobName = WORD(g.1, 1)       /* trim blanks */
   instr = '"SUBMIT '"'"JobName"'"'"' /* message text is a job name */
   INTERPRET instr
   /* Get next message */
   g.Ø      = 54
   g.1      = ''
   igmo.opt = MQGMO_ACCEPT_TRUNCATED_MSG
   rcc = RXMQV('GET',  h1,'g.','igmd.','ogmd.','igmo.','ogmo.’)
END /* DO WHILE WORD(rcc, 1) <> 'FAILED'*/
SAY
SAY i' messages have been read'
SAY
/* Stop access to a Queue */
```

```
RXMQVTRACE = ''
rcc = RXMQV('CLOSE', h1, mqco_none)
SAY 'RC=' rcc
/* Disconnect from the QM */
RXMQVTRACE = ''
rcc = RXMQV('DISC', )
SAY 'RC=' rcc
/* Remove the Interface functions from the Rexx Workspace ... */
RXMQVTRACE = 'TERM'
rcc = RXMQV('TERM', )
SAY 'RC=' rcc
RETURN
```

BATCH TRIGGER MONITOR (SUPPORTPAC MA12 SET-UP)

You need to compile the COBOL program and place the load module in a suitable library, then start the monitor. You can use following job:

```
//MONITOR  JOB CLASS=E,TIME=144Ø
//CKTIBAØ1 EXEC PGM=CKTIBAT2,
//         PARM='QMGR,BATCH.INITQ'
//STEPLIB  DD  DSN=MQM.SCSQAUTH,DISP=SHR
//         DD  DSN=USER.LOAD
//SYSPRINT DD  SYSOUT=*
//INTRDR   DD  SYSOUT=(*,INTRDR)
//JOBCARD  DD  DSN=MARCIN.JCL.MQS(JOBCARD),DISP=SHR
```

Tailoring the above code

- Line one: supply a valid job card. The batch trigger monitor is a long-running job so make sure it will be run in a suitable class. To avoid abending the job, set TIME to unlimited (1440 will do that).
- Line three: batch monitor parameters
  - queue manager name
  - initiation queue name (the queue is opened by the monitor for exclusive use, so make sure no other job uses it).
- Line four: MQSeries *hlq.SCSQAUTH* library.
- Line five: library where the CKTIBAT2 load module is located.
- Line six: sysout class for output.
- Line seven: JES2 internal reader.

- Line eight: a data set (sequential or PDS member) containing a job card for jobs that will be triggered by the monitor.

MQSERIES OBJECT DEFINITIONS

Implementing triggering requires three object definitions: a triggering queue, an initiation queue (both defined as local queues), and a process. In the definitions given below I show only attributes related to triggering (the rest may have default values) and I assume TRIGTYPE(FIRST) will be used (for a description of triggering types please refer to the *MQSeries System Management* manual).

All names in apostrophes are subject to change: supply your own, but be careful, because some names appear in several places and should be repeated as in the examples given below. Do not put apostrophes where I have not put them!

Triggering and initiation queue definitions are pretty simple (provided you are familiar with triggering concepts); the process definition is strange – an explanation is given later.

In the process definitions there are some blanks. They are not a mistake! What's more, there needs to be a certain number of them! I explain it below. The best way to define it is to use the MQSeries ISPF panels. To make it as simple as possible I show here the panel contents, not the commands.

The triggered job can be submitted as either a procedure or a normal job. I show process definitions for both cases (triggering and initiation queues definitions are the same). It is up to you to decide which option to use (but you may be not authorized to start procedures at your site).

- Triggering a queue definition:

```
DEFINE QLOCAL('your.trigg.queue') +
TRIGGER +
TRIGTYPE(FIRST) +
PROCESS('your.process') +
INITQ('your.initq')
```

- Process definitions (these are MQSeries ISPF panel contents, not commands):
  - Process definition for triggering a JCL procedure

```
Process name  . . . . . . . . REXXPROC.SUBMIT
Description . . . . . . . . . :
Application type  . . . . . . : MVS
Application ID  . . . . . . . : //REXXPROC EXEC REXXPROC
User data . . . . . . . . . . : //SYSTSIN  DD *
                                %TRIG
                                !
Environment data  . . . . . . :
```

The process name REXXPROC.SUBMIT is mine; supply your own here.

– Process definition for triggering a job:

```
Process name  . . . . . . . . REXXJOB.SUBMIT
Description . . . . . . . . . :
Application type  . . . . . . : MVS
Application ID  . . . . . . . : //REXX EXEC PGM=IKJEFT01
User data . . . . . . . . . . : //STEPLIB DD DISP=SHR,
                                // DSN=MQM.SCSQAUTH
                                //SYSEXEC DD DSN=MARCIN.REXX,
                                //        DISP=SHR
Environment data  . . . . . . : //SYSTSPRT  DD SYSOUT=*
                                //SYSTSIN   DD *
                                %TRIG
                                !
```

The process name REXXJOB.SUBMIT is mine; supply your own here.

– Initiation queue definition:

```
DEFINE QLOCAL('your.initq')
```

PROCESS DEFINITION EXPLAINED

The following indented text is taken from *CKTIBAT2* comments and was contributed by Wayne M Schutz.

> "The job that runs batch trigger monitor (CKTIBAT2) has a DD statement JOBCARD. It points to a data set containing the JOB card of the triggered job. Here is the JOB card I used:
>
> ```
> //TRIGER   JOB
> ```
>
> The JOB name must be exactly six characters long and have at least three blanks between the name and the JOB keyword. The batch trigger will append the job name with a two character numeric ('00' for the first triggered job, '01' for the second and so on).

Actually the batch trigger monitor will alter the ninth and tenth bytes of the JCL card. So if your JOB card is //TRG JOB the trigger monitor will change it to //TRG 00JOB and that will cause a JCL error. The best idea is to code a name of six characters and then have at least three blanks before the JOB keyword.

The process definition must contain the remaining JCL. The application identifier (ApplicId) contains the // EXEC card. It may either execute a single program or call a procedure to be executed.

Up to four additional JCL cards may be included in the stream by specifying them in the UserData field. Cards are limited to 32-bytes in length. The cards are taken out of the field at offsets 0, 32, 64, and 96.

Four additional JCL cards may be included in the stream by specifying them in the EnvironmentData field. Cards are limited to 32-bytes in length. The cards are taken out of the field at offsets 0, 32, 64, and 96.

Any of the nine JCL cards may contain a single exclamation mark (!), which will be replaced by the name of the queue causing the trigger message (MQTM-QNAME)."

TRIGGERING A PROCEDURE

Earlier, I gave the JCL needed to trigger a procedure. Since there is no place to code the JCLLIB statement (it must come before the EXEC statement) the procedure must be stored in *SYS1.PROCLIB* (or its equivalent at your site). The procedure will be started using the authority of the user that submitted the batch trigger monitor, so make sure it is an appropriate user.

JES will receive first part of the procedure JCL from the process definition. Let us see the details:

```
//REXXPROC EXEC REXXPROC
//SYSTSIN  DD *
%TRIG
QmgrName !
```

Tailoring the above code

- Line one: name of the procedure (you can use your own).

- Line two: JCL does not allow DD * statements in procedure 'body', so we must code it here.
- Line three: name of my REXX program.
- Line four: parameters for my program. QmgrName is your queue manager name. The exclamation mark (!) will be substituted by the batch trigger monitor for the name of the triggering queue. There should be at least one blank between the queue manager name and the exclamation mark. My program will accept this line as a parameter. It is required!

Beware!

In some systems (depending on national character coding) the exclamation mark can be shown as ']' (right bracket). Its hex code should be '5A'.

The body of the procedure is shown below:

```
//REXXPROC PROC
//REXX     EXEC PGM=IKJEFT01
//STEPLIB  DD DISP=SHR,DSN=MQM.SCSQAUTH
//         DD DISP=SHR,DSN=MA18.LOAD
//SYSEXEC  DD DISP=SHR,DSN=MARCIN.REXX
//SYSTSPRT DD SYSOUT=*
```

Tailoring the above code

- Line three: MQSeries *hlq.SCSQAUTH* library.
- Line four: SupportPac MA18 (REXX interface to MQSeries) load library.
- Line five: library containing my REXX code.
- Line six: a sysout class (I print some reports there).

TRIGGERING A JOB

Triggering a 'plain' job is more difficult but if you are not authorized to use procedures it is your only choice.

The whole JCL must be coded in the nine lines (APPLICID, USERDATA, ENVDATA); each 32 bytes long. If you want to span JCL statements over

several lines you must use JCL syntax.

The JCL is explained below:

