9 MQ update

March 2000

In this issue

MQ Update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: +44 1635 550955
e-mail: HarryL@xephon.com

North American office

Xephon/QNA
1301 West Highway 407, Suite 201-405
Lewisville, TX 75077-2150
USA
Telephone: +1 940 455 7050

Contributions

Articles published in *MQ Update* are paid for at the rate of £170 ($250) per 1000 words and £90 ($140) per 100 lines of code. For more information about contributing an article, please check Xephon's Web site, where you can download *Notes for Contributors*.

MQ Update on-line

Code from *MQ Update* is available from Xephon's Web site at www.xephon.com/mqupdate.html (you'll need the user-id shown on your address label to access it). If you've a problem with your user-id or password call Xephon's subscription department on +44 1635 33886.

Editor

Harry Lewis

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.

Using MQSeries as a transaction coordinator

INTRODUCTION

The MQSeries System Administration Guide and MQSeries Application Programming Guide discuss the subject of a unit of work in some detail. MQSeries treats a unit of work as local if it involves updates in one queue manager. The unit of work is global if it involves database updates. The coordination of a global unit of work may include any XA-compliant relational database. An XA-compliant relational database is defined in terms of the X/Open Distributed Transaction Processing standard (in other words, it comes from the XA specification). The XA specification identifies transaction managers, resource managers, and an XA interface to achieve two-phase commit. Most relational databases can act as resource managers, which is another way of saying that they are XA-compliant. MQSeries is a unique product in the sense that it can behave as either a transaction manager or a resource manager. This behaviour is controlled by the configuration of the queue manager, which can be that of a resource manager to other transaction monitors or a transaction manager to work alongside resource managers.

In this article I discuss how to use an MQSeries queue manager to coordinate updates to an Oracle database. While the MQSeries *Application Programming Guide* and *System Administration Guide* provide quite good detail on how to use MQSeries with DB2, the information on Oracle is rather sketchy. This article should provide enough detail for someone to configure an MQSeries queue manager and test a sample application using both commit and rollback scenarios. To demonstrate the process of coordinating a queue manager to update Oracle, I use MQSeries 5.0 on HP-UX 10.20 with Oracle 7.3.4.

PLANNING

The object of this exercise is to demonstrate the use of a queue manager as a transaction coordinator and Oracle as a resource manager. I also describe the steps necessary to configure the MQSeries queue

manager and verify that it is working properly. However, to verify that our objective is really achieved, we should at least write a test application to ensure that we are able to control transactions using MQI calls.

Our set-up comprises a queue manager called *QMGR* and a queue *TESTQUEUE*. Assume that, whenever a client puts a message on this queue, an MQSeries-controlled unit of work starts. This unit of work involves two steps:

- *EXEC SQL UPDATE* on the Oracle table.
- *MQPUT* a message on *TESTQUEUE* to flag that data held in Oracle has been updated.

If the transaction is committed successfully, the database should be updated and a confirmatory message should be present on the *TESTQUEUE*. If the transaction fails, no message should be placed on the *TESTQUEUE* and the Oracle table should be in the state that it was in prior to the unit of work commencing. If necessary, the test application can be extended to multiple tables, databases, etc. However, our test should establish that the basic aspects of unit of work management are functioning correctly.

CONFIGURATION

We assume that the queue manager has been created and is available. Similarly, the instance of the database has also been created and is available. In this section, we detail how to configure both the MQSeries queue manager and Oracle, and how to configure details that are specific to the test application. A number of these tasks are documented in IBM's *MQSeries System Administration Book*, including:

- Building the Oracle XA switch load file
- Configuring the queue manager
- Configuring the Oracle instance
- Application-specific configurations.

BUILDING THE ORACLE XA SWITCH LOAD FILE

The MQSeries product provides the program source files and makefiles required for building the Oracle XA switch load file. These files are located in directory *mqmtop/samp/xatm*, and we have to copy them to directory */home/mqm/xatm*. The required files are:

- *xa.h*
- *oraswit.c*
- *xaswit.mak*.

The *xaswit.mak* makefile assumes that the switch load file is called *oraswit* and is generated in the current directory. To build the file, export the variable *ORACLE_HOME* and then issue the following command:

```
make -f xaswit.mak oraswit
```

At our installation, I had to rebuild the file *libclntsh.sl* in the Oracle library because of a problem with symbols.

CONFIGURATION OF THE QUEUE MANAGER

Add the lines below to the *qm.ini* file (this is located in */var/mqm/qmgrs/QMGR/qm.ini* for MQSeries 5.0 on HP-UX 10.20) of the queue manager for *XAResourceManager*. (The continuation character, '➤', denotes a formatting line break.)

```
Name=OracleXA
SwitchFile=/home/mqm/xatm/oraswit
XAOpenString=Oracle_XA+Acc=P/orauser/orapass
➤  +SesTm=35+LogDir=/tmp
```

Here *Name* is a string given by the user. The name is usually not significant, except when you have multiple sections that refer to an *XAResourceManager* and one of them is failing. Note that, while *XAOpenString* has many options, we are setting only the ones that are necessary. *XAOpenString* comprises a number of parameters separated by the plus sign ('+'). The first parameter is always *Oracle_XA*. The second parameter, in this case, contains account information. Oracle user *orauser* with password *orapass* is used to connect to the instance

of Oracle concerned. *SesTm* is a timeout parameter for the session. This is a mandatory parameter and it denotes the maximum time that a transaction can be inactive before the system automatically deletes it. The last parameter of *XAOpenString* is *LogDir*, which is a directory for writing Oracle XA log messages. Note that the *mqm* user id should have permission to write to this directory. For more information on *XAOpenString*, check the *MQSeries System Administration Guide* or your Oracle documentation.

ORACLE SET-UP

In order for *XAOpenString* to work, we must be in possession of an Oracle user id 'orauser' with password 'orapass'. Additionally, we must have access to the *sys.v$xatrans$* table and the user 'orauser' must have *SELECT* permission on this table.

Log on to Oracle as *sys* and create *v$xatrans$* using *$ORACLE_HOME/rdbms/admin/xaview.sql* and *sqlplus*:

```
grant select on v$xatrans$ to orauser
```

TEST PLAN

The test application involves an Oracle table called *representative*. The only details in this table in which we are interested are *REP_ID* ('representative id') and *FIRST_NAME*. The table includes one row in which *REP_ID* is '34567' and *FIRST_NAME* is 'Bill'.

The application first issues an SQL statement to *UPDATE* the row with *REP_ID* '34567' by changing *FIRST_NAME* to 'William' and then it *PUT*s a message containing information about the change on *TESTQUEUE*.

The application can accept the name of a queue and that of a queue manager as parameters. The changes detailed in the message are committed unless an additional, optional 'rollback' parameter is specified. To initialize the test runs, we remove all the messages from the *TESTQUEUE* in *QMGR*.

In order to demonstrate the *COMMIT* scenario, we issue the message

and run the application. The expected results of this run are as follows:

1 The *UPDATE* is committed and the value of *FIRST_NAME* is changed to 'William'.

2 There is one message on *TESTQUEUE* in *QMGR* that contains information about the new name and *REP_ID*.

In order to demonstrate *the ROLLBACK* scenario, we re-initialize the test by clearing all messages from *TESTQUEUE* and resetting *FIRST_NAME* to 'Bill'. We then issue the message and run the application with the *ROLLBACK* option. The expected results of this run are as follows:

1 There are no messages on *TESTQUEUE*.

2 The SQL *UPDATE* is issued and then rolled back, so that the value of *FIRST_NAME* is 'Bill' at the end.

This test is scripted, and the script and expected results are also included in the article.

TEST APPLICATION

The program below contains an *MQBEGIN* statement to indicate the start of the unit of work. The unit of work terminates with either an *MQCMIT* or an *MQBACK*. The *EXEC SQL UPDATE* and *MQPUT* are executed between these transaction boundaries and thus form part of the unit of work.

