45

MQ update

March 2003

In this issue

MQ Update

Published by

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

North American office

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

Editor

Madeleine Hudson
E-mail: MadeleineH@xephon.com

Subscriptions and back-issues

A year's subscription to *MQ Update*, comprising twelve monthly issues, costs £255.00 in the UK; $380.00 in the USA and Canada; £261.00 in Europe; £267.00 in Australasia and Japan; and £265.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the July 1999 issue, are available separately to subscribers for £22.50 ($33.75) each including postage.

Contributions

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

MQ Update on-line

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

MQSeries First Steps for OS/390

INTRODUCTION

MQSeries for WinNT/2000/XP has an extra tool called ‘First Steps’. It allows the user to specify all fields of MQMD and other MQSeries options when using MQPUT or MQGET to put/get a message to/from a queue. It is a perfect tool for beginners. Programmers can see the effects of various MQSeries options without needing to write a program. It is also useful for testers or administrators when there is a need quickly to generate a message with specified values in MQMD.

This great tool is unfortunately available only for Windows users. I needed something similar on our mainframe. In Poland we have a saying – need is the mother of invention – so I have written a program I’ve called ‘MQSeries First Steps for OS/390’. It is an ISPF panel-based utility that has some of the features found in the original First Steps.

My version is not as sophisticated as the original program. It implements only two (extremely useful!) MQSeries functions – MQPUT and MQGET. I have provided the facility to MQGET a message and see its full contents and MQPUT a message with specified MQMD fields.

I have chosen REXX as my programming tool. Since the MQSeries API is not normally available for REXX I had to use Robert Harris’ SupportPac (MA18, available from *http://www-4.ibm.com/software/ts/mqseries/txppacs/ma18.html*), which provides a comfortable way of using MQSeries functions in REXX.

INSTALLATION

MQSeries First Steps for OS/390 is composed of several libraries:

- REXX (the REXX code).
- PANELS (panels definitions).

- MSG (messages).
- TABLES (ISPF tables, they contain mainly commands and key list definitions).

I prepared all of them as XMIT files. You will find them on the Xephon Web site at www.xephon.com/extras/msgs.xmt, panels.xmt, rexx.xmt, and tables.xmt. The REXX code is available from www.xephon.com/extras/xmit.txt.

To install my utility, follow the steps detailed below.

- Send all four XMIT files to your host. Use binary transfer and store them as sequential data sets (RECFM=FB, LRECL=80, BLKSIZE=3120).
- Use the TSO command **RECEIVE** to unpack each of them. The syntax is:

  ```
  RECEIVE INDSN('XMIT.FILE.NAME')
  ```

  When prompted, enter the destination dataset name ‘DSN('hlq.type')’, where ‘type’ is REXX, PANELS, MSG, and TABLES respectively, and ‘hlq’ is a high-level qualifier of your choice.
- Edit ‘hlq.REXX(MAIN)’ and change the five variables you will find in lines three to seven:
  - panels – the name of the library where you have put the panels (hlq.PANELS)
  - tables – the name of the library where you have put the tables (hlq.TABLES)
  - msgs – the name of the library where you have put the messages (hlq.MSGS)
  - QMgr – the name of the default queue manager that will be used by the utility (optional; if left blank you will need to supply it when you run First Steps)
  - QName – the name of the default queue that will be used by the utility (optional; if left blank you will need to supply it when you run First Steps).

I use LIBDEF so there is no need to add my panels, messages, and tables to ISPF concatenations. Just change the first three variables mentioned above and the utility is ready to use.

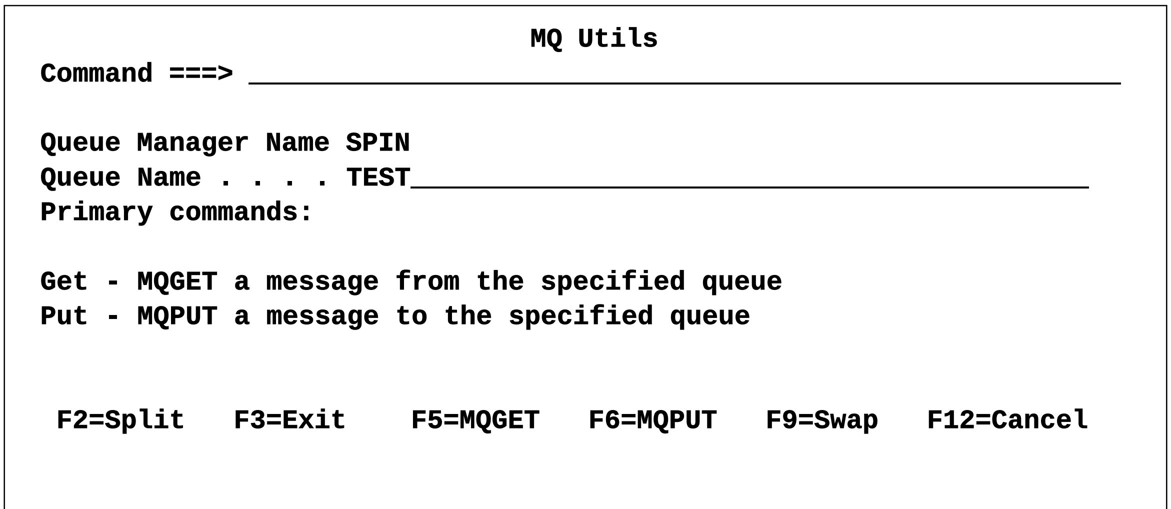
USE

To start MQSeries First Steps execute **userid.REXX(MAIN)**. You will see the panel shown in Figure 1.

If you have entered the values of the QMgr and QName variables, they will be shown at the top of the panel as ‘Queue Manager Name’ and ‘Queue Name’ accordingly. Otherwise they will be blank and you need to enter them now.

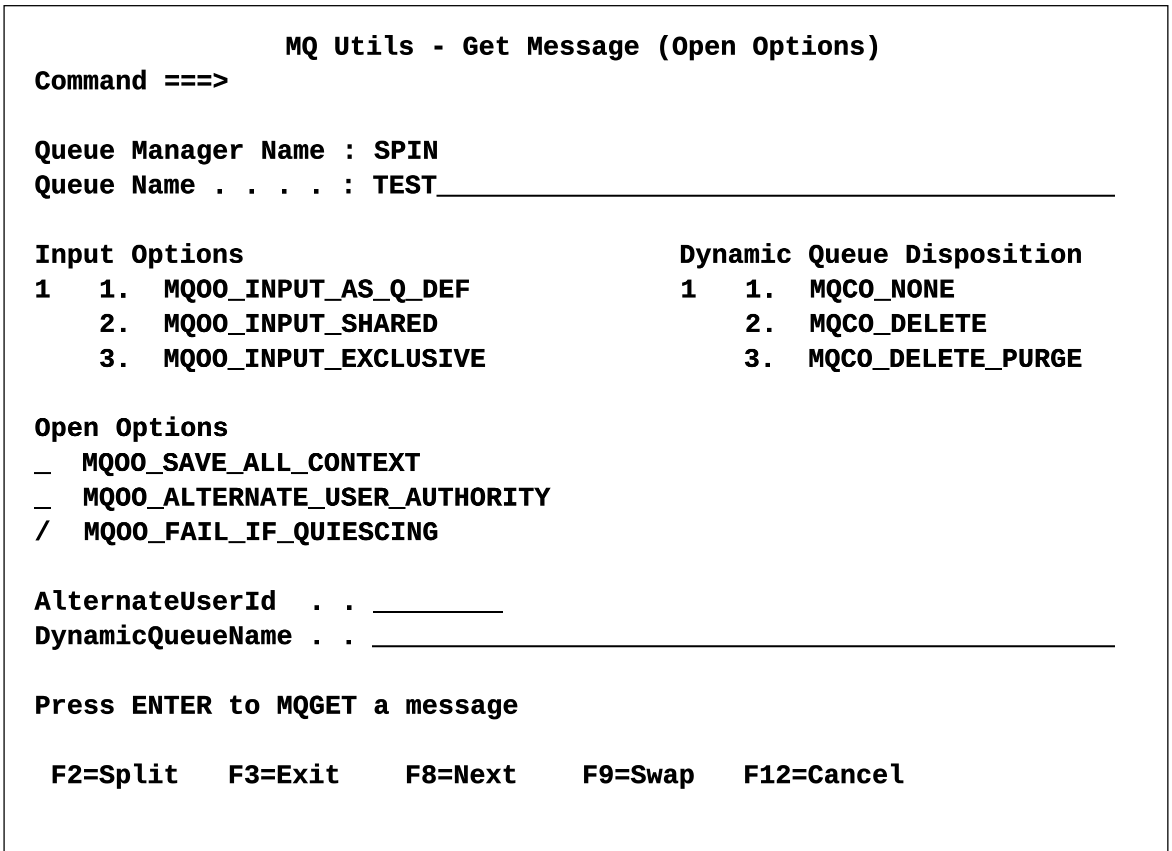
There are two commands available, **GET (PF5) - MQGET** and **PUT (PF6) - MQPUT**. Enter them at the command line or use the appropriate PF key. To exit the panel press PF3 or PF12.

In all of my panels I use standard ISPF input types: normal input field (you provide any value you want), multiple choice list (tick each choice you want by inserting ‘/’ (slash) next to the desired option), and single choice list (enter a number corresponding to the desired option). Some fields will have default values – you can change them of course.



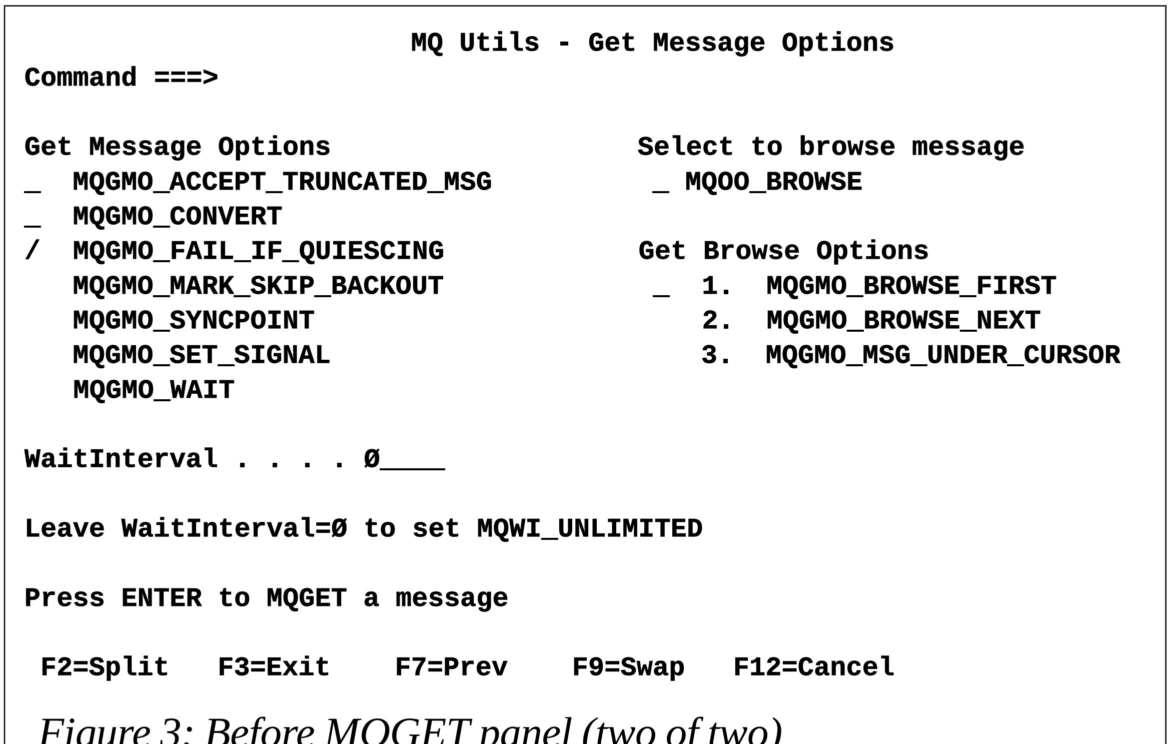
```
                              MQ Utils
Command ===> ______________________________________________________

Queue Manager Name SPIN
Queue Name . . . . TEST__________________________________________
Primary commands:

Get - MQGET a message from the specified queue
Put - MQPUT a message to the specified queue


 F2=Split   F3=Exit    F5=MQGET   F6=MQPUT   F9=Swap   F12=Cancel
```