```
//REXX EXEC PGM=IKJEFT01
//STEPLIB DD DISP=SHR,
// DSN=MQM.SCSQAUTH
//SYSEXEC DD DSN=MARCIN.REXX,
//         DISP=SHR
//SYSTSPRT  DD SYSOUT=*
//SYSTSIN   DD *
%TRIG
QmgrName !
```

Tailoring the above code

- Line two: the STEPLIB must contain both MQSeries *hlq.SCSQAUTH* modules and SupportPac MA18 (REXX interface to MQSeries) load module. There is no place to have separate lines for both so I decided to copy the MA18 load module to MQSeries *hlq.SCSQAUTH* library.
- Line four: library containing my REXX code.
- Line six: a sysout class (I print some reports there).
- Line eight: name of my program.
- Line nine: parameters for my program. QmgrName is your queue manager name. The exclamation mark (!) will be substituted by the batch trigger monitor for the name of the triggering queue. There should be at least one blank between the queue manager name and the exclamation mark. My program will accept this line as a parameter. It is required!

Beware!

In some systems (depending on national character coding) the exclamation mark can be shown as ‘]’ (right bracket). Its hex code should be ‘5A’.

Output

The output of the program will show all submitted job names along with some tracking information (I print the reason code after each MQSeries call).

If the reason code returned by MQGET operation is

```
'2Ø79     1        2Ø79     RXMQVGET    WARNING',
```

it means that the job name has been truncated. Please check it! I read only 54 bytes from each message (the maximum DSN name is 44 characters, eight characters for the member name, and two parentheses), so if this warning occurs it may mean that the job name was not valid. I present it as a warning, not a failure, because the truncated characters might be just blanks.

ENDING THE BATCH TRIGGER MONITOR

To end the trigger monitor do not cancel the job. Do it by sending a message of type REPORT and feedback of MQFB_QUIT (both are coded in the message descriptor, MQMD) to the initiation queue. There is a program that does just that. It is supplied within the MA12 SupportPac; its name is CKTIEND. To run it, use the following job:

```
//ENDMONIT JOB
//CKTIEND  EXEC PGM=CKTIEND,PARM='QMGR,BATCH.INITQ'
//STEPLIB  DD  DSN=MQM.SCSQAUTH,DISP=SHR
//         DD  DSN=USER.LOAD
//SYSPRINT DD  SYSOUT=*,
//             DCB=(DSORG=PS,RECFM=VBA,LRECL=133,BLKSIZE=137)
```

Tailoring the above code

- Line one: supply a valid job card
- Line two: CKTIEND parameters (both must be the same as you supplied for CKTIBAT2)
  - queue manager name
  - initiation queue name.
- Line three: MQSeries *hlq.SCSQAUTH* library.
- Line four: library where CKTIEND is located.
- Line five: sysout class for output.

SOME NOTES ON MY CODE

My program connects to a queue manager (a user-supplied parameter) and

opens a queue that caused the trigger to happen (its name is passed from the trigger monitor) and reads messages (in get mode) from it. Each message should be a valid job name (in the library member) format that will be submitted. I do no checking of the job name validity; the check is left to the TSO command **SUBMIT**.

SECURITY

The following authorizations are required:

- The batch trigger monitor needs authority to open the initiation queue for input (it does so in exclusive mode).
- My program needs authority to open the triggering queue for input (shared); it also needs authority to use the TSO command **SUBMIT**.

Marcin Grabinski
System Engineer, SPIN (Poland)

Global transactions with MQSeries and Oracle, part 2: set-up and configuration

This is the second (and concluding) article on using MQSeries and Oracle in global transactions. MQSeries and Oracle may be configured to act as components in the X/Open Distributed Transaction Processing (DTP) model. This article explains how to set up MQSeries to work as a Transaction Manager (TM) and as a Resource Manager (RM), and how to set up Oracle to work as an RM so that the atomicity, consistency, isolation, and durability (ACID) properties of the global transactions are preserved. Various configuration scenarios are explained that include multiple databases, remote databases, dynamic Oracle registration, tracing facilities, and multi-threaded applications.

ORACLE XA SWITCH LOAD FILE

While the XA communication between the MQSeries TM and MQSeries RM is performed internally in the DTP model, the Oracle RM runs as a

separate entity that may even reside on a separate machine from MQSeries. The MQSeries TM explicitly invokes XA functions on the Oracle RM.

Before the user application (AP) starts the first global transaction, the MQSeries TM must be provided with the information that instructs it how to invoke the XA functions on the Oracle RM. Providing the MQSeries TM with this information is a part of the DTP setup process and is implemented by the SLF (Switch Load File). The SLF is implemented as a dynamic shared library (*.dll* on WinNT and *.so* or *.sl* on Unix platforms).

In order for the Oracle RM to participate in a global transaction managed by the MQSeries TM it needs to have the SLF associated with it. The Oracle SLF for MQSeries exports a single function called MQStart(). MQStart() returns the address of a special structure implemented in the Oracle RM libraries that conforms to the xa_switch_t type defined in the *xa.h* header file. The *xa.h* file is part of the XA specification. This data structure contains pointers to the XA functions implemented in the Oracle RM. Once the MQSeries TM loads the Oracle SLF it gains access to the XA functions in the Oracle RM through the switch structure.

Building the Oracle SLF is a straightforward procedure and MQSeries Server installation comes with the files necessary to do it. These files, which are located in the *xatm* subdirectory of the *samp* directory in the MQSeries installation, are:

- *xa.h* – the XA header file that defines the xa_switch_t structure type. The declarations of all XA functions can be found in this structure.
- *oraswit.c* – the C file that is used to build the SLF. This C file defines and exports the MQStart() function that returns a pointer to the Oracle xa_switch_t structure implementation. The actual name of the Oracle implementation of the xa_switch_t structure is xaosw. It is implemented in the Oracle client library.
- *xaswit.mak* – makefile for the SLF.
- *oraswit.def* – (WinNT/2000 only) a definition file for the SLF.

MQSeries documentation and comments in the makefiles provide enough information to build the Oracle SLF for a particular platform. Note that the makefiles contain hard-coded paths to the Oracle XA library. This value needs to be checked and manually modified to match the underlying Oracle

installation. Since the Oracle xa_switch_t structure is implemented in the Oracle client library the most important thing to ensure is that the SLF contains a valid path reference to this library so that the address of the xaosw structure may be resolved when the MQSeries TM loads the SLF and invokes the MQStart() function.

XARESOURCEMANAGER STANZA FOR ORACLE

Once the Oracle SLF is built it is necessary to inform the MQSeries TM of the location of the file so that the Oracle RM may become a participant in the global transactions.

The location of the Oracle SLF, as well as some other relevant information about the Oracle RM, is stored in a special stanza in the *qm.ini* file of the queue manager selected to serve as a TM. This stanza is called the XAResourceManager stanza.

On WinNT/2000 this stanza is implicitly specified to the queue manager through the MQSeries Services GUI by selecting the Resources tab in the Properties dialogue of the queue manager. The information entered in this dialogue is stored in the Windows Registry. On Unix systems it is necessary to edit the *qm.ini* file manually and explicitly add the XAResourceManager stanzas.

Every RM that participates in global transactions requires its own XAResourceManager stanza. The format of the XAResourceManager stanza is:

XAResourceManager:

Name=*RMName*

SwitchFile=*FullPathToTheSwitchLoadFile*

XAOpenString=*XAOpenString*

XACloseString=*XACloseString*

ThreadOfControl=*MQSThreadOfControl*.

Where:

- *RMName* is a name that the user selects for the RM. The MQSeries

TM uses this name when it creates the entries that are related to the particular RM in the error log files.

- *FullPathToTheSwitchLoadFile* is a fully qualified name of the SLF. For an Oracle RM this is the path to the Oracle SLF. The MQSeries TM uses the value of this parameter to obtain the addresses of the XA functions implemented by the Oracle RM.
- *XAOpenString* is a string that the MQSeries TM passes to the RM in the xa_open() function call. The format of this string is defined by the RM vendor. The complete Oracle XA open string format is described in the MQSeries documentation as well as in the Oracle documentation. A simple example of the Oracle XA open string is:

```
Oracle_XA+Acc=P/scott/tiger+SesTm=35
```

The components of the Oracle XA open string are always separated with the '+' character. In the example above, the following components can be distinguished:

- *Oracle_XA* – an Oracle XA open string always starts with this identifying token.
- *Acc=P/username/password* is a mandatory Oracle XA open string component that specifies the Oracle user-ID and password that the MQSeries TM uses to connect to the RM in the xa_open() call. In this example the user-ID is 'scott' and the password is 'tiger'.
- *SesTm=* is another mandatory component that defines the maximum amount of time in seconds for which a transaction can be inactive before it gets automatically destroyed and rolled back. In this example a time limit of 35 seconds is specified.
- *XACloseString* is a string that the MQSeries TM passes to the RM in the xa_close() function call. For the Oracle RM this value should always be set to:

```
This_Is_Close_String
```

The Oracle RM does not require the XA close string and it always treats it as if it were NULL. But the XACloseString still needs to be specified in the XAResourceManager stanza for Oracle in order for the MQSeries TM to issue the xa_close() calls on the Oracle RM.

If the XACloseString is set to some other value than the one described above or if it is not set at all, the MQSeries TM will not call xa_close() on the Oracle RM and will report an error in the log file that will indicate the failure of the xa_close() call.

- MQSThreadOfControl defines the TOC for the MQSeries. It can be set to PROCESS or THREAD. The default value is PROCESS. This parameter is described later, in the section on multi-threading.

DBA_PENDING_TRANSACTIONS VIEW

An Oracle database contains in the sys schema a view called DBA_PENDING_TRANSACTIONS, which shows the transaction identifiers for the pending Oracle transactions. This view is based on the system tables *gv$global_transaction*, s*ys.pending_trans$*, and *sys.pending_sessions$*.

The pending transactions are the transactions that are prepared but are still not committed or rolled back, in addition to the transactions that were heuristically committed or rolled back. The MQSeries TM must obtain the list of pending transactions in all RMs every time it performs the recovery procedure. These lists help the MQSeries TM to recognize the state of a particular RM regarding the previously interrupted global transactions.

Every time the MQSeries TM (the queue manager) starts, it calls the xa_recover() function on the Oracle RM. The Oracle implementation of this function is called xaorecover(). xaorecover() internally calls the Oracle function xaofetch(), which accesses the DBA_PENDING_TRANSACTIONS view to collect the identifiers of the pending transactions. If xaofetch() fails to access this view for security reasons, xaorecover() returns with the error code XAER_RMFAIL [-3] and this prevents the successful completion of the recovery process. It is, therefore, extremely important to grant the SELECT privileges on this view to the user that was specified in the Oracle XA open string.

The system administrator may issue the following SQL statement to grant the required select privileges to the user with the username scott:

```
grant select on DBA_PENDING_TRANSACTIONS to scott;
```

DYNAMIC ORACLE RM REGISTRATION ON WinNT/2000

In addition to the previously described static Oracle SLF, it is possible to build a dynamic SLF for Oracle on the WinNT/2000 platform. WinNT/2000 installation of MQSeries comes with two additional files in the *xatm* subdirectory. These are *oraswitd.c* and *oraswitd.def* and they are used instead of *oraswit.c* and *oraswit.def* to build the dynamic SLF.

Like the static SLF, the dynamic SLF exports the single MQStart() function that returns a pointer to an internal instance of the xa_switch_t structure. The difference in the latter instance is that a TMREGISTER flag is set in the flags member of the structure. This informs the TM not to call xa_start() on this RM when a new global transaction starts but to wait for the RM to register itself in the transaction through the xa_reg() call that the TM must implement. The internal Oracle implementation (variable) of the xa_switch_t type that is returned by the MQStart() function in the dynamic SLF is called xaoswd.

The difference between the default static SLF and the dynamic SLF is in when and how the Oracle RM becomes a participant in a global transaction. In the static SLF case, when the AP starts a new global transaction through the MQBEGIN() call, the MQSeries TM immediately calls xa_start() on the Oracle RM to inform it of the start of the new global transaction. Conversely, when the dynamic SLF is used, the MQSeries TM never calls xa_start() on the Oracle RM. It is only when the AP calls the Oracle RM to perform a data operation (for example, insert a record in a table with the OCIStmtExecute() call) that the Oracle RM contacts the TM to see whether the AP previously started a global transaction in the same TOC. It needs to know this in order to decide whether the requested data operation should be made part of a global transaction.

The Oracle RM does this by invoking the ax_reg() function on the MQSeries TM. The Oracle RM deregisters itself later from the global transaction by invoking the ax_unreg() call on the MQSeries TM. The ax_reg() and ax_unreg() functions are XA functions that the MQSeries TM on WinNT/2000 implements in the *mqmax.dll* dynamic library.

When the MQSeries TM returns the confirming answer when called on ax_reg(), it provides the Oracle RM with the XID of the previously started global transaction. The Oracle data operation is then made part of the global transaction and is committed only when the AP later invokes MQCMIT()

on the MQSeries TM.

The MQSeries TM is provided with the addresses of the Oracle RM XA functions through the SwitchFile parameter in the XAResourceManager stanza. In this case, it is also necessary to provide the Oracle RM with the name of the DLL that implements the ax_reg() and ax_unreg() XA functions called by the Oracle RM. This is done through the ORA_XA_REG_DLL environment variable. The value of this variable must represent the fully qualified name of the *mqmax.dll* library in the local MQSeries installation. For example:

```
set ORA_XA_REG_DLL=c:\Program Files\MQSeries\bin\mqmax.dll
```

Note that, if this environment variable is not set correctly, the program may still seem to work although the ax_reg() function will not be called. The Oracle RM will not commit its changes during the MQCMIT() call, but only later, when the MQSeries calls it on xa_close(). The Oracle changes will not be committed during the MQCMIT() call because the Oracle RM was never associated with the global transaction.

The MQSeries TM calls xa_close() on the Oracle RM when the AP calls MQDISC(). This must be taken into consideration because, if the Oracle RM fails to locate the ax_reg() function, it will not report an error, and if the system crashes between the MQCMIT() and MQDISC() calls the atomicity of the global transaction will not be preserved, since the MQSeries operations will be committed and the Oracle operations will be implicitly rolled back.

Tracing XA/TX function calls

To trace the XA calls that the MQSeries TM makes on the RMs, as well as the TX calls that the AP makes to the MQSeries TM, it is enough to turn on the regular MQSeries trace for the used queue manager. All the operations that the TM performs are traced, from loading the SLFs to closing the RMs.

To trace the XA calls made to the Oracle RM it is necessary to add an optional parameter to the Oracle XA open string. This parameter is called LogDir and its value defines the directory path where the XA trace information is stored. For example, to define the *c:\xalogs* directory on WinNT/2000 as a directory for the Oracle XA trace files the following line

needs to be appended to the existing Oracle XA Open String:

```
+LogDir=c:\xalogs
```

By default, the Oracle RM traces only the XA errors that occur. It is possible to require additional information about the XA calls execution through the DbgFl optional parameter in the Oracle XA open string. The following hexadecimal values can be combined to form this parameter value.

- *0x01* – trace entry and exit points for every XA call made to the Oracle RM.
- *0x02* – trace entry and exit points for the internal supplementary XA functions invoked from within the XA functions (these functions also have the xao prefix, just like regular Oracle XA functions).
- *0x04* – trace entry and exit points for the additional functions called from within the XA functions (such as various OCI function calls).

For example, to specify the maximum trace level (which is very helpful for tracing problems) the following line needs to be appended to the existing Oracle XA open string:

```
+DbgFl=0x07
```

LOCAL AND REMOTE ORACLE DATABASES

The MQSeries queue manager and the AP must run on the same machine but the Oracle database can run on a separate machine in the network. Of course, the machine with the MQSeries queue manager and the AP must have Oracle client facilities installed. The Oracle client library serves as a proxy to the Oracle RM. The MQSeries TM treats the Oracle client library as an XA-compliant RM.

Regardless of the location of the Oracle database, the ORACLE_HOME environment variable on the machine with the queue manager and the AP needs to point to the directory where the Oracle product (client or server) is installed.

If the Oracle database resides on the same machine as the MQSeries queue manager and the AP (local database), it is possible to set the ORACLE_SID environment variable to point to the local Oracle service name that needs to be used. But if the Oracle database resides on another machine in the

network (remote database) it is necessary to specify the service name of the remote database in the Oracle XA open string. The name of the parameter used for this is 'SqlNe' and it needs to contain the service name specified in the Net8 string format (formerly called SqlNet string format).

For example, to specify the remote database BankDB the following string needs to be appended to the existing Oracle XA open string:

```
+SqlNet=BankDB
```

MULTIPLE ORACLE DATABASES

It is possible to use more than one Oracle database in a global transaction managed by the MQSeries TM. Every database is represented as a separate RM and must be assigned a unique name in its XAResourceManager stanza.

In cases where there are multiple Oracle XAResourceManager stanzas it is necessary to use the DB parameter in the Oracle XA open string. The value of this string needs to be unique for every Oracle RM. This value is then passed to the xaoEnv() and xaoSvcCtx() calls to obtain the environment handle and the service context handle for the particular Oracle XA connection. The environment handle will then be common for all the RM instances but the service context handles will be different. For example, to specify an Oracle database with the user-assigned name 'myDB' the following line needs to be appended to the Oracle XA Open String:

```
+DB=myDB1
```

When a single Oracle database is used the DB parameter is not needed in the XA Open String and the xaoEnv() and xaoSvcCtx() function should be called with the NULL argument.

When multiple Oracle databases are used, for every one used the corresponding DB parameter value must be passed to the xaoEnv() and xaoSvcCtx() functions to obtain the correct handles for that database.

Note that the DB value does not need to be the same as the corresponding database service name (SID). It is only necessary to ensure that the DB value is unique and that it matches the passed xaoEnv() and xaoSvcCtx() arguments.

The parameter in the XAResourceManager stanza that defines the TOC for the MQSeries TM is called ThreadOfControl, and its value can be PROCESS (the default value) or THREAD.

The parameter that defines the TOC for the Oracle RM is called Threads and it is specified in the Oracle XA open string. Its value can be true or false (the default value). If the value is true the TOC is THREAD and if the value is false the TOC is PROCESS.

If the AP runs in a single thread the default TOC interpretation works fine. All the global transaction operations – its start, the data operations on the RMs, and its end – are always performed in the same process (since they are called from the same application instance).

If a global transaction is started in one thread and the Oracle RM data operations are performed in another thread, the default TOC configuration will still work. Since both threads belong to the same process and the TOC is set to be a process the Oracle RM is able to associate the data operations performed in one thread with the global transaction that was started from another thread (same process ID).

Note, however, that it is never possible to start a global transaction in one thread and perform the MQSeries operations in another because of the MQSeries implementation that prohibits sharing the queue manager connection handle among threads. That means that the MQBEGIN(), MQGET(), MQPUT(), MQCMIT(), and MQDISC() calls must all be made from the same thread from which the MQCONN() call was made.

However, it is often required to implement multi-threaded applications to increase the data throughput. Such a multi-threaded AP would have multiple threads running concurrently where every thread would run global transactions on its own.

This is not possible to do with the default TOC configuration. If the first thread starts a global transaction with MQBEGIN() and the second thread independently tries to start another global transaction while the first one is still active, the second MQBEGIN() call fails with the error reason code MQRC_UOW_IN_PROGRESS. The MQSeries TM informs the AP that another transaction (unit of work) is active within the same TOC.

To achieve concurrency with global transactions using the default TOC

configuration it would be necessary to run multiple application instances at the same time (multiple processes).

To enable multiple threads to perform global transactions concurrently and independently of each other, the TOC needs to be set to THREAD, for both the MQSeries TM and the Oracle RM.

To set the TOC to THREAD for the MQSeries TM the following line needs to be specified in the XAResourceManager stanza:

```
ThreadOfControl=THREAD
```

To set the TOC to THREAD for the Oracle RM the following line needs to be appended to the existing Oracle XA open string:

```
+Threads=True
```

It is extremely important to match these two parameters. If ThreadOfControl is set to THREAD, the threads should be set to true; and if ThreadOfControl is set to PROCESS, the threads should be set to false. This applies to WinNT/2000 as well as Unix platforms.

Predrag Maksimovic
Software Engineer
2d3D Inc (USA)

Configuring MQSeries with Microsoft Cluster Server, part 2: installation and configuration

Last month we discussed the groundwork required to enable MQSeries and Microsoft Cluster Server (MSCS) to work together. In this concluding article we look at the installation and configuration stages.

INSTALLATION OF MSCS AND MQSERIES

MSCS must already be installed and configured, including testing of cluster operations such as the movement of the cluster group from one node to another. When creating the cluster decide whether you wish to make the cluster nodes the sole members of a 'domainlet' for which the

cluster nodes themselves are the domain controllers or directory servers. This approach avoids creating a dependency on a potential single point of failure outside the cluster and can also offer best performance. User groups and ids can be created at the domain level and included in the local groups on each of the cluster nodes. It can be useful to create a domain level user account which not only owns the cluster but is also an administrator on each of the cluster nodes.

MQSeries must also have been installed on both cluster nodes. It is recommended that you install MQSeries (or other product code) onto internal disks on each of the nodes rather than onto shared disks. The shared disks should be used only for the state data required by the cluster and highly-available applications. The location of the necessary state data for MQSeries queue managers is discussed later in this article in *Configuration: step four*.

It is important that under normal operating conditions you are running identical versions of the operating system, including MSCS and MQSeries, on each of the cluster nodes.

Configuration of shared disks

Configuration of shared disks can be performed before MC74 is installed. The shared disk software must have been installed and configured as part of installing MSCS and migrating the quorum to shared disk. It is also possible – and a good idea – to configure the remaining shared disks into disk arrays and logical drives ready to support the applications and services that will be run in the MSCS cluster.

MSCS insists that the quorum drive is configured in a RAID-1 disk array, which means that it will use two physical disks. You can divide the remaining disks into arrays that contain the logical drives needed for queue manager or application data. You are free to choose the RAID level you want for these drives.

User/group principals

The installation of MQSeries creates various user-ids on each of the cluster nodes, such as the MUSR_MQADMIN user account and the MQM group. These are local ids and, therefore, have different SIDs on each node. Don't

worry about this at this stage. Just install MQSeries as if the nodes were not part of an MSCS cluster.

Installing MQSeries MSCS support

Log into the cluster nodes as a user with sufficient permission to alter the MQSeries and MSCS configuration and download **the mc74.exe** from the IBM SupportPac Web site into a directory of your choice.

Run the **mc74.exe**, which is a self-extracting file built with InstallShield.

The install program will register a new resource type called 'IBM MQSeries MSCS'.

Verifying installation

When the install program has finished open the Cluster Administrator and click on Resource Types to check that the new resource type has been added.

CONFIGURING MQSERIES MSCS SUPPORT

Before performing any of the steps in this section ensure that you have completed all the steps in the previous section on installation and that the MSCS cluster is working as expected.

The following steps are described from the point of view of one queue manager. For additional queue managers just repeat them.

Step one: create a resource group

A resource group is used as a container for all the resources related to the queue manager. The resource group will contain the shared drives used by the queue manager, the IP address which clients and channels will use to connect to the queue manager, and the queue manager resource itself.

It is recommended that you consider disabling the failback option on the resource group to avoid the interruption to the queue manager that would be caused by restart of the default master node after a failure. Unless you have a specific requirement that would make failback desirable in your configuration, it is probably better manually to move the resource group back to the preferred node, where it will cause minimum disruption.

Actions

1 Create a resource group and ensure that failback is disabled.

2 Create an IP address resource (using the MSCS-provided resource type) and include it in the group. The shared drives will be added in the next step.

3 Test the operation of the resource group, ensuring that the IP address is configured correctly onto whichever node hosts the group.

Step two: configure the shared drives

As stated previously, it is recommended that each queue manager is placed in a separate resource group and has an exclusive drive or set of drives, which are defined as resources in the same group. This allows each queue manager to be moved from one node to another without disrupting any other queue managers.

You will need only one drive per queue manager if you plan to keep its logs and queue files on the same drive. For performance it is recommended that a queue manager uses separate drives for logs and data, so you may want to separate them, in which case, create two drives per queue manager. At the physical level each logical drive can use a single physical disk, a mirrored pair of disks, or a more sophisticated array of disks. You should decide the degree of redundancy and fault protection that you require.

Actions

1 If you have not already done so you now need to create the drives for the queue manager. The specific details of this step will vary depending on the hardware and shared disk support that you are using. Follow the documentation that came with your shared disk support to create the necessary drives. The drive letters that you assign are arbitrary but they must be the same across the nodes.

2 Create MSCS resources that manage the shared drives, adding them to the group you created in step one. The resource type to use will depend on the shared disk support that you are using. If, for example, you are using IBM ServeRAID, the installation of the ServeRAID cluster kit will have created a resource type called 'IBM ServeRAID Logical Disk'.

3 Make sure that the drives can be moved from one node to another and are visible and accessible on each node. Test this by moving the resource group from one node to another. Check that they are visible and accessible on each node.

Step three: create the queue manager

Select a node on which to create the queue manager.

Use MQSeries Explorer to create the queue manager in the normal manner, mostly accepting the suggested default settings. This will create the queue manager such that its logs and queue files are stored on drives which are internal to the node, rather than on shared drives. Step five will move the files onto shared drives and allows you to perform the same for an existing queue manager that you hadn't anticipated would be put under MSCS control.

You do not need to create any other instances of the queue manager – just create it once on one node.

Actions

1 Select a node on which to perform the actions detailed below.

2 Run MQSeries Explorer and create a new queue manager, accepting all the defaults, but don't configure it to be the default queue manager.

3 When the queue manager has been created, ensure that it works, ie start and stop it. Create any queues and channels and test the queue manager.

4 Set the queue manager's service startup property to manual – this is important because you don't want the IBM MQSeries service to try to restart it on failure, you want MSCS to do it.

5 Leave the queue manager in the stopped state.

Step four: prepare the queue manager for MSCS

A queue manager that is to be used in an MSCS Cluster needs to have its logs and data on shared disks so they can be accessed by a surviving node following a node failure.

The queue manager is not set up this way currently. Whether it's a new

queue manager that you have just created or an older one that you are now preparing to put under MSCS control, you need to move its queue files and log files to one or two shared drives. You can use the **hamvmqm** command to do this. This is something you need to do only once, when the queue manager is first placed under MSCS control. During normal operation the queue manager will be able to move back and forth between the nodes because the files it needs will be on the shared drives, which will be made available by MSCS to whichever node has the queue manager.

When, one day, you decide to remove the queue manager from MSCS control, you may want to run **hamvmqm** once more to move the files back from shared drives to internal drives within one of the nodes. This is explained in step seven.

Actions

1 On the node on which the queue manager exists perform the actions detailed below.

2 Use the **hamvmqm** command to move the queue manager's data and log files to shared drives.

- **hamvmqm**

  The **hamvmqm** command allows you to specify where you want a queue manager's data and log files to be stored. By default these files will be stored in M*QSeries\qmgrs\<qmgr name>* and *MQSeries\log\<qmgr name>*. T**he hamvmqm** command determines from the registry the current locations of the queue manager's files. It moves the files to the locations that you specify and updates the registry to reflect the changes. The command accepts each of the new paths as parameters. If you want to, you can use the same path for both data files and logs.

  You can use the **hamvmqm** command to move files from their current locations to any paths you choose, but it is most useful in moving files from internal drives to shared drives or from shared drives to internal drives.

  The syntax is:

```
hamvmqm /m <qmgr  name> /dd <newdatapath> /ld <newlogpath>
```

The parameters are:

- *qmgr name* – the name of the queue manager to which the command relates.
- *newdatapath* – the pathname of the directory to which you want the queue manager's data files to be moved. The path you specify for the */dd* option will be suffixed with *qmgrs\<qmgr name>*. See the example below.
- *newlogpath* – the pathname of the directory to which you want the log files to be moved. The path you specify for the */ld* option will be suffixed with *<qmgr name>*. See the example below.

Take care with the parameters to this command because it is based on the system of prefixes and directory structures that are typical in MQSeries. For example:

```
hamvmqm /m QM1 /dd j:\mqha /ld k:\mqha\log
```

will move the queue files for QM1 to *j:\mqha\qmgrs\QM1* and the log files for QM1 to *k:\mqha\log\QM1*.

Step five: create a resource for the queue manager

For MSCS to monitor and control the queue manager you need to create a resource of type IBM MQSeries MSCS. This is the custom resource type provided in the MC74 SupportPac. When you create the resource you will need to tell it the queue manager's name.

All MSCS resources have a 'restart threshold' parameter, which is the number of times MSCS will attempt locally to restart the resource within a time period defined by the associated 'restart interval' parameter. For MQSeries it is recommended that the threshold is set to 1 so that one restart is attempted, and that the restart interval is set to a small multiple of the expected start time for the queue manager. With these settings, if successive restarts fail without a significant period of stability in between, the resource group containing the queue manager will be failed over to a different node. Attempting more restarts on a node on which a restart has just failed is unlikely to succeed.

The IBM MQSeries MSCS resource type allows you the option of specifying the name of a program to be invoked just after the queue manager is brought online or just before it is taken offline. These events are referred to as 'postonline' and 'preoffline' respectively. This is to allow you to send a notification to an application, a monitoring system, or a human administrator.

Don't use the postonline or preoffline commands to start or stop a program that uses the queue manager because there is a better way of doing this. Consider using the generic application resource type to create a resource for the program you wish to start and make it depend on the queue manager. MSCS will then take care of sequencing the starting and stopping of the application relative to the starting and stopping of the queue manager. This is a better way of running such an application because if the queue manager or any resource on which it depends experiences a failure the application will be stopped and restarted by MSCS, locally or on the other node.

For the queue manager to run identically on both nodes MSCS keeps track of the registry settings on whichever node the queue manager is on and logs any changes to its quorum drive. When the queue manager is started on a node MSCS replays the log into the registry of that node. This is known as registry checkpointing. The custom resource type adds the checkpoints for the registry subtrees used by the queue manager. The custom resource type also allows you to configure registry checkpointing of custom services, such as the publish/subscribe broker from the MA0C SupportPac.

Actions

1 Use the Cluster Administrator to locate the IBM MQSeries MSCS resource type. Right-click it and press New.

2 The first few dialogue panels that appear are for the common properties of the resource and these include the group name, the nodes on which the resource can run, and whether it has any dependencies. Set these so that the resource is in the group you created in step one and make it depend on both the IP address resource (created in step one) and any shared drives onto which you have moved queue manager files (in step four).

3 You will then see the parameters dialogue panel – you can mostly

accept the defaults this offers as these parameters can easily be altered later from the resource properties dialogue. One parameter you should alter now is the restart threshold, which for MQSeries is best set so that it is limited to one at most.

4 The last dialogue panel during resource creation is for the private properties – these are the ones specific to this resource type. You must enter the name of the queue manager and optionally, enter the names of preoffline and/or postonline programs. On this panel you can also add registry checkpointing for custom services by clicking on Advanced.

5 When you have completed the above dialogue panels click on Finish to create the resource.

6 By right-clicking the resource, bring the queue manager resource online and check that it starts the queue manager successfully. You can see whether it starts by having MQSeries Explorer running at the same time as the Cluster Administrator.

7 By right-clicking the resource, take the queue manager resource offline and check that it stops the queue manager successfully.

Step six: test the queue manager

You have now completed all the steps needed to put an MQSeries queue manager under MSCS control. It is recommended that you thoroughly test the operation of the queue manager and ensure that the cluster configuration behaves as you require.

You can test movement from one node to another simply by using the Move Group menu item from within Cluster Administrator. You can introduce failures by:

- Unexpectedly shutting down a node.
- Issuing an **endmqm -i** command from a command line.
- Killing queue manager processes from the Task Manager.
- Using the 'Initiate Failure' popup menu item from within the Cluster Administrator.

Step seven: removing the queue manager from the cluster

Should you decide to remove the queue manager from cluster control you will need to take the queue manager resource offline first. The resource can then be deleted from the resource group. This does not destroy the queue manager, which should continue to function normally, but under manual control.

Once the queue manager has been removed from the cluster configuration it will not be highly available and will remain on one node. The queue manager's log and data filesystems will at this time still be in the shared drives and will still be configured to use the movable IP address, so you need to ensure that the queue manager's node retains access to the drives and IP address, or you will need to reconfigure the queue manager to remove these dependencies, as explained below.

If you have no further use for it you could delete the resource group.

Other nodes will still remember the queue manager (they will have registry entries for it) and you may wish to tidy up the other nodes. You can do this by running the **hadltmqm** command, described below, on the standby nodes – do not run it on the node with the queue manager unless you want to delete the queue manager.

- **hadltmqm**

  The **hadltmqm** command will delete the registry entries associated with a queue manager.

  Run this command only on nodes that were previously used as stand-by nodes. Do not run it on the node that is currently hosting the queue manager.

  The syntax is:

  ```
  hadltmqm <qmgr name>
  ```

  The parameter is:

  qmgr name – the name of the queue manager to be deleted.

Actions

1 Choose which node you want the queue manager to end up on and

move the resource group to that node.

2 Take the queue manager resource offline. This will stop the real queue manager.

3 Delete the queue manager resource. This does not destroy the real queue manager.

4 If you wish to keep the queue manager but not run it under cluster control you should ensure that its logs and data files stay on the same node as the queue manager. You can choose one of the following options:

- Move the queue manager's logs and data files from shared drives to internal drives, using the **hamvmqm** command described earlier, passing it the paths on the internal drives to which you want them moved. Or:
- Leave the queue manager's logs and data files on shared drives, but remove those drives from cluster control by deleting the corresponding resources from the resource group.

5 If you wish to keep the queue manager but not run it under cluster control, you also need to ensure that the IP address it uses stays on the same node as the queue manager. You can choose one of the following options:

- Reconfigure the queue manager and its clients and peers so that the queue manager uses a fixed IP address. This will release the movable IP address so that you can use it for another purpose. Or:
- Allow the queue manager to continue to use the configured IP address, but remove the IP address from cluster control by deleting the corresponding resource from the resource group.

6 To remove the registry entries for the queue manager from the standby nodes use the **hadltmqm** command. Do not run this command on the node with the queue manager – see step eight for details of how to delete the queue manager.

Step eight: deleting the queue manager

If you decide to delete the queue manager you should first remove it from

the cluster configuration, as described in step seven. Then, to delete the queue manager, perform the actions detailed below.

Actions

1 On the node that has the queue manager on it make sure the queue manager is stopped by issuing the **endmqm** command.

2 Use MQSeries Explorer to delete the queue manager.

The queue manager has now been completely removed from the cluster and the nodes.

Graham Wallis
Software Engineer
IBM Hursley (UK)

Enhancing MQSeries set authorities scripts

More often than not, scripts to set MQSeries authorities are simpy a list of **setmqaut** commands requiring the authorization values to be set. While this approach does do the job of setting the authorities, it fails to provide adequate feedback. Even if the output is captured to a file the success messages returned from **setmqaut** fail to distinguish one object from another, so are of little use. In addition, simply setting the desired authorizations may not be the best approach. It may be necessary to reset the existing authorizations, and on a V5.2 queue manager the security cache must be refreshed. This article will present techniques that can be used to resolve all of these issues and more, using standard Unix shell scripting tools.

A BASE-LEVEL AUTHORITIES SCRIPT

IBM MQSeries SupportPac MS63 provides a tool to save the authorities of a V5.1 or earlier queue manager. The output is simply a list of **setmqaut** commands necessary to recreate the existing authorities. To make this script useful most administrators will add comments, the date command, and a pointer to the Unix shell of their choice. Many also like to add the

dspmqaut command to provide visual confirmation of the object settings. An example is given below (listing one: the reference script). For brevity, all scripts in this article will be limited to a single queue.

LISTING ONE: REFERENCE SCRIPT