The use of *MQI* verbs, such as *MQPUT* and *MQGET*, within the unit of work needs closer inspection. In the test application, we use *MQPUT* with *MQPMO_SYNCPOINT*. This option must be used to identify that the *MQPUT* is in the unit of work. When this option is specified, the message put on the queue is not visible until the *MQCMIT* is issued. If this option is not used, the default for Unix systems is *MQPMO_NO_SYNCPOINT*, which results in the *MQPUT* not being included in the unit of work.

The sections that follow contain the pro*C source listing, a script to build the **MqOraTest** application, and a test script.

MQORATEST

```
/*****************************************************************/
/*                                                               */
/* Program name: MqOraTest                                       */
/*                                                               */
/* Description:  Test proC program for MQSeries and Oracle XA    */
/*                                                               */
/*****************************************************************/

/*****************************************************************/
/*                                                               */
/* MqOraTest parameters - The name of the message queue (required)  */
/*                      - The queue manager name (required)      */
/*                      - rollback (optional keyword)            */
/*                                                               */
/*****************************************************************/
/*                                                               */
/*****************************************************************/
/*                                                               */
/* Includes                                                      */
/*                                                               */
/*****************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <cmqc.h>

/*****************************************************************/
/*                                                               */
/* Defines                                                       */
/*                                                               */
/*****************************************************************/
#define OK      0                          /* define OK as zero    */
#define NOT_OK  1                          /* define NOT_OK as one */

/*****************************************************************/
/*                                                               */
/* Define and declare an SQLCA (SQL Communication Area) structure   */
/*                                                               */
/*****************************************************************/
EXEC SQL INCLUDE SQLCA;

void sql_error ();
                                       /*****************************/
int main(int argc, char *argv[])       /* ----- START OF MAIN ----- */
                                       /*****************************/
{
    EXEC SQL WHENEVER SQLERROR DO sql_error("ORACLE error--\n");
```

```
MQOD od = {
    MQOD_DEFAULT    };                  /* object descriptor   */
MQMD md = {
    MQMD_DEFAULT    };                  /* message descriptor  */
MQPMO pmo = {
    MQPMO_DEFAULT   };                  /* get message options */
MQBO bo = {
    MQBO_DEFAULT    };                  /* begin options       */

MQLONG rc=OK;                          /* return code             */
MQHCONN hCon;                          /* handle to connection    */
MQHOBJ hObj;                           /* handle to object        */
char QMName[50]="";                    /* default QM name         */
MQLONG options;                        /* options                 */
MQLONG reason;                         /* reason code             */
MQLONG connReason;                     /* MQCONN reason code      */
MQLONG compCode;                       /* completion code         */
MQLONG openCompCode;                   /* MQOPEN completion code  */
char msgBuf[100];                      /* message buffer          */
MQLONG msgBufLen;                      /* message buffer length   */
MQLONG msgLen;                         /* message length received */

char *pStr;                            /* ptr to string           */
int gotMsg;                            /* got message from queue  */
int committedUpdate;                   /* committed update        */
int rollback ;                         /* ask for rollback        */
long balanceChange;                    /* balance change          */

/*****************************************************************/
/* SQL host declarations                                         */
/*****************************************************************/
EXEC SQL BEGIN DECLARE SECTION;
char first_name[40];                           /* name           */
long rep_id;                                   /* account number */
EXEC SQL END DECLARE SECTION;

/*****************************************************************/
/* First check we have been given correct arguments              */
/*****************************************************************/
if (argc != 3 && argc != 4)
{
    printf("Usage : %s 'queue name' 'qmgr name' '[rollback]'.\n",
        argv[0]);
    exit(99);
}
strcpy(QMName, argv[2]);               /*  qmgr name supplied     */
if (argc == 4)
{
    if ( ! strcmp("rollback", argv[3]))
        rollback = 1;                  /*  set rollback to true   */
```

```
    else
        rollback = 0;                /*  set rollback to false  */
}
printf ("The current value of rollback = %d \n", rollback);

/****************************************************************/
/* Connect to queue manager                                     */
/****************************************************************/
MQCONN(QMName, &hCon, &compCode, &connReason);
if (compCode == MQCC_FAILED)
{
    printf("MQCONN ended with reason code %li\n", connReason);
    exit((int) connReason);
}

/****************************************************************/
/* Use input parameter as the name of the target queue          */
/****************************************************************/
strncpy(od.ObjectName, argv[1], (size_t) MQ_Q_NAME_LENGTH);
printf("Target queue is %s\n", od.ObjectName);

/****************************************************************/
/* Open the target message queue for output                     */
/****************************************************************/
options = MQOO_OUTPUT + MQOO_FAIL_IF_QUIESCING;
MQOPEN(hCon, &od, options, &hObj, &openCompCode, &reason);
if (reason != MQRC_NONE)
    printf("MQOPEN ended with reason code %li\n", reason);

if (openCompCode == MQCC_FAILED)
{
    printf("Unable to open queue for output\n");
    rc = openCompCode;                  /* stop further action */
}

/****************************************************************/
/* Set up MQPUT                                                 */
/****************************************************************/
msgBufLen = sizeof(msgBuf) - 1;
pmo.Options = MQPMO_SYNCPOINT;

{
    /************************************************************/
    /* Set flags so that we can back out if something goes      */
    /* wrong and not lose the message.                          */
    /************************************************************/
    gotMsg = 0;                          /* set flag to FALSE */
    committedUpdate = 0;                 /* set flag to FALSE */
```

```
/*************************************************************/
/* Start a unit of work                                      */
/*************************************************************/
MQBEGIN (hCon, &bo, &compCode, &reason);

if (reason == MQRC_NONE)
{
    printf("Unit of work started\n");
}
else
{
    printf("MQBEGIN ended with reason code %li\n", reason);
    rc = NOT_OK;
}

if (compCode == MQCC_FAILED)
    printf("Unable to start a unit of work\n");

/*************************************************************/
/* Put message on queue.  Hardcoded.                         */
/*************************************************************/
if (rc == OK)
{
    memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
    memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
    msgBufLen = 20 ;
    memcpy ( msgBuf , "William 34567 ",12 );
    msgBuf[20] = '\0' ;
    MQPUT(hCon, hObj, &md, &pmo, msgBufLen, msgBuf,
        &compCode, &reason);

    if (reason != MQRC_NONE)
    {
        printf("MQPUT ended with reason code %li\n",
                reason);
        compCode = MQCC_FAILED;    /* stop looping    */
    }
    else
    {
        gotMsg = 1;      /* set flag to TRUE          */
    }
    printf ( " completed MQPUT call \n" );
}

if (compCode != MQCC_FAILED && rc == OK)
{
    /*********************************************************/
    /* Put details in database                               */
    /*********************************************************/
    if (rc == OK)
```

```
    {
        rep_id = 34567 ;
        EXEC SQL SELECT first_name
            INTO :first_name
            FROM  REPRESENTATIVE
            WHERE REP_ID = :rep_id
            ;

        printf ( "The name is %s \n", first_name );
        printf ( "The rep_id  is %d \n", rep_id );
        strcpy ( first_name , "William" );

        EXEC SQL UPDATE REPRESENTATIVE
            SET FIRST_NAME = :first_name
            WHERE REP_ID= :rep_id
            ;

        printf ( " completed SQL UPDATE statement \n" );
        if (rc == OK)
        {
            /************************************/
            /* We are going to commit the update: */
            /* even if something goes             */
            /* wrong now, the message has been    */
            /* used so don't back out.            */
            /************************************/
            committedUpdate = 1;      /* set flag to TRUE */
            if ( !rollback ) {
                MQCMIT(hCon, &compCode, &reason);

                if (reason == MQRC_NONE)
                {
                    printf("Unit of work successfully
                            ➤  completed\n");
                }
                else
                {
                    printf("MQCMIT ended with reason code
                            ➤  %li completion code "
                        "%li\n", reason, compCode);
                    rc = NOT_OK;
                }
            }
        }
    }
}

/*************************************************************/
/* If we got the message, but something went wrong,         */
/* back out so that we don't lose the message.              */
```

```
        /*****************************************************************/
        if ( rollback || (gotMsg && !committedUpdate ))
        {
            MQBACK(hCon, &compCode, &reason);
            if (reason == MQRC_NONE)
                printf("MQBACK successfully issued\n");
            else
                printf("MQBACK ended with reason code %li\n", reason);
        }
    }
    /*****************************************************************/
    /* Close queue, if opened                                        */
    /*****************************************************************/
    if (openCompCode != MQCC_FAILED)
    {
        options = 0;                          /* no close options    */
        MQCLOSE(hCon, &hObj, options, &compCode, &reason);

        if (reason != MQRC_NONE)
            printf("MQCLOSE ended with reason code %li\n", reason);
    }

    /*****************************************************************/
    /* Disconnect from queue manager, if not already connected       */
    /*****************************************************************/
    if (connReason != MQRC_ALREADY_CONNECTED)
    {
        MQDISC(&hCon, &compCode, &reason);

        if (reason != MQRC_NONE)
            printf("MQDISC ended with reason code %li\n", reason);
    }
    return 0;
                                      /*****************************/
}                                     /* ------ END OF MAIN ------ */
                                      /*****************************/

void
sql_error(msg)
char *msg;
{
    char err_msg[128];
    size_t buf_len, msg_len;

    EXEC SQL WHENEVER SQLERROR CONTINUE;

    printf("\n%s\n", msg);
    buf_len = sizeof (err_msg);
    sqlglm(err_msg, &buf_len, &msg_len);
    printf("%.*s\n", msg_len, err_msg);
```

```
    /***    EXEC SQL ROLLBACK RELEASE; ****/
    exit(1);
}
```