Figure 1: Main panel



```
                MQ Utils - Get Message (Open Options)
Command ===>

Queue Manager Name : SPIN
Queue Name . . . . : TEST__________________________________________

Input Options                           Dynamic Queue Disposition
1   1.  MQOO_INPUT_AS_Q_DEF             1   1.  MQCO_NONE
    2.  MQOO_INPUT_SHARED                   2.  MQCO_DELETE
    3.  MQOO_INPUT_EXCLUSIVE                3.  MQCO_DELETE_PURGE

Open Options
_  MQOO_SAVE_ALL_CONTEXT
_  MQOO_ALTERNATE_USER_AUTHORITY
/  MQOO_FAIL_IF_QUIESCING

AlternateUserId  . . ________
DynamicQueueName . . _______________________________________________

Press ENTER to MQGET a message

 F2=Split   F3=Exit    F8=Next    F9=Swap   F12=Cancel
```

Figure 2: Before MQGET panel (one of two)



```
                        MQ Utils - Get Message Options
Command ===>

Get Message Options                    Select to browse message
_  MQGMO_ACCEPT_TRUNCATED_MSG           _ MQOO_BROWSE
_  MQGMO_CONVERT
/  MQGMO_FAIL_IF_QUIESCING              Get Browse Options
   MQGMO_MARK_SKIP_BACKOUT              _  1.  MQGMO_BROWSE_FIRST
   MQGMO_SYNCPOINT                         2.  MQGMO_BROWSE_NEXT
   MQGMO_SET_SIGNAL                        3.  MQGMO_MSG_UNDER_CURSOR
   MQGMO_WAIT

WaitInterval . . . . Ø____

Leave WaitInterval=Ø to set MQWI_UNLIMITED

Press ENTER to MQGET a message

 F2=Split   F3=Exit    F7=Prev    F9=Swap   F12=Cancel
```

Figure 3: Before MQGET panel (two of two)

GET

The GET panel is shown in Figure 2. Supply the options you wish (the queue manager name and queue name cannot be changed at this point) and press PF8 to go to the MQGMO options panel (shown in Figure 3), or press ENTER to MQGET a message with default ‘get message’ options.

You can press PF7 to review the previous panel or hit ENTER to MQGET a message.

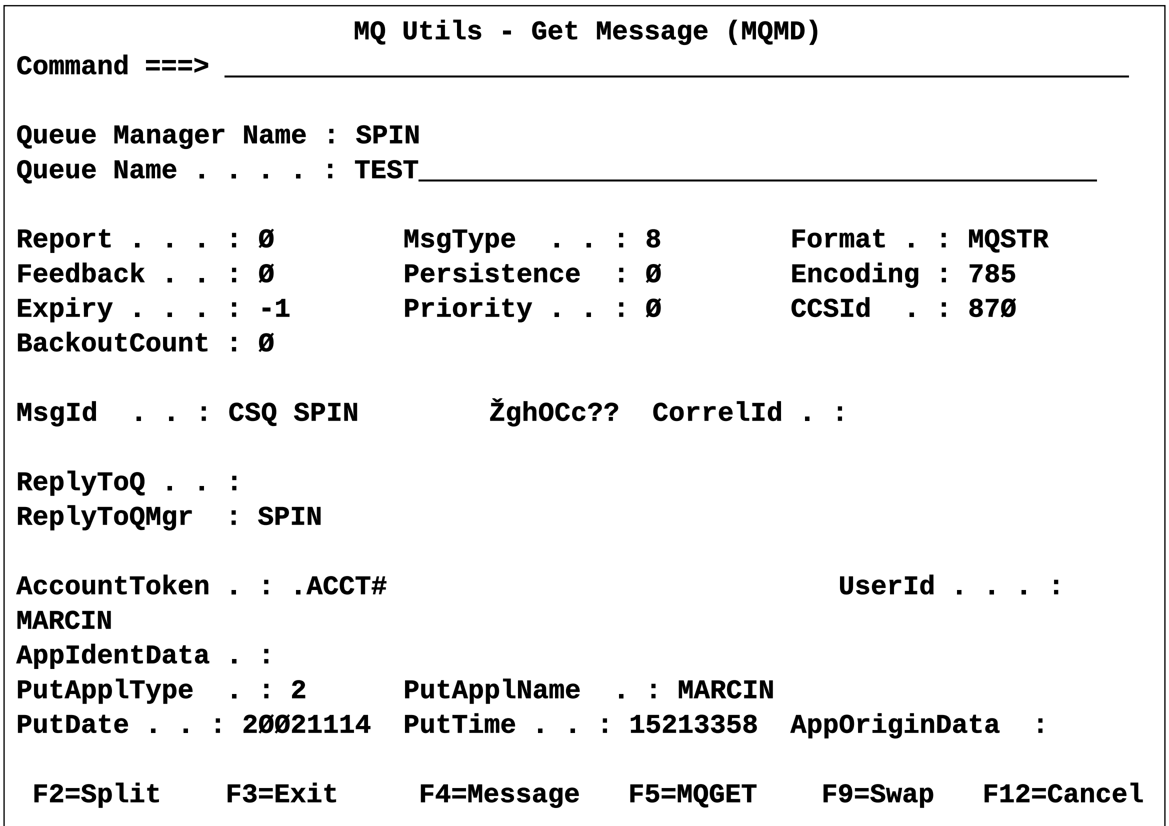
If there is a message on the specified queue you will see a panel with application data (Figure 4). To see the MQMD of the message read, press PF4 (Figure 5). If the queue is empty you will get an appropriate ISPF message. To return to the main panel press PF3.

PUT

After entering the PUT command from the main panel you will be presented with several panels that allow you to enter the message (both application data part and MQMD). Those panels are shown in Figures 6 to 10. You can move forwards and backwards between them using the PF7 and PF8 keys. To MQPUT your message hit ENTER on any of those panels.

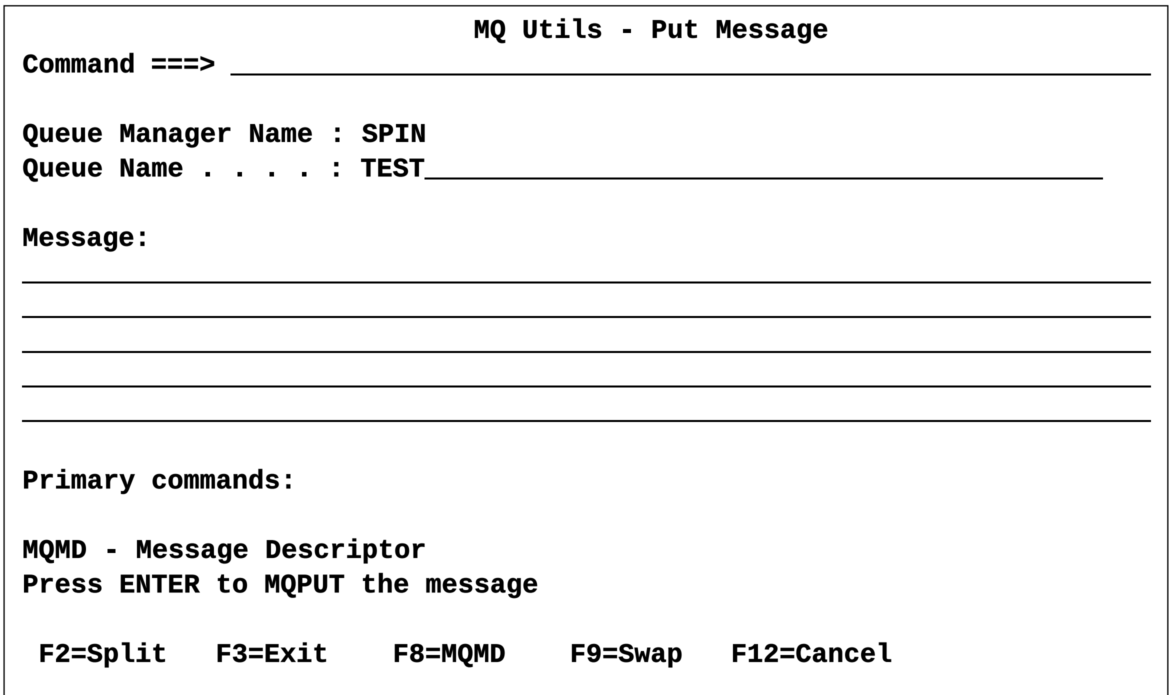
```
                      MQ Utils - Get Message
Command ===> ____________________________________________________

Queue Manager Name : SPIN
Queue Name . . . . :
TEST__________________________________________

Message:
test

Primary commands:

MQMD - Message Descriptor
                            |------------------|
                            | MQGET successful |
                            |------------------|
 F2=Split   F3=Exit    F4=MQMD    F9=Swap   F12=Cancel
```

Figure 4: After MQGET panel (application data part)



```
                        MQ Utils - Get Message (MQMD)
Command ===> ________________________________________________________

Queue Manager Name : SPIN
Queue Name . . . . : TEST__________________________________________

Report . . . : Ø        MsgType  . . : 8        Format . : MQSTR
Feedback . . : Ø        Persistence  : Ø        Encoding : 785
Expiry . . . : -1       Priority . . : Ø        CCSId  . : 87Ø
BackoutCount : Ø

MsgId  . . : CSQ SPIN        ŽghOCc??  CorrelId . :

ReplyToQ . . :
ReplyToQMgr  : SPIN

AccountToken . : .ACCT#                            UserId . . . :
MARCIN
AppIdentData . :
PutApplType  . : 2      PutApplName  . : MARCIN
PutDate . . : 2ØØ21114  PutTime . . : 15213358  AppOriginData  :

 F2=Split    F3=Exit     F4=Message   F5=MQGET    F9=Swap   F12=Cancel
```

Figure 5: After MQGET panel (MQMD part)



```
                           MQ Utils - Put Message
Command ===> _________________________________________________________

Queue Manager Name : SPIN
Queue Name . . . . : TEST__________________________________________

Message:
______________________________________________________________________
______________________________________________________________________
______________________________________________________________________
______________________________________________________________________
______________________________________________________________________

Primary commands:

MQMD - Message Descriptor
Press ENTER to MQPUT the message

 F2=Split   F3=Exit    F8=MQMD    F9=Swap   F12=Cancel
```

Figure 6: Before MQPUT panel (Application Data Part)

```
                    MQ Utils - Put Message (MQMD - 1 of 4)
Command ===>

Report
   MQRO_EXCEPTION                      MQRO_NEW_MSG_ID
   MQRO_EXCEPTION_WITH_DATA            MQRO_PASS_MSG_ID
   MQRO_EXPIRATION                     MQRO_PASS_CORREL_ID
   MQRO_EXPIRATION_WITH_DATA
MQRO_COPY_MSG_ID_TO_CORREL_ID
   MQRO_COA                            MQRO_DEAD_LETTER_Q
   MQRO_COA_WITH_DATA                  MQRO_DISCARD_MSG
   MQRO_COD
   MQRO_COD_WITH_DATA              If no Report Option is chosen,
                                    MQRO_NONE will be set

ReplyToQMgr . . SPIN
ReplyToQ  . . . ______________________________________________

Press ENTER to MQPUT the message

 F2=Split   F3=Exit    F7=Message   F8=Next    F9=Swap   F12=Cancel
```