```
#!/bin/ksh
# Set Authorities for QM1
# Generated February 1, 2002
date
setmqaut -m QM1 -n SYSTEM.DEFAULT.LOCAL.QUEUE -t queue -g grp +inq
dspmqaut -m QM1 -n SYSTEM.DEFAULT.LOCAL.QUEUE -t queue -g grp
# Done
```

CAPTURING STDERR AND SCRIPT COMMENTS

If the reference script is run with its STDOUT redirected to a file, the result will contain the output of the date command and the **dspmqaut** command. The comments are ignored and the output of the **setmqaut** command is sent to STDERR instead of to the text file. We could rely on the operator to construct the command line to capture both STDOUT and STDERR but it would be much more reliable simply to add it to the script. This is set up in the second line of listing two below. The << EOF tells the shell to read its standard input from the remaining script lines.

The other display feature to add is to print the comments from the script body itself. By their very nature these scripts are run only when the queue manager configuration is being changed or restored – the most likely time for errors to occur.

In the case of a problem occurring after running the script, capturing the script comments is a great diagnostic tool. Again, this could be done at the command line but is better placed inside the script. Setting this at the command line would cause the contents of any *.profile* or *.kshrc* scripts to be displayed, which would be confusing rather than helpful. Instead, the **set** command is used in the third line of listing two below to turn on the shell's verbose mode.

LISTING TWO: CAPTURING STDERR AND COMMENTS

```
#!/bin/ksh
/bin/ksh 2>&1 << EOF
```

```
set -v
# Set Authorities for QM1
# Generated February 1, 2002
date
setmqaut -m QM1 -n SYSTEM.DEFAULT.LOCAL.QUEUE -t queue -g grp +inq
dspmqaut -m QM1 -n SYSTEM.DEFAULT.LOCAL.QUEUE -t queue -g grp
# Done
```

ACTIVATE AND BACK-UP THE NEW CONFIGURATION

One of the most frequent errors made when setting authorities is forgetting to activate them by recycling the queue manager, or, in the case of V5.2, issuing the refresh security **mqsc** command. In an environment where there are V5.2 queue managers, adding the command to the script will ensure that this important task is completed. This is demonstrated near the end of listing three below.

If linear logging is in use an extra measure of security may be obtained by running the **rcdmqimg** command. MQSeries will record a media image of a queue when it empties or when the queue manager is quiesced. Since the *SYSTEM.AUTH.DATA.QUEUE* should never be empty, the only way to know for sure that its contents have been saved is to run the **rcdmqimg** command against it. An example of this also is near the end of listing three below.

Note that if the refresh security or **rcdmqimg** commands are run in environments where they do not apply they will generate warning messages but will not cause any harm. It may be desirable to include the commands in all authorities scripts for the sake of standardization.

On the other hand, if the scripts are automated in a scheduling tool, the warning messages may be interpreted as failures. In general, it is better to include the commands in an environment where V5.2 queue managers exist.

LISTING THREE: ACTIVATING AND SAVING SECURITY UPDATES

```
#!/bin/ksh
/bin/ksh 2>&1 << EOF
set -v
# Set Authorities for QM1
# Generated February 1, 2002
```

```
date
setmqaut -m QM1 -n SYSTEM.DEFAULT.LOCAL.QUEUE -t queue -g grp +inq
dspmqaut -m QM1 -n SYSTEM.DEFAULT.LOCAL.QUEUE -t queue -g grp
echo 'refresh security' | runmqsc
rcdmqimg -m QM1 -t queue SYSTEM.AUTH.DATA.QUEUE
# Done
```

TIGHTENING SECURITY

The **setmqaut** command as used so far will add authorizations for a specific group and object but does not reset any existing settings. When recreating a queue manager from scratch this method will restore the authorities adequately. However, this may not work as expected if used to update an existing configuration. For example, suppose that group 'grp' had already been granted '+all' permissions against the *SYSTEM.DEFAULT.LOCAL.QUEUE*. Running the script from listing three would not remove any of the unwanted permissions.

To make sure that the object authorizations match the permissions granted in the script, place '-all' in the command line as the first authorization parameter. This will remove all existing authorizations for group 'grp' and then add inquiry rights. Make sure that '-all' is the first parameter and not simply added to the end or 'grp' will have no authorizations. The completed script is provided as listing four below.

LISTING FOUR: ENHANCED SECURITY

```
#!/bin/ksh
/bin/ksh 2>&1 << EOF
set -v
# Set Authorities for QM1
# Generated February 1, 2002
date
setmqaut -m QM1 -n SYSTEM.DEFAULT.LOCAL.QUEUE -t queue -g grp -all +inq
dspmqaut -m QM1 -n SYSTEM.DEFAULT.LOCAL.QUEUE -t queue -g grp
echo 'refresh security' | runmqsc
rcdmqimg -m QM1 -t queue SYSTEM.AUTH.DATA.QUEUE
# Done
```

SAMPLE OUTPUT OF LISTING FOUR

```
# Set Authorities for QM1
# Generated February 1, 2002
```

```
date
Tue Feb 19 14:55:59 CST 2002
setmqaut -m QM1 -n SYSTEM.DEFAULT.LOCAL.QUEUE -t queue -g grp -all +inq
The setmqaut command completed successfully.
dspmqaut -m QM1 -n SYSTEM.DEFAULT.LOCAL.QUEUE -t queue -g grp
Entity grp has the following authorizations for object
SYSTEM.DEFAULT.LOCAL.QUEUE:
 inq
 echo 'refresh security' | runmqsc
0783889, 5765-B75 (C) Copyright IBM Corp. 1994, 2000. ALL RIGHTS
RESERVED.
Starting MQSeries Commands.
 1 : refresh security
AMQ8560: MQSeries security cache refreshed.
One MQSC command read.
No commands have a syntax error.
All valid MQSC commands were processed.
rcdmqimg -m QM1 -t queue SYSTEM.AUTH.DATA.QUEUE
AMQ7044: Media recovery not allowed.
# Done
```

T Robert Wyatt (USA)

PUT/GET using Correl-ID

INTRODUCTION

When starting to develop our MQSeries applications we were unable to find sample source code for a program to PUT or GET messages with Correl-ID so I would like to share my experiences of developing such a program in Java.

Basically, there are two ways to get your message from the queue; either you remove a message when you read it, or you copy the message, keeping the original message on the queue, which is called browsing.

There are various options that you can set to determine how and when you read messages, eg:

- Whether a call should wait for a message to arrive.
- Whether the call is part of a unit of work.

- Whether a non-persistent message is retrieved outside the syncpoint, thereby allowing fast messaging.
- Whether the message is removed from the queue or merely browsed.
- Whether to select a message by using a browse cursor or by other selection criteria.

You can also set 'Match Options' to match the message, which sets the selection criteria.

There are various ways you can group your messages, for example, Group-ID, Message-ID, and Correl-ID.

I am providing a sample PUT/GET using Correl-ID. This sample groups your messages using Correl-ID and you can safely retrieve only those with a specified Correl-ID.

Let's say there are messages in a single queue, which are destined for different applications. Let's also say that, when application A1 browses that queue, it should retrieve only messages which are destined for A1; all other messages should remain where they are.

Similarly, when application A2 browses that queue it should retrieve messages which are destined for A2 only. All other messages should remain in the queue.

This sample uses Correl-ID to distinguish between messages.

HOW IT WORKS

The source code contains two Java programs: *MQPutc.java* and *MQGetc.java*.

MQPutc connects to the queue manager and opens the destination queue with an open option for putting the message. Here we assign the desired Correl-ID to that message (passing it as a parameter with the application) and then we close the queue and queue manager.

MQGetc connects to the same queue manager and queue. We first count the number of messages in the queue. Now, with the supplied Correl-ID (passed as a parameter with the application), we search for the message employing the same Correl-ID (ie match option=

MQC.MQMO_MATCH_CORREL_ID) and retrieve it.

HOW TO USE IT

I tested the code on Win2000 and MQSeries5.1. It should support all platforms that run MQSeries and Java.

1 Open the MQSeries Explorer.

- create queue manager QM
- create queue Q1.

2 Download the code in the same folder as the two files *MQPutc.java* and *MQGetc.java*.

- compile the code from that folder, ie *javac MQPutc.java*
- compile the code from that folder, ie *javac MQGetc.java*.

3 First make sure that your queue manager is up and running. Run the *MQPutc* application with a Correl-ID of your choice, eg app1.

- run the PUT code from that folder, ie *java MQPutc app1*

  (You can put many messages with the above command, using different Correl-IDs.)

- run the GET code from that folder, ie *java MQGetc app1*

  The output will give the total number of messages in the queue and the output of the messages with the Correl-ID app1.

When you have many messages with many Correl-IDs you may find that, when retrieving with some Correl-IDs, you get the message 'Unable to load message catalog – mqji. There are no more messages available of specified ID.' Don't worry about this; the message occurs because no more messages are found with the same Correl-ID (reason code==2033).

MQPUTC.JAVA