To build the application, use following commands:

```
proc MqOraTest.pc
cc  -g -Aa -D_HPUX_SOURCE -D_REENTRANT MqOraTest.c \
-o MqOraTest \
-I $ORACLE_HOME/include  -L$ORACLE_HOME/lib -lclntsh \
-L/opt/mqm/lib  -lmqm -lc -lM
```

To test run the application, use **amqsget** to verify that the message is on queue *TESTQUEUE*.

TEST SCRIPT

```
ORACLE_HOME=/oracle/product/app/oracle/7.3.4.3
export ORACLE_HOME

TWO_TASK=grc2t
export TWO_TASK

PATH=$ORACLE_HOME/bin:$PATH
export PATH

echo      Restore Name to Bill
sqlplus   orauser/orapass @name2
echo      Query Current Values For name. Must be Bill.
sqlplus   orauser/orapass @name
echo      Drain queue TESTQUEUE on QMGR queue manager
echo      The number of messages at this time do not matter.
amqsget   TESTQUEUE QMGR
MqOraTest TESTQUEUE QMGR
sleep     5
sqlplus   orauser/orapass @name
sleep     5
echo      Drain queue TESTQUEUE on QMGR queue manager. Show EXACTLY
          ➤  one message.
amqsget   TESTQUEUE QMGR
echo      Restore Name to Bill.
sqlplus   orauser/orapass @name2
echo      Get Current Values For name  .Now Bill .
sqlplus   orauser/orapass @name
echo      Drain queue TESTQUEUE on QMGR queue manager. Nothing to
          ➤  drain.
amqsget   TESTQUEUE QMGR
echo      Update and message PUT but then rollback.
MqOraTest TESTQUEUE QMGR rollback
```

```
echo      Get Current Values For name .Should be Bill .
sqlplus   orauser/orapass @name
echo      Drain queue TESTQUEUE on QMGR queue manager. Show EXACTLY
          ➤  NO messages.
amqsget   TESTQUEUE QMGR
echo      End

--name.sql
set pagesize 9999
select * from representative where rep_id = 34567 ;
exit

--name2.sql
set pagesize 9999
update representative set first_name = 'Bill' where rep_id = 34567 ;
exit
```

TEST RESULTS

```
Restore Name to Bill

SQL*Plus: Release 3.3.4.0.0

1 row updated.

Disconnected from Oracle7 Server Release 7.3.4.2.0 - Production

Query Current Values For name. Must be Bill .

SQL*Plus: Release 3.3.4.0.0

    REP_ID FIRST_NAME
     34567 Bill

Disconnected from Oracle7 Server Release 7.3.4.2.0 - Production

Drain queue TESTQUEUE on QMGR queue manager
The number of messages at this time do not matter.
Sample AMQSGET0 start
no more messages
Sample AMQSGET0 end
The current value of rollback = 0
Target queue is TESTQUEUE
Unit of work started
 completed MQPUT call
The name is Bill
The rep_id  is 34567
 completed SQL UPDATE statement
Unit of work successfully completed
```

```
SQL*Plus: Release 3.3.4.0.0

    REP_ID  FIRST_NAME
      34567 William

Disconnected from Oracle7 Server Release 7.3.4.2.0 - Production
Drain queue TESTQUEUE on QMGR queue manager. Show EXACTLY one message.
Sample AMQSGET0 start
message <William 34567>
no more messages
Sample AMQSGET0 end

Restore Name to Bill.

SQL*Plus: Release 3.3.4.0.0

1 row updated

Disconnected from Oracle7 Server Release 7.3.4.2.0 - Production
Get Current Values For name .Now Bill .

SQL*Plus: Release 3.3.4.0.0
    REP_ID   FIRST_NAME
      34567  Bill

Disconnected from Oracle7 Server Release 7.3.4.2.0 - Production
Drain queue TESTQUEUE on QMGR queue manager .Nothing to drain..
Sample AMQSGET0 start
no more messages
Sample AMQSGET0 end
Update and message PUT but then rollback.
The current value of rollback = 1
Target queue is TESTQUEUE
Unit of work started
 completed MQPUT call
The name is Bill
The rep_id is 34567
 completed SQL UPDATE statement
MQBACK successfully issued
Get Current Values For name. Should be Bill.

SQL*Plus: Release 3.3.4.0.0

    REP_ID FIRST_NAME
      34567 Bill

Disconnected from Oracle7 Server Release 7.3.4.2.0 - Production
Drain queue TESTQUEUE on QMGR queue manager. Show EXACTLY NO messages.
```

```
Sample AMQSGET0 start
no more messages
Sample AMQSGET0 end
End
```

REMARKS

Once we demonstrate transaction coordination using an MQSeries queue manager, several other issues arise. The remarks below may assist in answering some of them.

- The ability to coordinate database updates within MQSeries units of work is not supported in an MQI client application. (This ability is provided in ECI.)
- The MQI updates and database updates must both be carried out on the same server as hosts the queue manager. This is a rather strict constraint that may limit the usability of MQSeries' transaction coordination, though the constraint can be relaxed in cases such as the one dealt with by the next bullet point.
- The database server may reside on a different machine from the queue manager server as long as the database is accessed via an XA-compliant client provided by the database manager itself.
- Although the queue manager is itself XA-compliant, it is not possible to configure other queue managers as participants in global units of work. This is because only one connection at a time can be supported.
- These restrictions also apply to MQSeries Version 5.1.
- If there are any errors in the *oraclexa* switch load file, or an error in the set-up, then your queue manager will not start.
- If an error is still returned by Oracle despite the queue manager starting, then *STD_OUT* may contain an error message or the routine *sql_error*. Alternatively, you may find a file in the directory named in *XAOpenString*'s *LogDir* parameter (*/tmp* in this example). The file name is usually similar to *xaNULL092099.trc*, where *092099* is a date. If *LogDir* is not specified, you will find this file in the current directory.