Figure 7: Before MQPUT panel (MQMD part – one of four)

```
    MQ Utils - Put Message (MQMD - 2 of 4)
Command ===>

Format                                MsgType
9   1.  MQMFT_COMMAND_1               1   1.  MQMT_DATAGRAM
    2.  MQMFT_COMMAND_2                   2.  MQMT_REQUEST
    3.  MQMFT_IMS                         3.  MQMT_REPLY
    4.  MQMFT_IMS_VAR_STRING              4.  MQMT_REPORT
    5.  MQMFT_CHANNEL_COMPLETED
    6.  MQMFT_DEAD_LETTER_HEADER      Encoding
    7.  MQMFT_NONE                    1   1.  MQENC_NATIVE
    8.  MQMFT_PCF                         2.  MQENC_INTEGER_MASK
    9.  MQMFT_STRING                      3.  MQENC_DECIMAL_MASK
    1Ø. MQMFT_XMITQ_Q_HEADER              4.  MQENC_FLOAT_MASK
    11. MQMFT_ADMIN
    12. MQMFT_TRIGGER                 CodedCharSetId
    13. MQMFT_EVENT                   1   1.  MQCCSI_Q_MGR
                                          2.  MQCCSI_EMBEDED

Press ENTER to MQPUT the message

 F2=Split   F3=Exit    F7=Prev    F8=Next    F9=Swap   F12=Cancel
```

Figure 8: Before MQ PUT panel (MQMD part – two of four)

```
                MQ Utils - Put Message (MQMD - 3 of 4)
Command ===>

Priority                             MsgId  . . .
____________________
                                     CorrelId . .
____________________
Persistence
3   1.  MQPER_PERSISTENT               Expiry . . . _____
    2.  MQPER_NOT_PERSISTENT
    3.  MQPER_PERSISTENCE_AS_Q_DEF

UserId . . . . . . . MARCIN______
AccountingToken  . . ________________________
ApplIdentityData . . ________________________

PutApplType  . . . : MQAT_MVS
PutApplName  . . . . MARCIN
PutDate  . . . . . . Ø2/11/15          PutTime  . . 15:15:Ø7
ApplOriginData . . . ____

Press ENTER to MQPUT the message

 F2=Split   F3=Exit    F7=Prev    F8=Next    F9=Swap   F12=Cancel
```

Figure 9: Before MQ PUT panel (MQMD part – three of four)

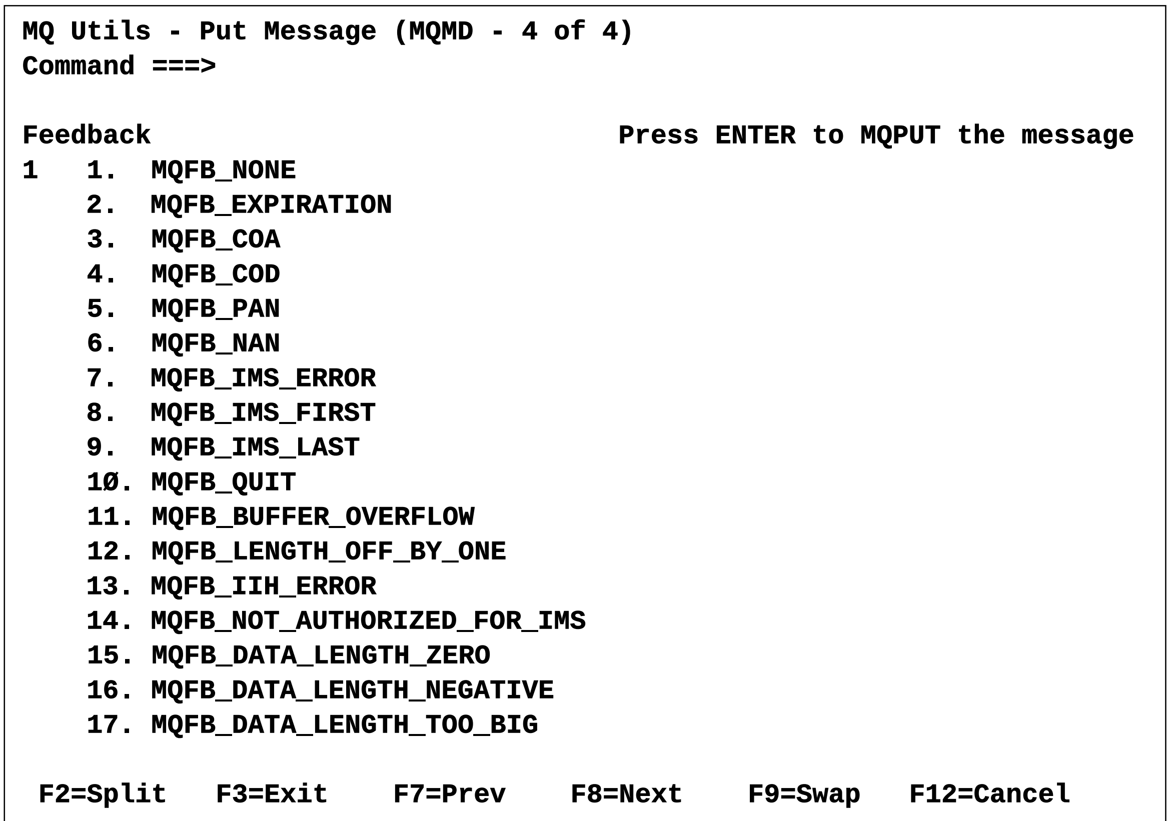
After you have MQPUT your message you will be returned to the main panel.

NOTES

My utility has several limitations. The reason for this is that I programmed only those features that were necessary for myself and my colleagues. So to be honest I must admit that my MQSeries First Steps for OS/390 is more of a beta version than a fully functional utility. It is, however, a good start. Maybe I will release an improved version in the future if there is sufficient interest.

The limitations are:

- Only MQGET and MQPUT are available.
- You cannot specify MQMD fields before MQGET (it means that getting a message based on its MSg-ID or Correl-ID is not



```
MQ Utils - Put Message (MQMD - 4 of 4)
Command ===>

Feedback                              Press ENTER to MQPUT the message
1   1.  MQFB_NONE
    2.  MQFB_EXPIRATION
    3.  MQFB_COA
    4.  MQFB_COD
    5.  MQFB_PAN
    6.  MQFB_NAN
    7.  MQFB_IMS_ERROR
    8.  MQFB_IMS_FIRST
    9.  MQFB_IMS_LAST
    1Ø. MQFB_QUIT
    11. MQFB_BUFFER_OVERFLOW
    12. MQFB_LENGTH_OFF_BY_ONE
    13. MQFB_IIH_ERROR
    14. MQFB_NOT_AUTHORIZED_FOR_IMS
    15. MQFB_DATA_LENGTH_ZERO
    16. MQFB_DATA_LENGTH_NEGATIVE
    17. MQFB_DATA_LENGTH_TOO_BIG

 F2=Split   F3=Exit    F7=Prev    F8=Next    F9=Swap   F12=Cancel
```

Figure 10: Before MQ PUT panel (MQMD part – four of four)

supported).

- MQCMIT and MQBACK are handled internally (depending of course on the values obtained from the panels).
- The message length is limited to 702 characters.
- No on-line help is available.
- Although I've handled most common errors there may still be some bugs.

I recommend using MQSeries First Steps for OS/390 for learning or testing purposes.

The XMIT files are meant to be copied to an OS/390 host and opened there. If the panels and tables are 'unpacked' and viewed as text, problems may occur when transferring them to the host.

The source of panels differs slightly from the final look – there are some special, non-printable characters that may be improperly converted.

Marcin Grabinski
Systems Engineer, SPIN (Poland)

Can WMQ really send non-text data?

I was recently preparing a hands-on course and pondered this question. More often then not, most of the data I've dealt with has been of a simple text or XML nature. Of course MQSeries can cope with non-text data, it's just that I'd never seen it or proved it for myself.

My self-imposed task was to 'prove' that MQSeries could be used to send both PDF and JPG files from one machine to another without them being corrupted along the way.

The task should successfully carry out the following:

- Read a PDF or JPG file.
- Store it as a message in a queue.
- Transfer the message to another queue manager on another machine.
- Retrieve that message from the queue.
- Store it as a file.
- Open the file to ensure that it can be read and has not been corrupted.

METHOD

First of all the supplied **amqsput** utility was tried:

```
amqsput RDEMO.Q MQDEMO1 < file1.pdf
```

Although this utility is limited to 100 bytes it's easy to alter this and recompile it. When this was tried, however, the 31 Kb file ended up as 32 small separate messages. Whenever **amqsput** encountered a carriage return/linefeed (hex characters '0D0A') it regarded it as the end of the message. In fact even if all 32 messages were to be reconstituted it would not contain the whole message. When **amqsput** encountered the hex character '1A0D' in the middle of the file it stopped scanning and finished.

The next utility I tried was the IBM SupportPac MA01. This very nice utility can do a lot of things and so this command was tried:

```
C:\>q -oRDEMO.Q -mMQDEMO1 -ffile1.pdf
MQSeries Q Program Version 3.Ø (written by Paul Clarke)
Connecting ...connected to 'MQDEMO1
```

Again, the result was very similar to **amqsput**. This time 30 messages were created but the program stopped at the same point.

The last utility to be tried was the IBM SupportPac known as IH03, which performed points one and two very efficiently. The utility uses a parameter file – here called *ih03pdf.txt* – as input; the important parameters are shown here.

For the header:

- qname="RDEMO.Q" – the name of the remote queue.
- qmgr="MQDEMO1" – the name of the local queue manager.
- msgcount="1" – write a single message.
- format= "   " – ensure it's not a text message.
- msgtype="8"   – message is a datagram.

For the filelist:

- *c:\file1.pdf* – the name of the file to be transported.

The remote queue was set up as:

```
DEF QR(RDEMO.Q) RQNAME(DEMO.Q) RQMNAME(MQDEMO2)
```

```
      XMITQ(MQDEMO2)
```

As I already had some utilities which were named similarly to the one supplied by IH03, I renamed mqput2 as ih03put. The syntax, therefore, to invoke the utility was:

```
   IhØ3put  ihØ3pdf.txt
```

This successfully added the message to the remote queue.

The IH03 command line utility is not written to read messages so the hunt was on to find another utility.

The supplied utility **amqsget** was tried, which of course gave an rc2080 message (MQRC_TRUNCATED_MSG_FAILED). Changing the 100 byte I/O area to a larger value meant that more of the binary message was read but as soon as a null (hex character ‘00’) was found the utility stopped.

After a lot of searching I decided that the only way to read data that contained ‘special’ characters was to read the whole lot as a ‘blob’ in one go.

A new utility called ‘mqgetbin’ was born, the code for which has been supplied.

To invoke it:

```
C:\>mqgetbin DEMO.Q MQDEMO2 copy1.pdf
*** Start of mqgetbin.
*** Author: Ruud van Zundert, November 2ØØ2
*** Function: Get 1st message as a binary message and store as a file.
*** parms: (1)queue name, (2)queue manager, (3)Output file
*** optional parm(4) data conversion y/n (default y)
*** optional parm(5) mode browse (default descructive read)
About to open file copy1.pdf
Wrote 31Ø55 bytes to file
*** end of mqgetbin
```

To prove it worked, open up *copy1.pdf*.

WILL THIS METHOD WORK FOR A JPG FILE?

Decide for yourself whether to create a new remote and local queue for this test. A copy of *ih03pdf.txt* was made and called *ih03jpg.txt*. Basically it contained the same data except for the last section:

For the filelist:

- *c:\file1.jpg* – the name of the file to be transported.

Again, run ih03put:

```
    IhØ3put  ihØ3jpg.txt
```

Again, run mqgetbin:

```
C:\>mqgetbin DEMO.Q MQDEMO2 copy1.jpg
*** Start of mqgetbin.
*** Author: Ruud van Zundert, November 2ØØ2
*** Function: Get 1st message as a binary message and store as a file.
*** parms: (1)queue name, (2)queue manager, (3)Output file
*** optional parm(4) data conversion y/n (default y)
*** optional parm(5) mode browse (default descructive read)
About to open file copy1.jpg
Wrote 34Ø4Ø bytes to file
*** end of mqgetbin
```

Open file *copy1.jpg* to prove it worked.