```
import com.ibm.mq.*;
public class MQPutc
{
  private String qManager ="QM";
```

```
  private MQQueueManager qMgr;
  public static void main (String args[])
  {
     if (args.length < 1)
     {
     System.out.println("Required parameter missing - Correlation ID");
     }
     else
     {
       new MQPutc(args);
      }
     }
  public MQPutc(String args[])
  {
    try{
        qMgr =new MQQueueManager(qManager);
          int openOptions =MQC.MQOO_INPUT_AS_Q_DEF |MQC.MQOO_OUTPUT ;
        MQQueue myque = qMgr.accessQueue("Q1",openOptions);
        MQMessage hello_world =new MQMessage();
        hello_world.writeUTF("Hello World");
        MQPutMessageOptions pmo =new MQPutMessageOptions();
          String correlationId = args[0];
         hello_world.correlationId = correlationId.getBytes();
        myque.put(hello_world,pmo);
        System.out.println("Message put on queue");
        myque.close();
    qMgr.disconnect();
    } // End of try
    catch (MQException ex)
    { System.out.println("An MQSeries error occurred :Completion code "+
        ex.completionCode +"Reason code "+ex.reasonCode);
    }
    catch (java.io.IOException ex)
    {
      System.out.println("An error occurred whilst writing to the
message buffer: " + ex);
      }
      catch (Exception ex)
      {
       System.out.println("An error occurred whilst writing to the
message buffer:"+ex);
      }
    }
}
```

MQGETC.JAVA

```
import com.ibm.mq.*;
public class MQGetc
{
```

```
  private String qManager ="QM";
  private MQQueueManager qMgr;
  public static void main(String[] args)
  {
    if (args.length < 1)
   {
     System.out.println("Required parameter missing - Correlation ID");
         }
         else
         {
           new MQGetc(args);
          }
  }
  public MQGetc(String args[])
   {
    try{
  qMgr =new MQQueueManager(qManager);
    int openOptions = MQC.MQOO_INPUT_SHARED | MQC.MQOO_BROWSE |
MQC.MQOO_INQUIRE;
  MQQueue myque = qMgr.accessQueue("Q1",openOptions, null, null, null);
      MQMessage retrievedMessage =new MQMessage();
      int count1=myque.getCurrentDepth();
      System.out.println("Total Number of messages is="+count1);
      int j;
      for(j=1;j<=count1;j++)
      {
    try
    {
      String m_strCorrelID =  args[0];
      retrievedMessage.correlationId = m_strCorrelID.getBytes();
      MQGetMessageOptions gmo =new MQGetMessageOptions();
      gmo.matchOptions = MQC.MQMO_MATCH_CORREL_ID;
      retrievedMessage.messageId = MQC.MQMI_NONE;
      myque.get(retrievedMessage, gmo);
      String msgText =retrievedMessage.readUTF();
      System.out.println("The matched message is:"+msgText);
      retrievedMessage.clearMessage();
        }
        catch (MQException ex)
        {
            if (ex.reasonCode==2033)
          {
       System.out.println("There are no more messages available of
specified ID");
        System.exit(0);
      }
      else
      {
     System.out.println("An MQSeries error occurred :Completion code "+
        ex.completionCode +"Reason code "+ex.reasonCode);
      }
```

```
          }
      catch (java.io.IOException ex)
      {
        System.out.println("Java exception: " + ex);
      }   } //end for loop
  int count4=myque.getCurrentDepth();
  System.out.println("Number of messages Finally in Queue="+count4);
  myque.close();
  qMgr.disconnect();
  } // End of try
  catch (MQException ex)
  {
    if (ex.reasonCode==2033)
    {
      System.out.println("There are no messages available of specified
type");
       System.exit(0);
    }
    else
    {
    System.out.println("An MQSeries error occurred :Completion code "+
        ex.completionCode +"Reason code "+ex.reasonCode);
        }
      }
      catch (Exception ex) {
        System.out.println("An error occurred whilst writing to the
message buffer:"+ex);
      } //End of catch.
  }
}
```

Jitendra M Mistry
Software Engineer, CSSL (India)

MQSI V2 database administration tips

This article is a compilation of database tips that may be useful when supporting or establishing an MQSI V2 infrastructure.

UNABLE TO DEPLOY BECAUSE OF AN OUTSTANDING DEPLOY REQUEST

When a deployment request is initiated from the MQSI Control Centre an entry is added to the CBROKER table in the configuration manager's

database (MQSICMDB). This entry indicates the broker that is being deployed to and that it is in a deploying status (DPLING). When the deploy completes, a message is returned from the broker to the configuration manager, which then deletes the table entry, allowing further deploys to take place.

However, in certain circumstances a response may not be received from the broker and you will be unable to carry out further deploys. In such exceptional circumstances you can delete the entry from the configuration manager database and, assuming you have taken the necessary steps to ensure that future deploys will work, carry on with further deploys without incident. This is how you would delete the entry from the database for broker BROKER1:

- Start a DB2 command line process.
- Connect to the configuration manager database (MQSICMDB) using an appropriate DB2 administrator's id:
  - connect to MQSICMDB user brokerid using bkpaswrd.
- Display the columns containing the broker name and status:
  - note that if you are connecting to the database using an id that is different from the database owner you will need to prefix the table name with the id of the owner, eg DB2ADMIN.CBROKER
  - this will display entries for each of the brokers supported by the configuration manager. Typically for each broker there will be entries with csections of SHARED and DPLED
  - if there is an outstanding deploy against the broker there will be a DPLING entry.
- Delete the deploying entry from the table:
  - delete from CBROKER where CNAME='BROKER1' and CSECTION='DPLING'
  - note that the quotation marks (') must be included.

DELETING A BROKER FROM A CONFIGURATION TOPOLOGY

When a broker is deleted from the topology of a configuration manager an

entry for that broker is added to the CBROKER table. The entry has a csection value of GRAVEY, for graveyard. There may be instances when you want to delete and recreate a broker within the same topology – for instance, if you are migrating from one platform to another, or, in some exceptional circumstances, where the broker has become corrupt.

While the GRAVEY entry remains you will not be able to add the broker back into the topology of the configuration manager. It will also cause errors when trying to deploy to other brokers in the topology. The example below illustrates how you would resolve this for broker BROKER1.

- Start a DB2 command line process.
- Connect to the configuration manager database (MQSICMDB) using an appropriate DB2 administrator's id:
  - connect to MQSICMDB user brokerid using bkpaswrd.
- Display the columns containing the broker name and status:

  Select cname, csection from CBROKER where cname='BROKER1'
- This will display the following entries:

```
CNAME          CSECTION
-----          --------
BROKER1        SHARED
BROKER1        DPLED
BROKER1        GRAVEY
```

- Delete all entries for the broker that has been deleted from the configuration manager's topology:
  - delete from CBROKER where cname='BROKER1'
  - note that the quotation marks (') must be included.

USE OF TABLE ALIASES

If you have a message flow that is performing database look-ups or updates you will use a compute, database, or warehouse node. Typically you will access the table using ESQL similar to the example below:

```
Set OutputRoot.XML.My.FirstName =
 THE (SELECT ITEM A.Fname FROM Database.CUSTOMER AS A
```

```
    WHERE A.Surname = 'Smith');
```

Access to the table is made under the id that the broker is running via an ODBC connection to the database. If the table has been set up under a schema that does not match the id of the broker, for example DB2USER, then the ESQL above would have to be amended as follows:

```
Set OutputRoot.XML.My.FirstName =
 THE (SELECT ITEM A.Fname FROM Database.DB2USER.CUSTOMER AS A
  WHERE A.Surname = 'Smith');
```

This can present a number of problems. Firstly, it means the tables have to be qualified, adding to the complexity of the statements. More importantly, it means the MQSI developer has to know the schemas used for the tables, requiring more knowledge of the database administration function than is necessary and providing additional development overhead.

As you move from environment to environment, eg development through integration to production, database administration may use different schemas. Therefore the flows would have to be amended in each environment to reflect this.

Even within a given environment there may be a number of releases of test data to the tables. It is useful in this situation to have different schemas to separate different releases of data. This would mean amending the flows for every new release of data.

However, if table aliases are created for the schemas, this problem is removed. Usually the database administrators would create the table aliases. This method allows flows to use different table schemas within a given environment as well as migration of flows from one environment to another without change.

It also means that, if you have a number of brokers running under different ids but accessing the same tables, setting up table aliases for both broker ids can cater for this.

Mark Richards (UK)

July 2000 – June 2002 index

Below is an index of all topics covered in *MQ Update* since Issue 13, July 2000. The numbers in **bold** are issue numbers and the ones in brackets are page numbers. Back issues of *MQ Update* are available from Xephon – see page two for details.

Topic	Issue (page)
64-bit applications	**34** (3-10)
ActiveX	**13** (21-31)
Administering WebSphere MQ	
MQSI	**34** (19-34)
APPC	
SideInfo dataset	**18** (35-41) **19** (3-6)
Application design	**23** (39-43)
Architecture	
Unix	**21** (27-39)
AS/400	
V5.1 and V5.2	**22** (33-47)
Journal management	**27** (25-31)
Availability	
OS/390	**26** (41-43)
Back-up	
Open system platforms	**33** (25-33)
Recovery	**15** (3-24)
BMC Patrol	
Message rates	**27** (22-24)
Channel events	
Printing details	**32** (3-12)
Channel exits	**28** (5-9)
Security exits	**21** (11-27)
Channel initiator	**16** (43-43)
Clusters	**15** (35-42) **18** (24-35) **24** (3-11) **34** (29-47)
Communication buffers	**26** (34-41)
Configurations	**23** (30-38)
Hints and tips	**25** (3-8)
MQ Bridge	**16** (26-39)
Queue Managers	**21** (39-43)
Configuration	
MQ and MSCS	**35** (35-43) **36** (7-17)
Connecting applications	**32** (23-33) **33** (11-24)
Copying messages	**33** (3-11)
Copying queue contents	
MQSeries V1.2 to V2.1	**24** (24-27)
Correl-ID	**36** (37-42)
Coupling Facility and DB2	
OS/390	**20** (5-20)
CSQUTIL	**28** (26-27)
Customizing files	
CSQ4APPL	**14** (34-43)
CSQ4CHNL	**25** (16-23)
CSQ4INP2	**14** (34-43)
CSQ4MQxx	**17** (39-42)
CSQ4XPRM	**25** (16-23)
CSQ4ZPRM	**16** (39-42)
CSQXMQxx	**24** (42-43)
DB2	
MQSI V2 on AIX	**23** (3-9)
Dead letter queues	
Handler	**34** (11-22)
Browser	**35** (3-8)
Distributed queueing	
CICS	**13** (41-43)
eNDI	**29** (6-27)
Error handling	**31** (37-43)
Error logs	
Unix	**24** (27-32)
Everyplace	**13** (31-41)
Exception processing	
MQSI	**28** (10-26)
MQSI V2	**27** (6-21)
Expired messages	**31** (8-20)
Filter and list MQ objects	
OS/390	**24** (33-41)
Firewalls	**26** (27-33)
Global transactions	
MQ and Oracle	**35** (9-19) **36** (18-27)

Topic	Issue (page)
Installable services	**16** (3-25)
JMS Publish-and-subscribe	**13** (3-20)
	14 (3-11)
Logging	
AIX	**20** (3-5)
OS/390	**25** (8-13)
Message length	**25** (40-43)
Message throughput	
MQSI V2	**24** (11-23)
Drivers	**25** (32-39)
Reporting on AIX	**29** (27-36)
Messages	
Deleting	**25** (13-15)
Processing	**30** (6-22)
Messaging models	
IP multicasting	**26** (3-8)
Microsoft Host Integration Server	
MQSeries Bridge	**27** (31-45)
Deployment, configuration	**28** (27-47)
Administration, performance	**29** (3-6)
Microsoft MSMQ	**17** (15-39)
MSMQ and MQ Bridge	**18** (7-23)
Microsoft Transaction Server	**15** (24-35)
MQSeries for NT	
Client/server security	**22** (4-21)
MQSI	**14** (18-33)
Database administration tips	**36** (42-45)
Hardware capacity	**34** (23-28)
Message processing	**21** (3-10)
Migrating rules and formats	**17** (43)
Service Trace	**28** (3-5)
V2.0.2	**29** (36-43)
Performance	**30** (39-43)
Performance trace analysis	
MQSI V2	**22** (28-33)
Queue Manager	
S/390	**18** (3-7)
Unix	**22** (22-28)
Queue Names	
CICS	**19** (6-10)
MQSI V2	**23** (9-10)
Queues	
OS/390	**25** (24-31)
Quick Start	**20** (25-43)
Response times	**33** (33-43)
Set authorities scripts	**36** (3-6)
Su	**27** (46-47)
System management	
OS/390	**23** (11-29)
System resources	
Unix	**30** (3-6)
TCP/IP channels	
Reliability	**31** (3-8)
Training	**19** (33-43)
Transaction coordination	**30** (22-38)
Transaction integrity	**20** (21-24)
Transaction processing	**14** (12-17)
Triggering	**32** (13-22)
Batch jobs on OS/390	**36** (28-37)
Unix	**17** (3-15)
Utilities	**24** (41)
Wireless applications	**19** (11-32)
Workflow	
Concepts and techniques	**15** (42-43)
MQSI	**32** (33-47)
Workload balancing	
Cluster queue	**27** (3-5)
Wrappers	**31** (21-36)
XML messages	
MRM	**22** (3-4)

CommerceQuest has announced the CommerceQuest CICS Process Integrator which accesses, exposes, and re-integrates CICS transactions and VSAM files via XML, supporting existing applications and data while creating new and more modern interfaces without conversions or migrations.

The software will let sites graphically assemble existing CICS transactions and distributed applications into new business processes, access and expose VSAM data stores using XML, and XML-enable CICS and batch COBOL programs.

It will also integrate distributed systems with CICS and VSAM via WebSphere MQ, HTTP/S, and TCP/IP and provide end-to-end visibility of all transactions.

For further information contact:
CommerceQuest, 2202 N Westshore Blvd, Tampa, FL 33607, USA.
Tel: (813) 639 6300.
URL: http://www.commercequest.com/business_process_integrator.asp.

* * *

Candle has announced immediate support for z/OS 1.3 and reaffirmed its support of IBM's Workload Licence Charges software pricing structure for CICS, DB2, and IMS.

Day-one support includes OMEGAMON II for MVS, CICS, DB2, IMS, DBCTL, SMS and mainframe networks; OMEGAMON XE for Sysplex, CICSplex, DB2plex, IMSplex, UNIX Systems Services, OS/390, CICS and DB2; OMEGAMON DE; DB/EXPLAIN, DB/DASD, DB/SMU, DB/WORKBENCH, DB/QUICKCHANGE and DB/QUICK COMPARE; OMEGAVIEW TN3270, OMEGACENTER Gateway, AF/OPERATOR and AF/REMOTE; OMEGAMON XE Management Pac for MQSeries; CL/SUPERSESSION and CL/CONFERENCE; and PQEdit for MQSeries.

The company also supports new CICS Transaction Service for z/OS Version 3, covering OMEGAMON XE for CICS, OMEGAMON XE for CICSplex, and OMEGAMON II for CICS.

Meanwhile, the Candle Workload Licence Charges pricing support extends to CICS, DB2, and IMS solutions for sites using z/OS 1.3 and IBM's reporting tools and can provide an IBM invoice as proof of pricing. For sites with ongoing maintenance, the company will negotiate the maintenance level based on the licensed subcapacity as shown by the IBM invoice and subcapacity report.

Computer Associates has also announced first-day support for z/OS Version 1 Release 3, coinciding with the general availability of the operating systems, as well as with the new zSeries 800 entry-level servers.

For further information contact:
Candle, 201 N Douglas St, El Segundo, CA 90245, USA.
Tel: (310) 535 3600.
URL: http://www.candle.com.
Computer Associates, One Computer Associates Plaza, Islandia, NY 11749, USA.
Tel: (631) 342 6000.
URL: http://ca.com/products/zos_e.