- Other common problems include:
  - No *write* permission on *$LogDir*.
  - *mqm* (or the user id under which the application runs) has no *grant* permission on *sys.v$xatrans$*.
  - The Oracle environment variables *ORACLE_HOME*, *PATH*, and *ORACLE_SID* or *TWO_TASK* and library paths (*SHLIB_PATH* for HP-UX, etc) are not set before queue manager is started.
  - The attempt to compile *libclntsh.sl* using the makefile supplied by Oracle is incomplete.
- There is a performance hit associated with any transactional update. The performance hit depends on the application and may need additional profiling.

SUMMARY

This article demonstrates the use of MQSeries as transaction coordinator for Oracle. Both commit and rollback scenarios are shown using a sample application, which is provided with details of its peripheral configuration. The application should help developers to answer several questions regarding how MQI is used for coordinating transactions using *MQBEGIN*, *MQCMIT*, and *MQBACK*, as well as highlight the changes required in options for *MQPUT* (*MQPMO_SYNCPOINT*). It should help developers to understand how to use embedded SQL with MQI. The article also lists the problems that are commonly encountered and restrictions on the use of MQSeries as transaction coordinator.

Ashish Joshi
Consultant (USA)

Invoking MQSeries tools using ISPF panels

We run many MQSeries images on different Logical Partitions as a way of managing a wide variety of work at our site. This allows us, for instance, to keep our test and acceptance systems separate. However, the downside of this is the additional effort required to run the utilities that we frequently use in both test and acceptance environments.

To handle this, we developed a panel-driven system to run our most frequently used utilities. The main menu can be called from a PANEL with the following statement:

```
+3 |Utilities
3,'PANEL(MQMUTIL)'
```

The utility's main menu is shown in Figure 1.

```
MQMUTIL ----------------- MQSeries/MVS Utilities  ------------------
OPTION  ===>


       BSDS management
       1 - MQ Print log              MQSeries system must be stopped


       Queue management           - -
       2 -  copy                    |
       3 -  empty                   | MQSeries system
       4 -  load                    | must be running
       5 -  print                   |
       6 -  define entries          |
                                    |
       Commands                     |
       7 -  command               --
```

Figure 1: Main menu

MQSERIES UTILITY PANEL DEFINITIONS

The ISPF panels listed in this section are invoked.

MQMUTIL

```
)ATTR
@ TYPE(OUTPUT) INTENS(HIGH) CAPS(OFF)  JUST(LEFT)
$ TYPE(OUTPUT) INTENS(LOW)  CAPS(OFF)  JUST(ASIS)
% TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(GREEN)
¢ TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(red)
~ TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(WHITE)
? TYPE(TEXT)   INTENS(LOW)  JUST(ASIS) COLOR(YELLOW)
# TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(YELLOW)
` TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(green) HILITE(REVERSE)
| TYPE(TEXT)   INTENS(high)  JUST(ASIS) COLOR(blue)
+ TYPE(TEXT)   INTENS(LOW) color(white)
_ TYPE(INPUT)  INTENS(LOW)
)BODY
|------------------------`MQSeries/MVS Utilities |-------------------------
%OPTION  ===>_ZCMD
+
+
      % BSDS management
+      ~1 - MQ Print log   ¢ MQSeries system must be stopped
+
+
      % Queue management   ¢--
       ~2 -  copy          ¢  |
       ~3 -  empty         ¢  | MQSeries system
       ~4 -  load          ¢  | must be running
       ~5 -  print         ¢  |
       ~6 -  define entries¢  |
+                          ¢  |
      % Commands           ¢  |
       ~7 -  command       ¢--
+
+
+
+
)INIT
  .HELP = TUTORPAN              /* insert name of tutorial panel    */
)PROC
   &ZSEL=TRANS(TRUNC(&ZCMD,'.')
              1,'CMD(MQMPRLM)'
              2,'CMD(MQMCOPY)'
              3,'CMD(MQMEMPTY)'
              4,'CMD(MQMLOAD)'
              5,'CMD(MQPRINTQ)'
              6,'CMD(MQDEFINE)'
              7,'CMD(MQMCOMM)'
              X,'EXIT'
              ' ',' '
              *,'?')
```

```
  &ZTRAIL = .TRAIL
  &PFKEY  = .PFKEY
)END
```

SUBMIT

```
)ATTR
   _ TYPE(INPUT) CAPS(OFF) INTENS(HIGH) FORMAT(&MIXED)

)BODY  WIDTH(&ZWIDTH) EXPAND(||)
%&CIVER EDIT ----------------------------|-|--------------------------------
-+
%COMMAND ===>_ZCMD                       | |                %SCROLL ===>_Z
%
+ **** IF YOU WISH TO SUBMIT THIS JOB NOW, TYPE 'SUBMIT' AND PRESS ENTER. ****
%
)INIT
  .HELP  = ISR20000
  .ZVARS = 'ZSCED'

  &MIXED = MIX
  IF (&ZPDMIX = N)
    &MIXED = EBCDIC

)PROC

)END
```

MQMPRLM

```
)ATTR
@ TYPE(OUTPUT) INTENS(HIGH) CAPS(OFF)  JUST(LEFT)
$ TYPE(INPUT) INTENS(LOW)  PAD(_)
% TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(GREEN)
¢ TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(red)
~ TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(WHITE)
? TYPE(TEXT)   INTENS(LOW)  JUST(ASIS) COLOR(YELLOW)
# TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(YELLOW)
` TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(green) HILITE(REVERSE)
| TYPE(TEXT)   INTENS(high)  JUST(ASIS) COLOR(blue)
+ TYPE(TEXT)   INTENS(LOW) color(white)
_ TYPE(INPUT)  INTENS(LOW)
)BODY
|------------------------------`MQSeries Utility Panel|--------------------
%COMMAND ===>_ZCMD
+
+
+   MQSeries system id   ....$z   +
```

```
+
+
+
+
+
+
+
+
+
+   PF3 = Exit
+
)INIT
  .ZVARS = 'SYSID'
  .HELP = TUTORPAN           /* Insert name of tutorial panel    */
    &sysid=' '
    &pfkey=.pfkey
)PROC
    VER (&SYSID,NB,MSG=mqmsg001)
    &pfkey=.pfkey
)END
```

MQMCOPY

```
)ATTR
@ TYPE(OUTPUT) INTENS(HIGH) CAPS(OFF)  JUST(LEFT)
$ TYPE(INPUT) INTENS(LOW)  PAD(_)
% TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(GREEN)
¢ TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(red)
~ TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(WHITE)
? TYPE(TEXT)   INTENS(LOW)  JUST(ASIS) COLOR(YELLOW)
# TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(YELLOW)
` TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(green) HILITE(REVERSE)
| TYPE(TEXT)   INTENS(high)  JUST(ASIS) COLOR(blue)
+ TYPE(TEXT)   INTENS(LOW) color(white)
_ TYPE(INPUT)  INTENS(LOW)
)BODY
|------------------------------`MQSeries Utility Panel|----------------------
%COMMAND ===>_ZCMD
+
+
+   MQSeries system id   ....$z   +
+
+   Selection            ....$s+         P/Q    (Pageset/Queue)
+
+   Page set number      ....$psn+       0 to 99
+   Queue name           ....$queue                          +
+
+
+
+   PF3 = Exit
```

```
+
)INIT
  .ZVARS = 'SYSID'
  .HELP = TUTORPAN           /* Insert name of tutorial panel    */
    &sysid=' '
    &s=' '
    &psn=' '
    &queue=' '
    &pfkey=.pfkey
)PROC
    VER (&SYSID,NB,MSG=mqmsg001)
    VER (&S,NB,MSG=mqmsg001)
    &pfkey=.pfkey
)END
```