Before using the ‘mqgetbin’ program please review two of the parameters:

- The outFile, which is the length of the output file and is currently set to 40.
- The buffer, which is the maximum length of the I/O buffer and is currently set to 50,000.

My conclusion? Yes, MQSeries can transport PDF and JPG files without any problems.

MQGETBIN

```
 /* Program name: mqgetbin (originally based on amqsget)           */
 /* Author      : Ruud van Zundert, November 2ØØ2.                 */
 /* Environment : Compiled under MS Visual C++ 6.Ø                 */
 /*               MQSeries V5.2 CSD5                               */
 /*               Windows NT4 SP6A / Windows 2ØØØ SP3              */
 /* Warrenty    : None, supplied ASIS                              */
 /* Get an MQSeries message off a queue and store it as a file.    */
 /* The intention is to show that binary files like Adobe PDF or   */
```

```
/* picture files like jpeg can be transported - unaltered - and     */
/* displayed.                                                       */
/* The reason this program was written was existing utilities could */
/* not cope with some of the 'special' characters found in binary   */
/* files - e.g. CRLF (x'ØAØD'), null (x'ØØ' or x'1AØD'.             */
/*   3 required parameters -                                        */
/*                 - the name of the message queue (required)      */
/*                 - the queue manager name (optional)             */
/*                 - output file                                    */
/*   2 optional parameters -                                        */
/*                 - data conversion y/n (default n)                */
/*                 - 'browse' (optional i.e. non-destructive)       */
/* Variables that may need reviewing :                              */
/*   outFile   length of the output file                            */
/*   buffer    maximum length of i/o buffer                         */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>            /* Needed for _O_RDWR def             */
#include <io.h>               /* Needed for _read function          */
    /* includes for MQI  */
#include <cmqc.h>
char      outFile1[41];       /* length of output file              */
int      fHandle1;            /* file handle                        */
unsigned   byteswritten;
void open_sesame (void);      /* outine to open file                */
int main(int argc, char **argv)
{
  /*   Declare MQI structures needed                                */
  MQOD     od = {MQOD_DEFAULT};    /* Object Descriptor             */
  MQMD     md = {MQMD_DEFAULT};    /* Message Descriptor            */
  MQGMO   gmo = {MQGMO_DEFAULT};   /* get message options           */
   MQHCONN  Hcon;                   /* connection handle
*/
  MQHOBJ   Hobj;                   /* object handle                 */
  MQLONG   O_options;              /* MQOPEN options                */
  MQLONG   C_options;              /* MQCLOSE options               */
  MQLONG   CompCode;               /* completion code               */
  MQLONG   OpenCode;               /* MQOPEN completion code        */
  MQLONG   Reason;                 /* reason code                   */
  MQLONG   CReason;                /* reason code for MQCONN        */
  MQBYTE   buffer[5ØØØ1];          /* message buffer                */
  MQLONG   buflen;                 /* buffer length                 */
  MQLONG   messlen;                /* message length received       */
  char     QMName[5Ø];             /* queue manager name            */
  char      mode[6] = "nnnnn";      /* mode BROWSE
*/
  char      Conv[2] = "n";         /* data conversion y/n          */
  fprintf(stderr,"*** Start of mqgetbin.\n");
  fprintf(stderr,"*** Author: Ruud van Zundert, November 2ØØ2\n");
```

```
   fprintf(stderr,"*** Function: Get 1st message as a binary message
and
       store as a file.\n\n");
   fprintf(stderr,"*** parms: (1)queue name, (2)queue manager,
(3)Output
       file\n");
   fprintf(stderr,"*** optional parm(4) data conversion y/n (default
       y)\n");
   fprintf(stderr,"*** optional parm(5) mode browse (default
descructive
       read)\n");
   if (argc < 4)
   {
     fprintf(stderr,"Required parameters missing.\n");
     exit(99);
   }
   /*   Store various parameters.                                   */
   /*                                                               */
   strcpy(od.ObjectName, argv[1]);   /* pick up queue name          */
   QMName[Ø] = Ø;   /* default */
   if (argc > 2)
     strcpy(QMName, argv[2]);        /* pick up qmgr name           */
   if (argc > 3)
     strcpy(outFile1, argv[3]);      /* pick up output file         */
   if (argc > 4)
     strncpy(Conv, argv[4], 1);      /* pick up conversion flag     */
   if (argc > 5)
     strncpy(mode, argv[5], 5);      /* pick up mode                */
   open_sesame();                    /* open output file            */
   /*   Connect to queue manager                                    */
   MQCONN(QMName,                    /* queue manager               */
      &Hcon,                         /* connection handle           */
      &CompCode,                     /* completion code             */
        &CReason);                   /* reason code                 */
   /* report reason and stop if it failed     */
   if (CompCode == MQCC_FAILED)
   {
     fprintf(stderr,"MQCONN ended with reason code %ld\n", CReason);
     exit( (int)CReason );
   }
   /*   Open the named message queue for input; exclusive or shared  */
   /*   use of the queue is controlled by the queue definition here  */
   O_options = MQOO_INPUT_AS_Q_DEF    /* open queue for input        */
         + MQOO_FAIL_IF_QUIESCING;    /* but not if MQM stopping     */
   if (memcmp(mode, "browse" , 5) == Ø)
      O_options = O_options + MQOO_BROWSE;
   MQOPEN(Hcon,                       /* connection handle           */
         &od,                         /* object descriptor for queue */
           O_options,                 /* open options                */
           &Hobj,                     /* object handle               */
        &OpenCode,                 /* completion code            */
        &Reason);                  /* reason code                */
/* report reason, if any; stop if failed      */
if (Reason != MQRC_NONE)
   fprintf(stderr,"MQOPEN ended with reason code %ld\n", Reason);
if (OpenCode == MQCC_FAILED)
   fprintf(stderr,"unable to open queue for input\n");
/*   Get first message from the message queue and store as a file */
CompCode = OpenCode;       /* use MQOPEN result for initial test  */
gmo.Options = MQGMO_WAIT;       /* wait for new messages          */
// + MQGMO_ACCEPT_TRUNCATED_MSG;   decide if you want this
if ( strncmp(Conv, "y", 1) == Ø )
   gmo.Options = gmo.Options + MQGMO_CONVERT;
if (memcmp(mode, "browse" , 5) == Ø)
   gmo.Options = gmo.Options + MQGMO_BROWSE_NEXT;
gmo.WaitInterval = Ø;      /* Ø second limit for waiting          */
buflen = sizeof(buffer) - 1; /* buffer size available for GET     */
 memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
 memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
 md.Encoding       = MQENC_NATIVE;
 md.CodedCharSetId = MQCCSI_Q_MGR;
MQGET(Hcon,                 /* connection handle                 */
       Hobj,                /* object handle                     */
       &md,                 /* message descriptor                */
       &gmo,                /* get message options               */
       buflen,              /* buffer length                     */
       buffer,              /* message buffer                    */
       &messlen,            /* message length                    */
       &CompCode,           /* completion code                   */
       &Reason);            /* reason code                       */
/* report reason, if any     */
 if (Reason != MQRC_NONE)
  {
    if (Reason == MQRC_NO_MSG_AVAILABLE)
    {                        /* special report for normal end     */
      fprintf(stderr,"no messages on this queue.\n");
    }
    else                     /* general report for other reasons  */
    {
      fprintf(stderr,"MQGET ended with reason code %ld\n", Reason);
      /*   treat truncated message as a failure for this sample   */
      if (Reason == MQRC_TRUNCATED_MSG_FAILED)
      {
        CompCode = MQCC_FAILED;
      }
    }
  }
if (CompCode != MQCC_FAILED)
  {
    buffer[messlen] = '\Ø';            /* add terminator          */
```

```
      /* Write buffer as a 'blob' of data using _write */
      if(( byteswritten = _write( fHandle1, buffer, messlen)) == -1 )
         fprintf( stderr, "Write failed to file %s\n", outFile1 );
       else
         fprintf( stderr, "Wrote %u bytes to file\n", byteswritten );
    }
   /*   Close the source queue (if it was opened)                   */
   if (OpenCode != MQCC_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)
      fprintf(stderr,"MQCLOSE ended with reason code %ld\n", Reason);
   }
   /*   Disconnect from MQM if not already connected                */
   if (CReason != MQRC_ALREADY_CONNECTED )
   {
     MQDISC(&Hcon,                    /* connection handle          */
            &CompCode,                /* completion code            */
            &Reason);                 /* reason code                */
     /* report reason, if any     */
     if (Reason != MQRC_NONE)
      fprintf(stderr,"MQDISC ended with reason code %ld\n", Reason);
   }
   /* End of mqgetbin. Close file(s).                              */
   _close (fHandle1);
   fprintf(stderr,"*** end of mqgetbin\n");
   return(0);
 }
void open_sesame (void)
{
   /* Open data file for output: */
   fprintf(stderr, "About to open file %s\n", outFile1 );
   if( (fHandle1 = _open( outFile1, _O_CREAT | _O_WRONLY | _O_BINARY))
== -1 )
   {
      fprintf( stderr, "open failed on output file=%s\n", outFile1 );
      exit( 1 );
   }
}
```

Ruud van Zundert, Independent Consultant (UK)
ruudvz@btclick.com

Contributing to *MQ Update*

In addition to *MQ Update*, the Xephon family of *Update* publications now includes *CICS Update, MVS Update*, *DB2 Update*, *AIX Update*, *and RACF Update*. Although the articles published are of a very high standard, the vast majority are not written by professional writers, and we rely heavily on our readers themselves taking the time and trouble to share their experiences with others. Many have discovered that writing an article is not the daunting task that it might appear to be at first glance.

They have found that the effort needed to pass on valuable information to others is more than offset by our generous terms and conditions and the recognition they gain from their fellow professionals. Often, just a few hundred words are sufficient to describe a problem and the steps taken to solve it.

If you have ever experienced any difficulties with MQ, or made an interesting discovery, you could receive a cash payment, a free subscription to any of our *Updates*, or a credit against any of Xephon's wide range of products and services, simply by telling us all about it. For a copy of our *Notes for Contributors*, which explains the terms and conditions under which we publish articles, please point your browser at www.xephon.com/nfc.

Procedures in WMQI

INTRODUCTION

ESQL, which is derived from SQL V3, is a language with functions and statements unique to WMQI.

A stored procedure is a set of SQL statements with an assigned name that is stored in the database in compiled form so that it can be shared by a number of programs. A stored procedure is processed as a unit and it can be run with one call from an application program.

Stored procedures can be used for any purpose for which you would use SQL statements, with these advantages:

- They allow more flexibility, offering capabilities such as conditional logic.
- A series of SQL statements can be executed in single stored procedure.
- Another stored procedure can be referred to within your stored procedure, which can simplify a series of complex statements/logic. Since stored procedures are stored within the DBMS, bandwidth and execution time are reduced.
- A stored procedure is compiled on the server when it is created so it executes faster than individual SQL statements. SQL Server pre-compiles stored procedures such that they execute optimally.
- Client developers are abstracted from complex design. They would simply know the stored procedure's name, its function, and the parameter list.

There are two different ways of calling a stored procedure in a WMQI message flow using ESQL, PASSTHRU and CALL.

PASSTHRU

You can call a stored procedure in WMQI by using the PASSTHRU ESQL function. This support is limited to INPUT parameters only; OUT and INOUT parameters are not supported. SqlMoreResults cannot be used by WMQI to retrieve result sets.

A stored procedure can exist either:

- Individually – supported by both DB2 and Oracle.
  - here is an example of calling an individual stored procedure using a PASSTHRU ESQL statement:

```
PASSTHRU('call myProc(?,?)',InputBody.Test.Param1,
InputBody.Test.Param2);
```

- As a part of a collective that is accessed via a package mechanism (supported by Oracle).
  - here is an example of calling such a stored procedure:

```
PASSTHRU('call myPkg.myProc(?,?)',InputBody.Test.Param1,
InputBody.Test.Param2);
```

A stored procedure can be either ‘noncommittal’ or ‘committal’.