MQMEMPTY

```
)ATTR
@ TYPE(OUTPUT) INTENS(HIGH) CAPS(OFF)  JUST(LEFT)
$ TYPE(INPUT) INTENS(LOW)  PAD(_)
% TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(GREEN)
¢ TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(red)
~ TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(WHITE)
? TYPE(TEXT)   INTENS(LOW)  JUST(ASIS) COLOR(YELLOW)
# TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(YELLOW)
` TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(green) HILITE(REVERSE)
| TYPE(TEXT)   INTENS(high)  JUST(ASIS) COLOR(blue)
+ TYPE(TEXT)   INTENS(LOW) color(white)
_ TYPE(INPUT)  INTENS(LOW)
)BODY
|------------------------------`MQSeries Utility Panel|----------------------
%COMMAND ===>_ZCMD
+
+
+   MQSeries system id   ....$z   +
+
+   Selection            ....$s+         P/Q    (Pageset/Queue)
+
+   Page set number      ....$psn+       0 to 99
+   Queue name           ....$queue                              +
+
+
+
+   PF3 = Exit
+
)INIT
  .ZVARS = 'SYSID'
  .HELP = TUTORPAN           /* Insert name of tutorial panel    */
    &sysid=' '
    &s=' '
```

```
    &psn=' '
    &queue=' '
    &pfkey=.pfkey
)PROC
    VER (&SYSID,NB,MSG=mqmsg001)
    VER (&S,NB,MSG=mqmsg001)
    &pfkey=.pfkey
)END
```

MQMLOAD

```
)ATTR
@ TYPE(OUTPUT) INTENS(HIGH) CAPS(OFF)  JUST(LEFT)
$ TYPE(INPUT) INTENS(LOW)  PAD(_)
% TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(GREEN)
¢ TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(red)
~ TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(WHITE)
? TYPE(TEXT)   INTENS(LOW)  JUST(ASIS) COLOR(YELLOW)
# TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(YELLOW)
` TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(green) HILITE(REVERSE)
| TYPE(TEXT)   INTENS(high)  JUST(ASIS) COLOR(blue)
+ TYPE(TEXT)   INTENS(LOW) color(white)
_ TYPE(INPUT)  INTENS(LOW)
)BODY
|------------------------------`MQSeries Utility Panel|----------------------
%COMMAND ===>_ZCMD
+
+
+   MQSeries system id   ....$z   +
+
+
+   Queue                ....$queue                        +
+
+
+
+
+   PF3 = Exit
+
)INIT
  .ZVARS = 'SYSID'
  .HELP = TUTORPAN           /* Insert name of tutorial panel    */
    &sysid=' '
    &queue=' '
    &pfkey=.pfkey
)PROC
    VER (&SYSID,NB,MSG=mqmsg001)
    VER (&queue,NB,MSG=mqmsg001)
    &pfkey=.pfkey
)END
```

MQPRINTQ

```
)ATTR
@ TYPE(OUTPUT) INTENS(HIGH) CAPS(OFF)  JUST(LEFT)
$ TYPE(INPUT) INTENS(LOW)  PAD(_)
% TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(GREEN)
¢ TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(red)
~ TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(WHITE)
? TYPE(TEXT)   INTENS(LOW)  JUST(ASIS) COLOR(YELLOW)
# TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(YELLOW)
` TYPE(TEXT)   INTENS(HIGH) JUST(ASIS) COLOR(green) HILITE(REVERSE)
| TYPE(TEXT)   INTENS(high)  JUST(ASIS) COLOR(blue)
+ TYPE(TEXT)   INTENS(LOW) color(white)
_ TYPE(INPUT)  INTENS(LOW)
)BODY
|-----------------------------`MQSeries Utility Panel|----------------------
%COMMAND ===>_ZCMD
+
+
+   MQSeries system id   ....$z   +
+
+   Queue name           ....$queue                              +
+
+
+
+
+
+
+
+   PF3 = Exit
+
)INIT
  .ZVARS = 'SYSID'
  .HELP = TUTORPAN           /* Insert name of tutorial panel    */
    &sysid=' '
    &queue=' '
    &pfkey=.pfkey
)PROC
    VER (&SYSID,NB,MSG=mqmsg001)
    VER (&QUEUE,NB,MSG=mqmsg001)
    &pfkey=.pfkey
)END
```

This article concludes in next month's issue of *MQ Update*.

Paul Jansen
System Programmer
Interpay (The Netherlands)

Administering MQSeries for MVS/ESA from Unix

We conclude this article on remote administration of MQSeries for MVS/ESA from Unix, which started in last month's issue.

MYCONTROL.C (CONTINUED)

```
/* Report reason, if any; stop if failed */
if (Reason != MQRC_NONE)
{
  printf("MQOPEN ended with reason code %ld\n", Reason);
}

if (OpenCode == MQCC_FAILED)
{
  printf("unable to open server queue for output\n");
  exit(Reason);
}

/******************************************************************/
/*                                                                */
/*          Open the queue to receive the reply messages          */
/*                                                                */
/******************************************************************/
O_options = MQOO_INPUT_EXCLUSIVE
       + MQOO_FAIL_IF_QUIESCING;

strcpy(odr.ObjectName, "SYSTEM.MQSC.REPLY.QUEUE");

strcpy(odr.DynamicQName, "*");

MQOPEN(Hcon,                    /* Connection handle             */
       &odr,                    /* Object descriptor for queue   */
       O_options,               /* Open options                  */
       &Hreply,                 /* Reply object handle           */
       &OpenCode,               /* Completion code               */
       &Reason);                /* Reason code                   */
/* Report reason, if any; stop if failed      */
if (Reason != MQRC_NONE)
{
  printf("MQOPEN ended with reason code %ld\n", Reason);
}

if (OpenCode == MQCC_FAILED)
{
  printf("unable to open reply queue\n");
```

```
}
else
{
  strncpy(replyQ, odr.ObjectName, MQ_Q_NAME_LENGTH);
  printf("replies to %.48s\n", replyQ);
}

/*********************************************************************/
/*                                                                   */
/*   Read lines from the file and put them to the message queue      */
/*   Loop until null line or end of file, or there is a failure      */
/*                                                                   */
/*********************************************************************/

CompCode = OpenCode;       /* Use MQOPEN result for initial test  */
fp = stdin;

while (CompCode != MQCC_FAILED)
{
  if (fgets(buffer, sizeof(buffer) - 1, fp)
                            != NULL)   /* Read next line          */
  {
    buflen = strlen(buffer) - 1;       /* Length without end line */
    buffer[buflen] = '\0';             /* Remove end line         */
  }
  else buflen = 0;                     /* Treat EOF as null line  */

  /*******************************************************************/
  /*                                                                 */
  /*             Put each buffer to the message queue                */
  /*                                                                 */
  /*******************************************************************/
  if (buflen > 0)
  {
    md.MsgType = MQMT_REQUEST; /* Message is a request            */

    /* Ask for exceptions to be reported with original text */
    md.Report = MQRO_EXCEPTION_WITH_DATA;

    /* Reply-to-queue name */
    strncpy(md.ReplyToQ, replyQ, MQ_Q_NAME_LENGTH);

    /* Character string format */
    memcpy(md.Format, MQFMT_STRING, MQ_FORMAT_LENGTH);

    MQPUT(Hcon,                /* Connection handle               */
          Hobj,                /* Object handle                   */
          &md,                 /* Message descriptor              */
          &pmo,                /* Default options                 */
          buflen,              /* Buffer length                   */
```

```
          buffer,              /* Message buffer                  */
          &CompCode,           /* Completion code                 */
          &Reason);            /* Reason code                     */

    /* Report reason, if any */
    if (Reason != MQRC_NONE)
    {
      printf("MQPUT ended with reason code %ld\n", Reason);
    }
  }
  else
    /* Close the message loop if null line in file */
    CompCode = MQCC_FAILED;
}

/*****************************************************************/
/*                                                               */
/*               Get and display the reply messages              */
/*                                                               */
/*****************************************************************/
CompCode = OpenCode;          /* Only if the reply queue is open  */
gmo.WaitInterval = 60000;     /* 1 minute limit for first reply   */
while (CompCode != MQCC_FAILED)
{
  gmo.Options = MQGMO_WAIT                /* Wait for replies    */
    + MQGMO_CONVERT                       /* Request conversion  */
    + MQGMO_ACCEPT_TRUNCATED_MSG;         /* Can truncate        */

  /** specify representation that is required **/
  md.Encoding = MQENC_NATIVE;
  md.CodedCharSetId = MQCCSI_Q_MGR;

  /*****************************************************************/
  /*                                                               */
  /*   In order to read the messages in sequence, MsgId and        */
  /*   CorrelID must have the default value. MQGET sets them       */
  /*   to the values in the message it returns, so re-initialize   */
  /*   them before every call.                                     */
  /*                                                               */
  /*****************************************************************/
  memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
  memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));

  /*****************************************************************/
  /*                                                               */
  /*                       Get reply message                       */
  /*                                                               */
  /*****************************************************************/
  buflen = MAX_BUFFER_LEN;
  MQGET(Hcon,                 /* Connection handle                 */
```

```
        Hreply,                /* Object handle for reply         */
        &md,                   /* Message descriptor              */
        &gmo,                  /* GET options                     */
        buflen,                /* Buffer length                   */
        buffer,                /* Message buffer                  */
        &replylen,             /* Reply length                    */
        &CompCode,             /* Completion code                 */
        &Reason);              /* Reason code                     */
  gmo.WaitInterval = 15000;    /* 15 second limit for others      */

  /* Report reason, if any */
  switch(Reason)
  {
    case MQRC_NONE:
      break;
    case MQRC_NO_MSG_AVAILABLE:
      printf("no more replies\n");
      break;
    default:
      printf("MQGET ended with reason code %ld\n", Reason);
      break;
  }

  /*********************************************************************/
  /*                                                                   */
  /*                    Display reply message                          */
  /*                                                                   */
  /*********************************************************************/
  if (CompCode != MQCC_FAILED)
  {
    if (replylen < buflen)      /* Terminate (truncated) string   */
      buffer[replylen] = '\0';
    else
      buffer[buflen] = '\0';

    printf ( "\n" );            /*** Separator ***/
    parse_buffer () ;

    if (md.MsgType == MQMT_REPORT)     /* Display report feedback */
      printf("  report with feedback = %ld\n", md.Feedback);
  }
}

/*********************************************************************/
/*                                                                   */
/*     Close server queue - program terminated if open failed        */
/*                                                                   */
/*********************************************************************/
C_options = 0;                  /* No close options                */
```

```
MQCLOSE(Hcon,                    /* Connection handle            */
        &Hobj,                   /* Object handle                */
        C_options,
        &CompCode,               /* Completion code              */
        &Reason);                /* Reason code                  */

/* Report reason, if any */
if (Reason != MQRC_NONE)
{
  printf("MQCLOSE (server) ended with reason code %ld\n", Reason);
}

/*****************************************************************/
/*                                                               */
/*              Close reply queue - if it was open               */
/*                                                               */
/*****************************************************************/
if (OpenCode != MQCC_FAILED)
{
  C_options = MQCO_DELETE;         /* Delete dynamic queue        */
  MQCLOSE(Hcon,                    /* Connection handle           */
          &Hreply,                 /* Object handle               */
          C_options,
          &CompCode,               /* Completion code             */
          &Reason);                /* Reason code                 */

  /* Report reason, if any */
  if (Reason != MQRC_NONE)
  {
    printf("MQCLOSE (reply) ended with reason code %ld\n", Reason);
  }
}

/*****************************************************************/
/*                                                               */
/*       Disconnect from MQM (unless previously connected)       */
/*                                                               */
/*****************************************************************/
if (CReason != MQRC_ALREADY_CONNECTED)
{
  MQDISC(&Hcon,                    /* Connection handle           */
         &CompCode,                /* Completion code             */
         &Reason);                 /* Reason code                 */

  /* Report reason, if any */
  if (Reason != MQRC_NONE)
  {
    printf("MQDISC ended with reason code %ld\n", Reason);
  }
}
```

```
   printf("mycontrol end\n");
   return(0);
}
parse_buffer ()
{

/*********************************************************************/
/*                                                                   */
/*  parse_buffer : The message buffer contains a long string of      */
/*                 parameters and values that need to be separated   */
/*                                                                   */
/*********************************************************************/

#define UPDATING_PARAMETER 0
#define UPDATING_VALUE 1
#define HAS_PARAMETERS 2
#define NO_PARAMETERS 3

int Cbuffer = 0 ;
int CParameter = 0 ;
int CValue = 0 ;
char c ;
int flag ;
int flag2 ;

flag = UPDATING_PARAMETER ;
flag2 = NO_PARAMETERS ;
parameter[0] ='\0' ;

while  ( c = buffer [Cbuffer]  )
{
   if ( c == '(' )
   {
         parameter [ CParameter] = '\0' ;
         display_parameter () ;
         flag = UPDATING_VALUE ;
         flag2 = HAS_PARAMETERS ;
   }
   else if ( c == ')' )
   {
         value [CValue ] = '\0' ;
         display_value ();
         printf ( "%24s : %s \n" , parameter , value );
         CParameter = 0 ; CValue = 0 ;
         flag = UPDATING_PARAMETER ;
   }
   else
   {
         if ( flag == UPDATING_PARAMETER )
         {
```

```
            parameter[CParameter] = c ;
            CParameter++ ;
            }
            if ( flag == UPDATING_VALUE )
            {
            value[CValue] = c ;
            CValue ++ ;
            }
      }
      Cbuffer ++ ;
}
      if  ( flag2  == NO_PARAMETERS )
      {
      parameter[Cbuffer] = '\0' ;
      printf ( "%s\n", parameter );
      }
}

display_parameter ()
{
/*********************************************************************/
/*                                                                   */
/* display_parameter : The message may contain a CSQM message code   */
/*                     that is thrown away. Certain parameters       */
/*                     contain values and require additional         */
/*                     processing as they don't follow the           */
/*                     convention for parameter-value pairs.         */
/*                                                                   */
/*********************************************************************/

#define MAX_TOKENS  3
char temp_area [ 1000 ] ;
char * s1 ;
int parmeter_length ;
int i ;
int in_text ;
char * tok [ MAX_TOKENS ] ;

      /*** Remove CSQM identifier from the message ***/
      temp_area [0] = '\0' ;

      if ( s1 = strstr ( parameter , "CSQM" ) )
      {
         strcpy ( temp_area , s1+8 );
         strcpy ( parameter , temp_area );
         temp_area [0] = '\0' ;
      }

      if ( s1 = strstr ( parameter, "NOTRIGGER" ) )
      {
```

```
      strcpy ( temp_area , s1+9 );
      strcpy ( parameter , temp_area );
      temp_area [0] = '\0' ;
      printf ( "%24s :\n", "NOTRIGGER" );
   }
   if ( s1 = strstr ( parameter, "TRIGGER" ) )
   {
      strcpy ( temp_area , s1+7 );
      strcpy ( parameter , temp_area );
      temp_area [0] = '\0' ;
      printf ( "%24s :\n", "TRIGGER" );
   }
   if ( s1 = strstr ( parameter, "NOSHARE" ) )
   {
      strcpy ( temp_area , s1+7 );
      strcpy ( parameter , temp_area );
      temp_area [0] = '\0' ;
      printf ( "%24s :\n", "NOSHARE" );
   }
   if ( s1 = strstr ( parameter, "SHARE" ) )
   {
      strcpy ( temp_area , s1+5 );
      strcpy ( parameter , temp_area );
      temp_area [0] = '\0' ;
      printf ( "%24s :\n", "SHARE" );
   }
   if ( s1 = strstr ( parameter, "NOHARDENBO" ) )
   {
      strcpy ( temp_area , s1+10 );
      strcpy ( parameter , temp_area );
      temp_area [0] = '\0' ;
      printf ( "%24s :\n", "NOHARDENBO" );
   }
   if ( s1 = strstr ( parameter, "HARDENBO" ) )
   {
      strcpy ( temp_area , s1+8 );
      strcpy ( parameter , temp_area );
      temp_area [0] = '\0' ;
      printf ( "%24s :\n", "HARDENBO" );
   }
}

display_value () {
/*********************************************************************/
/*                                                                   */
/* display_value : Removes spaces before first character and after   */
/*                 the last character. Spaces in the text are        */
/*                 preserved.                                        */
/*                                                                   */
/*********************************************************************/
```

```
int i , j ;
char temp_value [1000 ] ;
int len ;

   temp_value [0] = '\0' ;
   len = strlen ( value );

   /* Go forward to first non-space character  */
   for ( i = 0 ; (i < len) && (value [i] == ' ') ; i++ )
      ;

   strcpy ( temp_value , value +i  ) ;
   strcpy ( value , temp_value );

   /* Go backward to last non-space character */
   for ( i = len-1 ; (i > 0 ) && (value[i] == ' ' ) ; i-- )
   {

   }
   /*  What if value contains only spaces? */
   if ( i > 0 )
   {
      value [i+1] = '\0' ;
   }

}
```