A stored procedure that does not take explicit commit or rollback actions is known as a noncommittal stored procedure. Both DB2 and Oracle support this type of stored procedure.

Stored procedure database operations are either committed or rolled back, depending on the message flow’s commit/roll back. This behaviour is in line with that of database/warehouse nodes, which have a transaction property of ‘automatic’.

To achieve these results you should run your message flows in globally coordinated transaction mode, using XA technology. You will find more information on transactional support in the *WMQI: Introduction and Planning* manual (GC34-5599-04).

A stored procedure that contains explicit commit and rollback actions is knows as a committal stored procedure, and this feature is supported by Oracle Database.

In case of message flow rollback, stored procedure database actions are committed. This behavior is in line with the behaviour of database/warehouse nodes that have a transaction property of ‘commit’.

CALL

The latest versions of WMQI (WMQI V2.1 CSD02 and higher) include support for calling a stored procedure with IN, OUT, and INOUT parameters. CALL and CREATE PROCEDURE statements support this.

PROCEDURE

A procedure is a section of a program that performs a specific task. It is also known as a subroutine that does not return any value. These procedures are local in scope to the current node only. If you want to use the same procedure from more than one node you must define it in each node.

In WMQI you can define a procedure using a CREATE PROCEDURE statement, the syntax of which is:

```
CREATE PROCEDURE ProcedureName (ParameterList) RoutineBody
```

Where:

- *ProcedureName* is the name of the procedure.
- *ParameterList* is the list of parameters to the procedure.

ESQL procedure supports IN, OUT, and INOUT types of parameter, where:

- IN – parameters are input-only parameters.
- OUT – parameters are output-only parameters.
- INOUT – parameters are both input and output parameters.

When using OUT and INOUT parameters a procedure can return a value to the caller, although it doesn’t have a RETURN value/ statement as such. A CREATE PROCEDURE statement must be at the end of all the ESQL code within a node, otherwise you will get a syntax error at deployment.

A procedure always receives an OUT parameter with a NULL value of the correct data type. This happens irrespective of its value before the CALL statement. IN and INOUT parameters can be NULL when received by the procedure.

WMQI has the following restrictions for stored procedures:

- A parameter cannot be of ESQL reference type.
- A parameter cannot map onto the database LOB types (for example, BLOB and CLOB).
- Result sets cannot be returned or used as input or output parameters.

Types of procedure

Internal and external procedures can be implemented in WMQI.

Internal procedure

This type of procedure acts like a subroutine. Its body is composed of single or multiple (compound) statements.

The following compute node ESQL code shows an example of an internal procedure within WMQI.

```
   DECLARE in1 INTEGER;
   DECLARE in2 INTEGER;
   -- Assign values to input parameters
   SET in1 = InputRoot.XML.Data.in1;
   SET in2 = InputRoot.XML.Data.in2;
   -- Call Procedure
   CALL SwapInt(in1, in2);
   -- Update output message with swapped values
   SET OutputRoot.XML.Data.in1 = in1;
   SET OutputRoot.XML.Data.in2 = in2;
   -- Define internal procedure
   CREATE PROCEDURE swapInt (INOUT param1 INTEGER, INOUT param2
INTEGER)
   BEGIN
     DECLARE param3 INTEGER;
     SET param3 = param1;
     SET param1 = param2;
     SET param2 = param3;
   END;
```

In the above code a procedure called *swapInt* is created using a CREATE…PROCEDURE statement. This procedure accepts two integer parameters, both of type INOUT. When called by the broker this procedure will swap these parameters.

If you send the following input message to this compute node:

```
<Data>
    <in1>1Ø</in1>
    <in2>2Ø</in2>
</Data>
```

It will generate the following output message:

```
<Data>
    <in1>2Ø</in1>
    <in2>1Ø</in2>
</Data>
```

As you can see here, the numbers are swapped after calling the internal procedure.

External procedure

You can define an external store procedure in the broker using a CREATE ..PROCEDURE EXTERNAL ESQL statement. This procedure calls the named routine as a stored procedure in the database specified by the containing node's data source.

The syntax of a CREATE PROCEDURE statement for an external procedure is as follows:

```
CREATE PROCEDURE ProcedureName (ParameterList) EXTERNAL NAME
dbStoredProcName;
```

where *DbStoredProcName* is the name of the stored procedure in the database.

The name specified in the CREATE PROCEDURE statement need not match the actual name of the stored procedure in the database but the name of the stored procedure must match with the name specified in the EXTERNAL NAME clause of the CREATE PROCEDURE ESQL statement.

You can specify either a qualified or unqualified procedure name in the EXTERNAL NAME clause of the CREATE PROCEDURE statement.

Before calling a stored procedure using a CALL ESQL statement it must be defined in the database as well as in the broker.

The following compute node ESQL code shows an example of an external procedure within WMQI.

```
DECLARE in1 INTEGER;
DECLARE in2 INTEGER;
SET in1 = InputRoot.XML.Data.in1;
SET in2 = InputRoot.XML.Data.in2;
CALL SwapInt(in1, in2);
SET OutputRoot.XML.Data.in1 = in1;
SET OutputRoot.XML.Data.in2 = in2;
CREATE PROCEDURE swapInt(INOUT param1 INTEGER, INOUT param2 INTEGER)
  EXTERNAL NAME dbSwapInt;
```

In the above code we defined an external procedure – *swapInt* – in the broker, which maps to the *dbSwapInt* stored procedure in the DB2 database. This procedure will swap two input parameters when executed.

Please note that the following commands are executed on the Windows platform. The same or equivalent commands can be executed on other platforms to create stored procedures. Please make sure that support for SQL stored procedures is installed on your database server in order to create the stored procedure.

To define this stored procedure in the database, first copy and paste the following code into a text file and name the file *swap.sql*. Save the file somewhere on your local drive, for example *C:\sproc*.

```
--DB2 Example Stored Procedure
DROP PROCEDURE dbSwapInt @
CREATE PROCEDURE dbSwapInt
(INOUT in1_param INT,
 INOUT in2_param INT)
LANGUAGE SQL
BEGIN
DECLARE temp_param INT;
SET temp_param = in1_param;
SET in1_param = in2_param;
SET in2_param = temp_param;
END @
```

Start the DB2 command window and enter the commands listed below.

- Creating a stored procedure in a DB2 database:

```
C:\sproc>db2 connect to <dbname> user <userid> using <pwd>
   Database Connection Information
 Database server        = DB2/NT 7.2.2
 SQL authorization ID   = <userid>
 Local database alias   = <dbname>
C:\sproc>db2 -td@ -f swap.sql
DB2ØØØØI  The SQL command completed successfully.
DB2ØØØØI  The SQL command completed successfully.
C:\sproc>db2 disconnect current
```

With WMQI 2.1 CSD3 there is a slight change in the mechanism of calling a stored procedure using the CREATE PROCEDURE command.

For DB2 and Oracle, if you don't specify a schema name when defining external stored procedures in WMQI the broker will use a database connection user name as a default schema name. You must specify a schema name explicitly in the EXTERNAL NAME clause if the stored procedure lies in a different schema.

In Oracle, if the procedure belongs to an Oracle Package you must specify the schema name and package name in the EXTERNAL NAME clause, for example, *mySchema.MyPkg.mySProc*.

THE PROBLEM AND THE SOLUTION

WMQI ESQL doesn't provide all the standard ESQL functions, such as ASCII, HEX, EXP, LOG, DAYNAME, DAYOFWEEK, etc. Since WMQI does not support the calling of stored functions using ESQL we cannot call these database functions directly from our message flows.

It would be a time-consuming job to write our own ESQL code to implement such standard functionality. In addition, a lot of testing would be required on that piece of code before we could use it in our message flows.

To get around this problem you could write a stored procedure that would call these database functions, with appropriate parameters.

Listed below are a few examples of calling such standard functions using DB2 stored procedures.

ASCII function

Compute Node ESQL code

```
DECLARE a CHAR;
SET a = 'K';
DECLARE in1 INTEGER;
CALL ASCIIDB(a, in1);
SET OutputRoot.XML.Data.CharValue = a;
SET OutputRoot.XML.Data.ASCII = in1;
CREATE PROCEDURE ASCIIDB(IN param1 CHAR, OUT param2 INT) EXTERNAL
  NAME DBASCII;
```

Defining a stored procedure in DB2 (dbASCII.sql)

```
--DB2 Example Stored Procedure
DROP PROCEDURE DBASCII @
CREATE PROCEDURE DBASCII (IN in_param CHAR, OUT out_param INT)
LANGUAGE SQL
BEGIN
  SET out_param = ASCII(in_param);
END @
```

Creating a stored procedure in a DB2 database

```
C:\sproc>db2 -td@ -f dbASCII.sql
```

DAYNAME function

Compute Node ESQL code

```
   DECLARE today DATE;
   DECLARE out1 CHAR;
   SET today = CURRENT_DATE;
   CALL DBDAYNAME(today, out1);
   SET OutputRoot.XML.Data.Today = today;
   SET OutputRoot.XML.Data.DayName = out1;
   CREATE PROCEDURE DBDAYNAME( IN param1 DATE, OUT param2 CHAR)
EXTERNAL
     NAME DBDAYNAME;
```

Defining a stored procedure in DB2 (dbDayName.sql)

```
--DB2 Example Stored Procedure
DROP PROCEDURE dbDAYNAME @
CREATE PROCEDURE dbDAYNAME (IN in_param DATE, OUT out_param CHAR)
LANGUAGE SQL
BEGIN
```

```
  SET out_param = DAYNAME(in_param);
END @
```

Creating a stored procedure in a DB2 database

```
C:\sproc>db2 -td@ -f dbDayName.sql
```

MONTHNAME function

Compute Node ESQL code

```
DECLARE today DATE;
DECLARE out1 CHAR;
SET today = CURRENT_DATE;
CALL DBMONTHNAME(today, out1);
SET OutputRoot.XML.Data.Today = today;
SET OutputRoot.XML.Data.MonthName = out1;
CREATE PROCEDURE DBMONTHNAME( IN param1 DATE, OUT param2 CHAR)
  EXTERNAL NAME DBMONTHNAME;
```

Defining a stored procedure in DB2 (dbDayName.sql)

```
--DB2 Example Stored Procedure
DROP PROCEDURE dbMONTHNAME @
CREATE PROCEDURE dbMONTHNAME (IN in_param DATE, OUT out_param
   VARCHAR(2Ø))
LANGUAGE SQL
BEGIN
  SET out_param = MONTHNAME(in_param);
END @
```

Creating a stored procedure in a DB2 database

```
C:\sproc>db2 -td@ -f dbMonthName.sql
```

CONCLUSION

Stored procedures offer developers a lot of flexibility with many features that are not available using standard ESQL. WMQI allows us to use stored procedures in our message flows seamlessly. The combination of these two factors allows us to create very powerful WMQI applications rapidly.

REFERENCES

- *WMQI 2.1 ESQL Reference* manual (SC34-5923-02).

If you found this article useful or have any suggestions or comments to make, please send me an e-mail at kiran@ingale.net.

Kiran Ingale, EAI Architect
Aviana Global Technologies (USA)

JMS to IMS via WMQ

INTRODUCTION

Even before the recent slowdown it was rare for a business to replace all its key applications at the same time. A result of this is that new technologies often have to interface with existing applications, addressing them in terms and with data formats they can understand.

When adding a JMS-based application to an existing business system there can be a need to produce messages in a JMS application which are to be processed in an existing non-JMS application. The *WebSphere MQ Using Java* manual has a section entitled *Mapping JMS to a native WebSphere MQ application*, which describes how to send a message from a JMS Client application to a traditional WebSphere MQ application that has no knowledge of the WebSphere MQ Rules and Formatting Header (MQRFH2).