Once the changes necessary to customize the application for your site are made to the code, you're ready for the first 'cut' of the remote administration application. Compile the changed source code using the command:

```
cc -g -Aa -D_HPUX_SOURCE -D_REENTRANT amqsreq0.c -lmqm
➤  -o mycontrol
```

The executable created is called **mycontrol**, and it can be invoked as illustrated in the example below.

INVOKING MYCONTROL

```
hp$mycontrol MVMQ HPMQ
mycontrol  start
display qmgr all

CSQN205I   COUNT=       3, RETURN=00000000, REASON=00000000

                 QMNAME : MVMQ
                  DESCR : MVMQ, IBM MQSeries for MVS/ESA - V1.2
```

```
   PLATFORM : MVS
   CPILEVEL : 100
   CMDLEVEL : 120
      CCSID : 500
    MAXPRTY : 9
    MAXMSGL : 4194304
     SYNCPT : AVAILABLE
   COMMANDQ : SYSTEM.COMMAND.INPUT
      DEADQ : MQT1.DEAD.QUEUE
    TRIGINT : 999999999
   MAXHANDS : 256
   AUTHOREV : DISABLED
   INHIBTEV : DISABLED
    LOCALEV : DISABLED
   REMOTEEV : DISABLED
   STRSTPEV : ENABLED
    PERFMEV : DISABLED
   DEFXMITQ : MVMQ.DEFXMIT.QUEUE

CSQ9022I ]MVM CSQMDRTS ' DISPLAY QMGR' NORMAL COMPLETION
```

The application can be refined further to achieve better formatting and parsing, as required.

SUMMARY

This article has discussed the remote administration of MQSeries for MVS/ESA from MQSeries for HP-UX Version 5, presenting a working set-up. Except for a few basic start-up issues, most MQSeries-related activities can be managed using remote administration.

Despite the benefits of remote administration, the limitation of having to use the default queue manager is likely to be a problem at many installations, and this will probably result in many writing their own custom remote administration applications. There are other reasons why a custom administration application may be implemented, such as handling the presentation of message data and implementing ‘decision layers’. This article also presents a sample MQSeries application that provides a framework for your own custom application.

Ashish Joshi
Consultant (USA)

Guidelines for MQSeries for OS/390 users

This article provides a series of guidelines for MQSeries for OS/390 users. I've compiled them from my notes and procedures that I have developed and implemented. The article specifically relates to the MQSeries Message Queuing Service for MVS Version 1.4.1 product that's supplied with ProductPac for OS/390 V2R4.

INSTALLATION

Install MQSeries on OS/390 V2R4 via SMP/E. The product runs in the MVS SMP/E environment for OS/390 V2R4. Note that you should remove the middle qualifier and replace the DDDEFs for the relevant datasets.

NAMING CONVENTION

Naming convention is an important issue for MQSeries users and a number of systems have been adopted by various organizations. I recommend the one set out below for OS/390 users.

MQSeries queue managers

I use a naming convention comprising two types of queue manager name. The first type of name takes the form *MQxn*, where:

- *x* is the LPAR id.
- *n* is used to identify the role of the system: *n=0* is a testing system dedicated to verifying system changes, *n=1* is an acceptance system dedicated to application changes, etc.

Hence, *MQx1* is a queue manager with LPAR id *x*, and the value *n=1* signifies an acceptance system.

If the queue manager's name takes the form *MQPn*, then *P* indicates a production system and *n* is a number that identifies an application dedicated to a specific queue manager. Subsequent production systems would then use queue manager names *MQP2*, *MQP3*, etc for additional applications that require dedicated queue managers. Production queue

manager names are, therefore, not dependent on LPAR ids.

Subsystem names

Use the same naming convention for subsystems as described above for queue managers.

Subsystem command prefix

The most common convention is to prefix all subsystem names with a forward slash ('/'). If you'd like to adopt a different convention, a list of characters that can be used is available in the *MQSeries Systems Management Guide*.

Procedure names

In keeping with the convention for naming queue managers, *MQxnMSTR* is a master address space, where *x* is the LPAR id and *n* signifies a test, acceptance, or production environment. The suffix *MSTR* is mandatory.

Similarly, *MQxnCHIN* is a channel initiator, where *x* is the LPAR id and *n* signifies a test, acceptance, or production environment. As before, the suffix *CHIN* is mandatory.

Queue names

The default queue definitions supplied with the system should be left unchanged.

Queue manager-specific queue names

- Local queues – use the convention: *QL.Subsystem_Name.**.

  For example:

  ```
  QL.MQxn.*
  ```

Project-specific queue names

- Local queues – use the convention: *QL.Project_id.**.

  For example:

  ```
  QL.APPL.*
  ```

- Remote queues – use the convention: *QR.project id.**.

  For example:

  ```
  QR.APPL.*
  ```

- Alias queues – use the convention: *QA.project id.**.

  For example:

  ```
  QA.APPL.*
  ```

- Processes – use the convention: *PR.project id.**.

  For example:

  ```
  PR.APPL.*
  ```

Destination-specific queue definitions

- The convention for receiver channels (type RCVR) is:

  ```
  CH.Sender_id.TO.Receiver_id.Protocol_id
  ```

  For example:

  ```
  CH.4D100.TO.MQxn.LU
  ```

  Here the channel is to receive messages on *MQxn* from queue manager *4D100*.

- The convention for a sender channel (type SDR) is:

  ```
  CH.Receiver_id.TO.Sender_id.Protocol_id
  ```

  For example:

  ```
  CH.MQC0.TO.20003.LU
  ```

  Here the channel is to send messages to *20003* from *MQxn*.

- The convention for an XMIT local queue is:

  ```
  QL.XMIT.TO.Remote_Queue_Manager_Name
  ```

  For example:

  ```
  QL.XMIT.TO.17817
  ```

  is an XMIT queue to queue manager *17817*.

STARTING AND STOPPING TASKS

Below are commands that can be used for starting and stopping various types of task using either the console or SDSF.

Starting and stopping *MQxnMSTR* tasks using SDSF:

```
/ /MQxn START QMGR PARM(CSQZMQxn)
/ /MQxn STOP  QMGR
```

Starting and stopping *MQxnMSTR* tasks using the console:

```
/MQxn START QMGR PARM(CSQZMQxn)
/MQxn STOP  QMGR
```

Starting and stopping *MQxnCHIN* tasks (only) using SDSF:

```
/ /MQxn START CHINIT PARM(CSQXMQxn)
/ /MQxn STOP  CHINIT
```

Starting and stopping *MQxnCHIN* tasks (only) using the console:

```
/MQxn START CHINIT PARM(CSQXMQxn)
/MQxn STOP  CHINIT
```

CLIENT ATTACHMENT FEATURE

This is an MQSeries component that facilitates Distributed Queuing Management (DQM) and is necessary to allow an MQSeries application running on an MQSeries client to interact with one or more MQSeries servers and connect to their queue managers by means of a communication protocol. It comes free with both MQSeries for MVS and OS/390, and it can be used to avoid the cost of multiple MQSeries installs, as well as the associated licence cost at the client, once it's installed on the server (the mainframe) and the communication architecture described above is implemented.

The MQSeries Client Attachment Feature can be ordered as a separate FMID and supplied as part of the base MQSeries V1.1.4 product when ordered with OS/390.

SYSTEM UPDATES

After installing MQSeries on your S/390, the items listed below need to be updated on your system.

PROCLIB

SYSx.PROCLIB:

- *MQxnMSTR.*

  The queue manager started task (set *TIME=NOLIMIT*).

- *MQxnCHIN*.

  The channel initiator started task.

PARMLIB

SYS1.PARMLIB

- *APPCPMxx*

  The *MVSMQxx ACBNAME* definition.

- *IEFSSNxx*

  Subsystem names and the command prefix.

- *LNKLSTxx*

  MQM.SCSQAUTH/SCSQLINK/SCSQSNLE should be added.

- *PROGxx*

  Ensure *MQM.SCSQANLE* is APF-authorized.

- *SCHEDxx*

  The *PPT* entry for *CSQYASCP* module.

- *IEAIPSxx*

  The *SRM* parameters for MQ started tasks.

- *IEAICSxx*

  The *SRM* parameters for MQ started tasks.

RACF

- *STASK* entries should be added for new *MQ** started tasks.
- The following items in your datasets' profile set-up:

– *MQM.**

  MQ subsystems.

– *MQARCH.**

  MQ subsystems log archiving.

– *MQxn.**

  MQ subsystem PAGE and BOOTSTARP datasets.

- The profiles require access list entries for *MQxnMSTR* tasks (in other words, *UPDATE* the *MQM.** profiles and *ALTER* the *MQARCH.** profile). *READ* access to *DASDVOL* is required at installation for allocating LOGs, etc.

VTAM/network

- Add a *SideInfo* entry to the *APPC.APPCSI* dataset and stop and restart the *APPC* task.
- *MVSMQxn ACBNAMEs* need to be defined to VTAM in the *VTAMLST(APPLxxxx)* member.
- Nodenames need to be defined to provide links to remote MQ subsystems.
- All required nodes should be added and activated via the *VTAMLST(ATCCONxx)* member.
- The APPC/LU62 built-in protocol should be used to communicate between MQ subsystems.
- Further customization is required if TCP/IP is also to be used to communicate between MQ subsystems.

ISPF

- Set up the dynamic (LIBDEF) MQ/ISPF interface by creating a new clist called ‘MQ’. Ensure that the Hlq of all datasets with references in this clist are the same.
- Add *MQ* to a clist library in the *SYSPROC* concatenation.

- Catalogue *MQM.SCSQ** ISPFLIBs to ensure their availability in the *MQ* clist.
- Include *MQM.SCSQEXEC* dataset in the system *SYSEXEC* or *SYSPROC* concatenations.
- Copy base members supplied in the *MQM.SCSQPROC* dataset to create procedures that are specific to LPAR and MQ subsystems. Update and further customize these procedures to suit each MQ subsystem's requirement.

Additional requirements

- Request SMS disk space for the *MQARCH.MQ** log archive's datasets and define an SMS rule for it.
- Define and catalogue a new Hlq alias for *MQARCH*.

MQ CUSTOMIZATION

Copy base options members *CSQ4ZPRM/CSQ4XPRM* supplied in library *SCSQAUTH* and allocate a new dataset called *MQM.PRMMODS* to contain the updated options modules created for each MQ subsystem. Concatenate *MQM.PRMMODS* above any other loadlibs in the JCL steplibs of the started task procedure.

The options modules must be specified on the **START** command. For example, the *MQPn* start commands are:

```
/MQnx START QMGR PARM(CSQZMQnx)
/MQnx START CHI PARM(CSQXMQnx)
```

Note that, if the PARM is omitted, Lmods are used that have default values set by IBM. This would probably cause errors.

Update the base batch adapter options module *CSQBDEFV* that is supplied in library *SCSQAUTH* so that it contains the MQ subsystem name and copy it to *MQM.PRMMODS*. Application programs connect to the subsystem defined in this module, if the subsystem is not specified in the *MQCONN* call.

This module cannot be renamed, though it may contain different MQ subsystem names.

New members should be created in *SCSQPROC* dataset:

- *CSQ4MQ***

  These are subsystem-specific queue definitions. Rename members to identify the subsystem suffix and change the subsystem reference to the correct name.

- *CSQ4CHNL*

  The start channel initiator, command server, and channel definitions. Change the subsystem reference to the correct name.

- *CSQ4APPL*

  Application-specific queue definitions.

Note that, to keep customization to a minimum, additional applications and relative queue definitions require only the addition of a member similar to existing ones.

This article concludes in next month's issue of *MQ Update*.

Saida Davies (UK)

Contributing to *MQ Update*

Contributions to *MQ Update* may be sent to the editor, Harry Lewis, at: *MQ Update, Xephon, 27-35 London Road, Newbury, Berkshire RG14 1JL, UK*. You may also e-mail articles to *harryl@xephon.com*. For more information about contributing, please download a copy of *Notes for contributors* from Xephon's Web site at *www.xephon.com.*

IBM has unveiled VisualAge Interspace Version 6.0, which now includes MQSeries Clients V5 and CICS Universal Clients Version 3. Formerly known as Planetworks Interspace, VisualAge Interspace is aimed at designers working with middleware. The development framework integrates desktop development tools with MQSeries, CICS, Encina, and VisualAge Generator, integrating front-end and back-end applications and providing productivity tools in the application development life-cycle.

Out now, prices start at US$2,695.

The company also announced that it's to resell Extricity Software's AllianceSeries of B2B e-commerce software, adding it to its MQSeries range of integration and collaboration tools. AllianceSeries is an XML-based family of products that use the Internet to automate business processes.

For further information contact your local IBM representative.

* * *

Candle has announced that it's incorporated Tibco's TIB/Rendezvous publish/subscribe messaging technology APIs into CandleNet product. This is a family of e-business lifecycle products that includes the Service Provider Platform, which provides workload balancing, PKI-based security, QoS, and change management for Web applications, also offering performance monitoring for MQSeries, Web applications, and database servers. The product also helps developers build Web sites, Internet portals, and e-business back-end integration systems. Tibco's software manages the delivery of real-time information to a subscriber-based audience. Each message traverses the network only once, but the software carries it to all subscribers on the network.

Roma's support for Tibco's APIs is available for field test, with general availability expected in the coming quarter.

In a separate announcement, Candle stated that it's re-aligning its Enterprise Computing Group to make better use of its system, application, and middleware management technologies in the e-business market. The re-alignment includes moving MQSeries management to the ECG.

For further information contact:
Candle Corp, 2425 Olympic Blvd, Santa Monica, CA 90404, USA
Tel: +1 310 829 5800
Fax: +1 310 582 4287
Web: http://www.candle.com

Candle Ltd, 1 Archipelago, Lyon Way, Frimley, Camberley, Surrey GU16 5ER, UK
Tel: +44 1276 4147000
Fax: +44 1276 414777

* * *