This header, which can be used to send data that has been encoded using an XML-like syntax, is used in the WebSphere MQ JMS implementation to carry header information from one JMS application to another. As this extract from a question indicates, knowledge of how to do this is not universal, nor, as I found out, is sample code to show this readily available:

"We currently use the MQ-IMS Bridge for all data inbound to IMS scattered across roughly seven mainframes (test and production). This is working very well – kudos to your group. We are working with an application group that adamantly insists that they must use the IMS Adapter because they can find no examples of coding JMS to use the IIH Header, and have even stated that their research indicates they cannot code JMS to use the IIH Header.

"I hope you can find some sample code for us."

A SOLUTION

This example code builds a message using JMS in a Java application, which can be sent via WebSphere MQ and the MQ-IMS Bridge into IMS. The use of WebSphere MQ data conversion and the output from IMS are also shown.

The message flow from which the examples are drawn is shown in Figure 1. The message is put by the Java application *JMStoIMSviaWMQ*, running using a WMQ for Windows V5.3 queue manager, onto a remote queue (1). This resolves to an MQ-IMS Bridge queue on the WMQ for z/OS queue manager (2). From the Bridge queue the message is sent by the MQ-IMS Bridge to IMS, where the transaction, or, as in this case, command, is processed by IMS. The response is then sent back to the reply-to queue (3) specified on the initial message.

The sample code is listed in the appendices following this article.

The message format expected by IMS is:

```
   LLZZ<tcode><data>
```

To achieve this we build the message shown in Figure 2.

In the case of this simple example the <tcode> is replaced by the IMS command **/DIS** and the data is 'POOL LUMP'. All IMS systems using the MQ-IMS Bridge have a LUMP pool, so valid output will always be received when this command is processed in IMS.

When using the MQ-IMS Bridge to invoke an IMS transaction or command an MQ-IMS Information Header (MQIIH) is normally added to control the way in which the transaction is invoked and the information the transaction or command is passed in its IMS header. The MQIIH can also be used to control the WebSphere MQ data conversion of the message and the response.

The MQ-IMS Bridge expects this message:

```
ØØØØØØØØ:  C9C9 C84Ø ØØØØ ØØØ1 ØØØØ ØØ54 ØØØØ Ø311 'IIH ............'
ØØØØØØ1Ø:  ØØØØ Ø1F4 D4D8 C9D4 E2E5 E24Ø ØØØØ ØØØØ '...4MQIMSVS ....'
ØØØØØØ2Ø:  4Ø4Ø 4Ø4Ø 4Ø4Ø 4Ø4Ø 4Ø4Ø 4Ø4Ø 4Ø4Ø 4Ø4Ø '                '
ØØØØØØ3Ø:  D4D8 C9D4 E2E5 E24Ø D4C1 E3E3 C8F3 E6E2 'MQIMSVS         '
ØØØØØØ4Ø:  ØØØØ ØØØØ ØØØØ ØØØØ ØØØØ ØØØØ ØØØØ ØØØØ '................'
ØØØØØØ5Ø:  4ØFØ C34Ø ØØ12 ØØØØ 61C4 C9E2 4ØD7 D6D6 ' ØC ..../DIS POO'
ØØØØØØ6Ø:  D34Ø D3E4 D4D7                          'L LUMP          '
```

This message format, displayed using the WebSphere MQ sample programs *CSQ4BCG1* or *amqsbcg*, is the one needed to input the command message **/DIS POOL LUMP** to IMS through the MQ-IMS Bridge.

The sample code demonstrates the techniques necessary to get messages from JMS into IMS. The MQIIH Format and

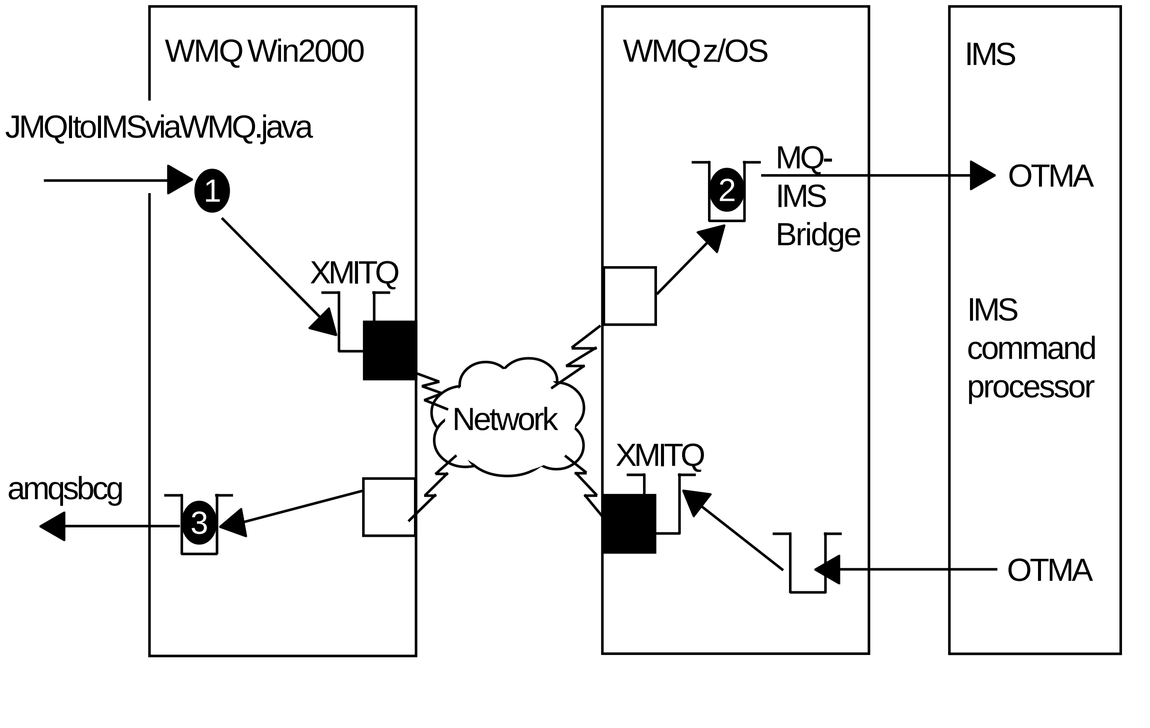


Figure 1: Message flow from which examples are drawn

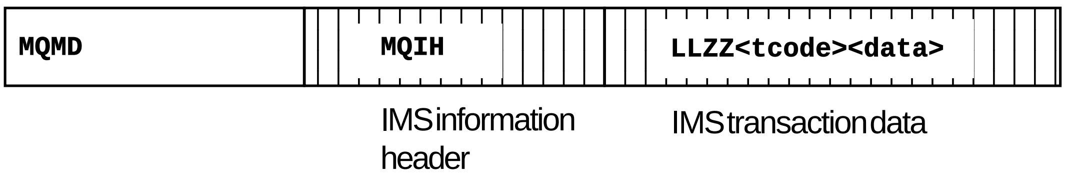


Figure 2: Message structure required to deliver the format expected by IMS

ReplyToFormat fields, together with normal MQ data conversion routines, get the data into the correct character set and encoding.

The simple sample code produces this message:

```
ØØØØØØØØ:  4949 482Ø Ø1ØØ ØØØØ 54ØØ ØØØØ 22Ø2 ØØØØ 'IIH ....T..."...'
ØØØØØØ1Ø:  B8Ø4 ØØØØ 4D51 494D 5356 532Ø ØØØØ ØØØØ '¸...MQIMSVS ....'
ØØØØØØ2Ø:  2Ø2Ø 2Ø2Ø 2Ø2Ø 2Ø2Ø 2Ø2Ø 2Ø2Ø 2Ø2Ø 2Ø2Ø '                '
ØØØØØØ3Ø:  4D51 494D 5356 532Ø 2Ø2Ø 2Ø2Ø 2Ø2Ø 2Ø2Ø 'MQIMSVS         '
ØØØØØØ4Ø:  ØØØØ ØØØØ ØØØØ ØØØØ ØØØØ ØØØØ ØØØØ ØØØØ '................'
ØØØØØØ5Ø:  2Ø3Ø 432Ø 12ØØ ØØØØ 2F44 4953 2Ø5Ø 4F4F ' ØC ..../DIS POO'
ØØØØØØ6Ø:  4C2Ø 4C55 4D5Ø                          'L LUMP          '
```

THE TECHNIQUES

Using MQ JMS extensions

MQQueueConnectionFactory and MQQueue classes are used because some of the parameters needed to build the correct format message are set with methods supported by these classes.

Connecting to MQ

The connection to the queue manager to be used is normally defined externally to the program, using the JNDI interface to enable application portability. It can also be coded into the application as follows:

```
// Set the Queue Manager and Queue names to be used
//    queue manager will be used
private final String QUEUE_MANAGER = " ";
```

And used by the code as follows:

```
// Set bindings mode and define the queue manager to use
factory.setTransportType(JMSC.MQJMS_TP_BINDINGS_MQ);
factory.setQueueManager(QUEUE_MANAGER);
```

Building the message without an MQRFH2

Neither the MQ-IMS Bridge nor IMS is capable of handling the MQRFH2 so it is necessary to build the message without it. The JMS specification allows the target client to be set to non-JMS; this suppresses the generation of the header information usually carried in the MQRFH2 – the code extract below shows how this is coded.

```
// queue.targetClient must be set to JMSC.MQJMS_CLIENT_NONJMS_MQ
// as the target client is WMQ and not a JMS application.
//   This causes the message to be produced without an MQRFH2
//   header and thus be suitable for use by a non-JMS program
queue.setTargetClient(JMSC.MQJMS_CLIENT_NONJMS_MQ);
```

Building and writing the message

The message data required by the MQ-IMS Bridge is a mixture of integers, strings, and byte arrays. To enable the output of these types of data from a JMS program a BytesMessage must be built.

The string, as required by the target programs, is an array of characters, not a string with a defining length indicator such as that written by the WriteObject method. To accomplish this the string is written as a byte array as shown in the method addChars.

In this example the values of the data required in the message are set simply with constants and then written, in order, to the message.

Using the MQMD and MQIIH Format fields to convert the data

The application sets the Encoding and CodedCharSetId in the MQMD and MQIIH to values appropriate for the platform on which the application is running.

Setting the MQMD.Format field to MQIMS allows WebSphere MQ to convert the character and integer data in the MQIIH, and setting the MQIIH.Format field allows WebSphere MQ to convert the LLZZ<data>.

This conversion is done when the receiver channel puts to a bridge queue on WebSphere MQ for z/OS, not, as may have been expected, when the MQ-IMS Bridge gets the message.

Setting the MQMD.ReplyToFormat field to MQIMSVS allows WebSphere MQ to convert the LLZZ<data> of the reply message, as this field is used as the MQIIH.Format field of the reply.

The MQ-IMS Bridge sets the Encoding and CodedCharSetId fields of the MQMD and MQIIH when generating the reply message. When the receiving application uses the MQGMO_CONVERT option the data is returned in the format and encoding requested by the application.

Handling exceptions

To aid debugging it is important to capture the linked exception. The exception will indicate that a problem occurred with the JMS transport layer; the linked exception will give the WebSphere MQ return code, which is necessary to debug transport problems.

```
catch (JMSException je) {
      System.out.println("Exception in JMStoIMSviaWMQ: " +je);
      Exception e = je.getLinkedException();
      if (e != null) {
        System.out.println("linked exception: "+e);
      }
```

For example:

```
   Exception in JMStoIMSviaWMQ: javax.jms.JMSException: MQJMS2ØØ5:
     failed to create MQQueueManager for 'JOHNJ'
   linked exception: com.ibm.mq.MQException:  MQJEØØ1: Completion Code
     2, Reason 2Ø59
```

APPENDICES

- JMStoIMSviaWMQ.java.
- MQIIH structure.
- The message produced by JMStoIMSviaWMQ.java.

  The code shows the output from the WMQ sample program *amqsbcg*.

- The message on the z/OS Bridge queue (conversion by WMQ).

  The message produced by JMStoIMSviaWMQ.java has been sent to WMQ on z/OS and data conversion was carried out when the message was put onto the bridge queue. The code shows the output from the WMQ sample program CSQ4BCG1.

- The response from IMS (on Windows, unconverted).

  The code shows the output from the WMQ sample program *amqsbcg*.

- Response from IMS (on Windows, converted by WMQ).

  The code shows the output from a modified version of the WMQ sample program *amqsbcg*. The program was changed to specify MQGMO_CONVERT on the MQGET.

JMStoIMSviaWMQ.JAVA

```
//* Module Name      : JMStoIMSviaWMQ
//* Environment      : JMS with WebSphere MQ as transport layer
//* Description      : This program shows how to construct a message
//*                      which will be used by a non-JMS consumer.
//*                      The consumer in this example is IMS accessed
//*                      through the MQ-IMS Bridge.
//*                      Alter the QUEUE_MANAGER and QUEUE attributes
//*                      before using.  As written it will use the
//*                      default WebSphere MQ Queue Manager,
//*                      a queue "Q1", and a reply queue "Q1_REPLY"
//* Limitations      : This program was tested with WebSphere MQ V5.3
//* Function         : This program takes no parameters and produces
//*                      a message which if sent via WMQ to IMS
//*                      executes the IMS /DIS POOL LUMP command.
import java.util.*;
import java.lang.Exception;
import javax.jms.*;
import javax.naming.*;
import com.ibm.mq.jms.*;
public class JMStoIMSviaWMQ {
  private BytesMessage message = null;
  // Set the Queue Manager and Queue names to be used
  //    If the lines below are unchanged, queues Q1 and Q1_REPLY
  //    on the default queue manager will be used
  private final String QUEUE_MANAGER = "";
  private final String QUEUE = "Q1";
  private final String REPLY_TO = "Q1_REPLY";
```

```
// Define constants to hold the values to be used
// in the fields of the MQMD and MQIIH in the message
private String strucID =       "IIH ";
private int version =          1;
private int strucLength =      84;
private int encoding =         546;
private int codedCharSetID =   12Ø8;
private String format =        "MQIMSVS ";
private int flags =            Ø;
private String lTermOverride = "        ";
private String mfsMapName =    "        ";
private String replyToFormat = "MQIMSVS ";
private String authenticator = "        ";
private byte[] tranInstanceID = new byte[16];
private String tranState =     " ";
private String commitMode =    "Ø";
private String securityScope = "C";
private String reserved =      " ";
// Define the content of the message body
public String content = "/DIS POOL LUMP";
public static void main(String[] args) {
  JMStoIMSviaWMQ PutMessage = new JMStoIMSviaWMQ();
  PutMessage.go();
}
//*
//* Constructor
//*
JMStoIMSviaWMQ() {
  super();
}
//*
//*  Set up the necessary JMS objects, build and send the message
//*
private void go() {
  try {
    MQQueueConnectionFactory factory = null;
    MQQueue queue = null;
    MQQueue replyToQueue = null;
    QueueConnection connection = null;
    QueueSession session = null;
    QueueSender sender = null;
    System.out.println("Creating a QueueConnectionFactory");
    factory = new MQQueueConnectionFactory();
    // Set bindings mode and define the queue manager to use
    factory.setTransportType(JMSC.MQJMS_TP_BINDINGS_MQ);
    factory.setQueueManager(QUEUE_MANAGER);
    System.out.println("Creating a QueueConnection");
    connection = factory.createQueueConnection();
    System.out.println("Starting the connection");
    connection.start();
    System.out.println("Creating QueueSession");
    boolean transacted = false;
    int acknowledgeMode = Session.AUTO_ACKNOWLEDGE;
    session = connection.createQueueSession(transacted,
    acknowledgeMode);
    System.out.println("Creating ReplyTo Queue");
    replyToQueue = new MQQueue("",REPLY_TO);
    System.out.println("Creating Queue");
    queue = new MQQueue();
    queue.setBaseQueueName(QUEUE);
    queue.setPersistence(DeliveryMode.NON_PERSISTENT);
    // queue.targetClient must be set to JMSC.MQJMS_CLIENT_NONJMS_MQ
    // as the target client is WMQ and not a JMS application.
    //   This causes the message to be produced without an MQRFH2
         header
    //   and thus be suitable for use by a non-JMS program
    queue.setTargetClient(JMSC.MQJMS_CLIENT_NONJMS_MQ);
    // Set the Encoding and CodedCharSetId in the MQMD
    queue.setEncoding(encoding);
    queue.setCCSID(codedCharSetID);
    System.out.println("Creating QueueSender");
    sender = session.createSender(queue);
    System.out.println("Creating BytesMessage");
    message = session.createBytesMessage();
    // Set the Format and ReplyToQ in the MQMD
    message.setStringProperty("JMS_IBM_Format","MQIMS   ");
    message.setJMSReplyTo(replyToQueue);
    // Write the output message
    //   Data appears in the message in the order it is written.
    //   buildMessage writes all the data into the message,
    //   it assumes that the values have been set prior to it being
    //   invoked - this is done using constants in this example
    System.out.println("Building BytesMessage");
    buildMessage();
    System.out.println("Sending BytesMessage");
    sender.send(message);
    System.out.println("Closing objects");
    sender.close();
    session.close();
    connection.close();
    System.out.println("Finished");
  }
  catch (JMSException je) {
    System.out.println("Exception in JMStoIMSviaWMQ: " +je);
    Exception e = je.getLinkedException();
    if (e != null) {
      System.out.println("linked exception: "+e);
    }
  }
}
//*
```

```
  //* Build the message to put onto the queue
  //*   - The message is built using BytesMessage; this message
  //*     type is for encoding a body to match an existing message
  //*     format. Fields are added to the message in the order in
  //*     which they appear.
  //*
  private void buildMessage() {
    buildHeader();
    buildBody();
  }
  //*
  //* Build the MQIIH with the defined value
  //*
  //*   The C declaration of the MQIIH structure is:
  //* typedef struct tagMQIIH MQIIH;
  //* struct tagMQIIH {
  //*   MQCHAR4   StrucId;         /* Structure identifier */
  //*   MQLONG    Version;         /* Structure version number */
  //*   MQLONG    StrucLength;     /* Length of MQIIH structure */
  //*   MQLONG    Encoding;        /* Reserved */
  //*   MQLONG    CodedCharSetId;  /* Reserved */
  //*   MQCHAR8   Format;          /* MQ format name of data that
                                      follows
  //*                                 MQIIH */
  //*   MQLONG    Flags;           /* Flags */
  //*   MQCHAR8   LTermOverride;   /* Logical terminal override */
  //*   MQCHAR8   MFSMapName;      /* Message format services map name
*/
  //*   MQCHAR8   ReplyToFormat;   /* MQ format name of reply message
*/
  //*   MQCHAR8   Authenticator;   /* RACF password or passticket */
  //*   MQBYTE16  TranInstanceId;  /* Transaction instance identifier
*/
  //*   MQCHAR    TranState;       /* Transaction state */
  //*   MQCHAR    CommitMode;      /* Commit mode */
  //*   MQCHAR    SecurityScope;   /* Security scope */
  //*   MQCHAR    Reserved;        /* Reserved */
  //* };
  //*
private void buildHeader() {
    try {
      addChars(strucID);
      message.writeInt(version);
      message.writeInt(strucLength);
      message.writeInt(encoding);
      message.writeInt(codedCharSetID);
      addChars(format);
      message.writeInt(flags);
      addChars(lTermOverride);
      addChars(mfsMapName);
      addChars(replyToFormat);
      addChars(authenticator);
      message.writeBytes(tranInstanceID);
      addChars(tranState);
      addChars(commitMode);
      addChars(securityScope);
      addChars(reserved);
    } catch(JMSException je) {
      System.out.println("Exception in buildHeader: " +je);
      Exception e = je.getLinkedException();
      if (e != null) {
        System.out.println("linked exception: "+e);
      }
    }
  }
  //*
  //* Build up the body of the text using the content specified
  //*  A message to be sent into IMS has a format LLZZ<data> where:
  //*  - LL is a two byte integer defining the length of the message
  //*    segment
  //*  - ZZ is two flag bytes set to NULL
  //*  - <data> is the application data structure being sent into IMS
  //*    (in this case it is in variable 'content')
  //*
  private void buildBody() {
    try {
      // Find the length of the body section + LLZZ
      short bodyLength = (short)(content.length() + 4);
      // Two bytes to represent the length of the body
      message.writeShort(bodyLength);
      // Two null bytes
      message.writeShort((short)Ø);
      // Add the message content
      addChars(content);
    } catch(JMSException je) {
      System.out.println("Exception in buildBody: " +je);
      Exception e = je.getLinkedException();
      if (e != null) {
        System.out.println("linked exception: "+e);
      }
    }
  }
  //* Add a String to the message, character by character
  //*
  private void addChars(String chars) {
    try {
        byte[] b = chars.getBytes();
        message.writeBytes(b);
    } catch(JMSException je) {
      System.out.println("Exception in addChars: " +je);
```

```
      Exception e = je.getLinkedException();
      if (e != null) {
        System.out.println("linked exception: "+e);
      }
    }
  }
}
```

MQIIH structure

```
/*  MQIIH Structure -- IMS Interface Header                          */
 typedef struct tagMQIIH {
   MQCHAR4   StrucId;         /* Structure identifier */
   MQLONG    Version;         /* Structure version number */
   MQLONG    StrucLength;     /* Length of MQIIH structure */
   MQLONG    Encoding;        /* Reserved */
   MQLONG    CodedCharSetId;  /* Reserved */
   MQCHAR8   Format;          /* MQ Format name */
   MQLONG    Flags;           /* Reserved */
   MQCHAR8   LTermOverride;   /* Logical terminal override */
   MQCHAR8   MFSMapName;      /* Message format services map name */
   MQCHAR8   ReplyToFormat;   /* MQ Format name of reply message */
   MQCHAR8   Authenticator;   /* RACF password or pass ticket */
   MQBYTE16  TranInstanceId;  /* Transaction instance id */
   MQCHAR    TranState;       /* Transaction state */
   MQCHAR    CommitMode;      /* Commit mode */
   MQCHAR    SecurityScope;   /* Security scope */
   MQCHAR    Reserved;        /* Reserved */
  } MQIIH;
```

Message produced by JMQtoIMSviaWMQ.JAVA

```
****Message descriptor****
  StrucId  : 'MD  '  Version : 2
  Report   : 0  MsgType : 1
  Expiry   : -1  Feedback : 0
  Encoding : 546  CodedCharSetId : 1208
  Format : 'MQIMS   '
  Priority : 4  Persistence : 0
  MsgId : X'414D5120514D5F6A6F686E20202020202E63913D20000201'
  CorrelId : X'000000000000000000000000000000000000000000000000'
  BackoutCount : 0
  ReplyToQ       : 'Q1_REPLY                                        '
  ReplyToQMgr    : 'JOHNJ                                           '
  ** Identity Context
  UserIdentifier : 'johnj       '
  AccountingToken :
   X'16010515000000B72EEA1114086F4A3A37067FF401000000000000000000000B'
  ApplIdentityData : '                                '
  ** Origin Context
  PutApplType    : '11'
  PutApplName    : 'C:\WINNT\SYSTEM32\java.exe  '
  PutDate  : '20020925'    PutTime  : '08030555'
  ApplOriginData : '    '
  GroupId : X'000000000000000000000000000000000000000000000000'
  MsgSeqNumber   : '1'
  Offset         : '0'
  MsgFlags       : '0'
  OriginalLength : '-1'
****   Message      ****
 length - 102 bytes
00000000:  4949 4820 0100 0000 5400 0000 2202 0000 'IIH ....T..."...'
00000010:  B804 0000 4D51 494D 5356 5320 0000 0000 '¸...MQIMSVS ....'
00000020:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000030:  4D51 494D 5356 5320 2020 2020 2020 2020 'MQIMSVS         '
00000040:  0000 0000 0000 0000 0000 0000 0000 0000 '................'
00000050:  2030 4320 1200 0000 2F44 4953 2050 4F4F ' 0C ..../DIS POO'
00000060:  4C20 4C55 4D50                          'L LUMP          '
```

Message on z/OS bridge queue (conversion by WMQ)

```
****Message descriptor****
  StrucId  : 'MD  '  Version : 1
  Report   : 0  MsgType : 1
  Expiry   : -1  Feedback : 0
  Encoding : 785  CodedCharSetId : 500
  Format : 'MQIMS   '
  Priority : 4  Persistence : 0
  MsgId : X'414D51204A4F484E4A2020202020202020203D6C903D20001201'
  CorrelId : X'000000000000000000000000000000000000000000000000'
  BackoutCount : 0
  ReplyToQ       : 'Q1_REPLY                                        '
  ReplyToQMgr    : 'JOHNJ                                           '
  ** Identity Context
  UserIdentifier : 'JOHNJ       '
  AccountingToken :
   X'16010515000000B72EEA1114086F4A3A37067FF40100000000000000000000000B'
  ApplIdentityData : '                                '
  ** Origin Context
  PutApplType    : '11'
  PutApplName    : 'C:\WINNT\SYSTEM32\java.exe  '
  PutDate  : '20020924'    PutTime  : '15271673'
  ApplOriginData : '    '
****   Message      ****
 length - 101 bytes
00000000:  C9C9 C840 0000 0001 0000 0054 0000 0311 'IIH ............'
00000010:  0000 01F4 D4D8 C9D4 E2E5 E240 0000 0000 '...4MQIMSVS ....'
00000020:  4040 4040 4040 4040 4040 4040 4040 4040 '                '
00000030:  D4D8 C9D4 E2E5 E240 D4C1 E3E3 C8F3 E6E2 'MQIMSVS         '
00000040:  0000 0000 0000 0000 0000 0000 0000 0000 '................'
```

```
ØØØØØØ5Ø:  4ØFØ C34Ø ØØ11 ØØØØ 61C4 C9E2 4ØD7 D6D6 ' ØC ..../DIS POO'
ØØØØØØ6Ø:  D34Ø D3E4 D4D7                          'L LUMP         '
```

Response from IMS (on Windows, unconverted)

```
****Message descriptor****
  StrucId  : 'MD  '  Version : 2
  Report   : Ø  MsgType : 2
  Expiry   : -1  Feedback : Ø
  Encoding : 785  CodedCharSetId : 5ØØ
  Format : 'MQIMS   '
  Priority : 4  Persistence : 1
  MsgId : X'C3E2D84ØD4D8F3F54Ø4Ø4Ø4Ø4Ø4Ø4Ø4ØB8476ECØ63D71285'
  CorrelId : X'414D512Ø4A4F484E4A2Ø2Ø2Ø2Ø2Ø2Ø2Ø3D6C9Ø3D2ØØØ1AØ1'
  BackoutCount : Ø
  ReplyToQ       : '                                                 '
  ReplyToQMgr    : 'MQ35                                             '
  ** Identity Context
  UserIdentifier : 'johnj       '
  AccountingToken :
   X'16Ø1Ø515ØØØØØØB72EEA1114Ø86F4A3A37Ø67FF4Ø1ØØØØØØØØØØØØØØØØØØØØØØB'
  ApplIdentityData : '                                '
  ** Origin Context
  PutApplType    : '2Ø'
  PutApplName    : 'PROTOGRPIYEAZI6A            '
  PutDate  : '2ØØ2Ø924'    PutTime  : '1544Ø465'
  ApplOriginData : '    '
  GroupId : X'ØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØ'
  MsgSeqNumber   : '1'
  Offset         : 'Ø'
  MsgFlags       : 'Ø'
  OriginalLength : '-1'
****   Message      ****
 length - 196 bytes
ØØØØØØØØ:  C9C9 C84Ø ØØØØ ØØØ1 ØØØØ ØØ54 ØØØØ Ø311 'ÉÉÈ@.......T....'
ØØØØØØ1Ø:  ØØØØ ØØØØ D4D8 C9D4 E2E5 E24Ø ØØØØ ØØØØ '....ÔØÉÔâåâ@....'
ØØØØØØ2Ø:  4Ø4Ø 4Ø4Ø 4Ø4Ø 4Ø4Ø C4C6 E2C4 E2D7 D6F1 '@@@@@@@@ÄÆâÄâ×Öñ'
ØØØØØØ3Ø:  4Ø4Ø 4Ø4Ø 4Ø4Ø 4Ø4Ø 4Ø4Ø 4Ø4Ø 4Ø4Ø 4Ø4Ø '@@@@@@@@@@@@@@@@'
ØØØØØØ4Ø:  ØØØØ ØØØØ ØØØØ ØØØØ ØØØØ ØØØØ ØØØØ ØØØØ '................'
ØØØØØØ5Ø:  4ØFØ 4Ø4Ø ØØ17 1BØØ 15D3 E4D4 D74Ø C2E4 '@ð@@.....ÓäÔ×@Âä'
ØØØØØØ6Ø:  C6C6 C5D9 4ØD7 D6D6 D37A 15ØØ 451B ØØ4Ø 'ÆÆÅÙ@×ÖÖÓz..E..@'
ØØØØØØ7Ø:  E2C9 E9C5 4Ø4Ø 4Ø4Ø 4ØF1 F3F4 D24Ø 4ØC8 'âÉéÅ@@@@@ñóôÒ@@È'
ØØØØØØ8Ø:  C9C7 C84Ø 4Ø4Ø 4Ø4Ø F2FØ F8D2 4Ø4Ø D3C9 'ÉÇÈ@@@@@òðøÒ@@ÓÉ'
ØØØØØØ9Ø:  D4C9 E34Ø 4Ø4Ø 4Ø4Ø D5D6 D5C5 4Ø4Ø D6E5 'ÔÉã@@@@@ÕÖÕÅ@@Öå'
ØØØØØØAØ:  C5D9 C6D3 D6E6 4Ø4Ø 4Ø4Ø 4Ø4Ø 4ØFØ D215 'ÅÙÆÓÖæ@@@@@@@ðÒ.'
ØØØØØØBØ:  ØØ14 1BØØ 155C FØF2 F2F6 F761 F1F6 F4F3 '.....\ðòòö÷añöôó'
ØØØØØØCØ:  F4F4 5C15                               'ôô\.            '
```

Response from IMS (on Windows, converted by WMQ)

```
****Message descriptor****
  StrucId  : 'MD  '  Version : 2
  Report   : Ø  MsgType : 2
  Expiry   : -1  Feedback : Ø
  Encoding : 546  CodedCharSetId : 437
  Format : 'MQIMS   '
  Priority : 4  Persistence : 1
  MsgId : X'C3E2D84ØD4D8F3F54Ø4Ø4Ø4Ø4Ø4Ø4Ø4ØB8476ECØ63D71285'
  CorrelId : X'414D512Ø4A4F484E4A2Ø2Ø2Ø2Ø2Ø2Ø2Ø3D6C9Ø3D2ØØØ1AØ1'
  BackoutCount : Ø
  ReplyToQ       : '                                                '
  ReplyToQMgr    : 'MQ35                                            '
  ** Identity Context
  UserIdentifier : 'johnj       '
  AccountingToken :
   X'16Ø1Ø515ØØØØØØB72EEA1114Ø86F4A3A37Ø67FF4Ø1ØØØØØØØØØØØØØØØØØØØØØØB'
  ApplIdentityData : '                                '
  ** Origin Context
  PutApplType    : '2Ø'
  PutApplName    : 'PROTOGRPIYEAZI6A            '
  PutDate  : '2ØØ2Ø924'    PutTime  : '1544Ø465'
  ApplOriginData : '    '
  GroupId : X'ØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØ'
  MsgSeqNumber   : '1'
  Offset         : 'Ø'
  MsgFlags       : 'Ø'
  OriginalLength : '-1'
****   Message      ****
 length - 196 bytes
ØØØØØØØØ:  4949 482Ø Ø1ØØ ØØØØ 54ØØ ØØØØ 11Ø3 ØØØØ 'IIH ....T.......'
ØØØØØØ1Ø:  ØØØØ ØØØØ 4D51 494D 5356 532Ø ØØØØ ØØØØ '....MQIMSVS ....'
ØØØØØØ2Ø:  2Ø2Ø 2Ø2Ø 2Ø2Ø 2Ø2Ø 4446 5344 535Ø 4F31 '        DFSDSPO1'
ØØØØØØ3Ø:  2Ø2Ø 2Ø2Ø 2Ø2Ø 2Ø2Ø 2Ø2Ø 2Ø2Ø 2Ø2Ø 2Ø2Ø '                '
ØØØØØØ4Ø:  ØØØØ ØØØØ ØØØØ ØØØØ ØØØØ ØØØØ ØØØØ ØØØØ '................'
ØØØØØØ5Ø:  2Ø3Ø 2Ø2Ø 17ØØ 1BØØ ØA4C 554D 5Ø2Ø 4255 ' Ø  .....LUMP BU'
ØØØØØØ6Ø:  4646 4552 2Ø5Ø 4F4F 4C3A ØA45 ØØ1B ØØ2Ø 'FFER POOL:.E... '
ØØØØØØ7Ø:  5349 5A45 2Ø2Ø 2Ø2Ø 2Ø31 3334 4B2Ø 2Ø48 'SIZE     134K  H'
ØØØØØØ8Ø:  4947 482Ø 2Ø2Ø 2Ø2Ø 323Ø 384B 2Ø2Ø 4C49 'IGH     2Ø8K  LI'
ØØØØØØ9Ø:  4D49 542Ø 2Ø2Ø 2Ø2Ø 4E4F 4E45 2Ø2Ø 4F56 'MIT     NONE  OV'
ØØØØØØAØ:  4552 464C 4F57 2Ø2Ø 2Ø2Ø 2Ø2Ø 2Ø3Ø 4BØA 'ERFLOW       ØK.'
ØØØØØØBØ:  14ØØ 1BØØ ØA2A 3Ø32 3236 372F 3136 3433 '.....*Ø2267/1643'
ØØØØØØCØ:  3434 2AØA                               '44*.            '
```

John B Jones, BSc, MSc, MIEE, CEng
IBM Hursley (UK) © IBM 2003

MQ news

Systems integrator Promenix has recently announced that it has enabled the INTEG process group's JNIOR process control 'connectivity' unit to support the latest group of WebSphere MQ pervasive device protocols.

The JNIOR is a programmable Java device with a selection of available I/O resources onboard. The new IBM protocols allow the JNIOR devices to be utilized in a variety of industrial control and data acquisition applications.

Promenix claims that support for the new protocols, which combine a small bandwidth requirement with guaranteed message delivery, will overcome the problems with low bandwidth and unreliable connections that are commonly associated with industrial applications.

For more information contact:
Promenix, 130 Commons Court, Chadds Ford, PA 19317, USA.
Tel: +1 610 361 1560.
Fax: +1 610 361 7549.
Web: http://www.promenix.com

* * *

IBM's Transaction and Messaging Conference, featuring the CICS and MQSeries product families, takes place at the Las Vegas Hilton, Las Vegas, Nevada, USA, February 10-14, 2003.

Billed as the 'ultimate update', IBM claims the conference will equip you with the transaction and messaging technical skills you need to excel and extend your e-business applications.

For more information contact your local IBM representative.

* * *

IBM and Camstar, a provider of enterprise Manufacturing Execution Systems (MES), have recently announced a strategic alliance to deliver collaborative manufacturing solutions to the electronics industry.

The joint IBM and Camstar solution is claimed to improve manufacturing operations and reduce operational costs by integrating production information with supply chain, procurement, and customer support enterprise solutions.

Under the terms of the agreement the two companies will jointly market the Camstar InSite manufacturing execution software optimized for the IBM WebSphere infrastructure platform, including the WebSphere Application Server, WebSphere MQ, DB2, and eServer pSeries and xSeries hardware.

For more information contact:
Camstar, 900 E Hamilton Ave, Suite 400, Campbell, CA 95008, USA.
Tel: +1 408 559 5700.
Fax: +1 408 558 9350.
Web: http://www.camstar.com

* * *