4

MQ update

October 1999

In this issue

3 A trigger monitor for CICS on Open Systems

16 Client/server messages with CICS/ESA

20 Monitoring MQ events from the mainframe

25 An MQSeries batch trigger monitor for MVS/ESA

41 Client/server MQSeries with REXX

48 MQ news

MQ Update

Published by

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

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.

A trigger monitor for CICS on Open Systems

INTRODUCTION

IBM supplies an MQSeries trigger monitor for use with CICS on Open Systems. While this trigger monitor works, it has several deficiencies. For a start, it works only with the default queue manager. So, while the trigger monitor program that ships with MQSeries is useful to get someone started on using MQSeries and CICS on Open Systems together, as one adds new environments, such as unit test, QA, and production, it becomes necessary to be able to specify the queue manager to the trigger monitor program. Another reason for wanting to customize the trigger monitor application is that the IBM trigger monitor assumes that the name of the initiation queue is *SYSTEM.CICS.INITIATION.QUEUE*. It's possible that more than one CICS region are interacting with just one queue manager, and so we also need to be able to specify the initiation queue to the trigger monitor program.

This article presents a custom trigger monitor program. Before we deal with the program code, we'll describe a set-up that makes it necessary. We follow this up with a discussion on trigger monitor design, specifically looking at the trigger monitor interface between CICS on Open Systems and MQSeries. The custom trigger monitor code is included and explained in depth. Then a sample build command is shown for compiling the trigger monitor on a HP-UX 10.20 system.

PRE-REQUISITES FOR TRIGGERING CICS ON OPEN SYSTEMS.

In this section, I describe a set-up in which the trigger monitor is used. This comprises CICS for Open Systems as the external transaction monitor and a database, like Oracle or DB2, and MQSeries as resources. For simplicity, we assume that MQSeries is connecting a mainframe (OS/390) and a Unix system (HP-UX). Users update data on the mainframe, and the changes are then put on MQSeries/MVS, which transports the messages to a queue on MQSeries for HP-UX. When the message arrives on the destination queue, the queue manager puts a trigger message on the initiation queue. A custom trigger monitor running in CICS for HP-UX then reads the message and runs a transaction to update the database.

This is the broad picture, and we'll focus on the trigger monitor and its immediate environment, which is the queue manager on HP-UX, the local queue, the initiation queue, the trigger monitor, and triggered transaction. The details are listed below.

- Queue manager: *QMGR*
- CICS region: *APPLID*
- Initiation queue: *CICS.APPLID.INITIATION.QUEUE*
- Local queue: *QL.CICS.READ.QUEUE*
- CICS trigger monitor transaction: *TRMN*
- Started transaction: *APP0*
- Starting program: **applprg1**
- Trigger monitor: **trigmon2.**

(Note that the IBM-supplied trigger monitor for CICS on Open Systems is called **amqltmc0**.)

DEFINITION OF QL.CICS.READ.QUEUE

```
DEFINE QLOCAL (QL.CICS.READ.QUEUE) +
DESCR ('Queue for messages to CICS transaction APP0')
LIKE (SYSTEM.DEFAULT.LOCAL.QUEUE) +
TRIGGER (YES) +
TRIGTYPE (FIRST) +
INITQ (CICS.APPLID.INITIATION.QUEUE) +
PROCESS (PROCESS1) +
REPLACE
```

Note that *TRIGTYPE FIRST* is used for the sake of providing an illustration. Trigger monitor design and implementation doesn't change with *TRIGTYPE* parameter.

DEFINITION OF CICS.APPLID.INITIATION.QUEUE

```
DEFINE QLOCAL (CICS.APPLID.INITIATION.QUEUE) +
```

```
DESCR ('Initiation queue for CICS region APPLID') +
LIKE (SYSTEM.DEFAULT.LOCAL.QUEUE)
```

DEFINITION OF PROCESS.CICS

```
DEFINE PROCESS (PROCESS.CICS) +
REPLACE +
DESCR ('For CICS trigger monitor') +
APPLTYPE ('CICS') +
APPLICID ('APP0') +
USERDATA ('')
```

The transaction *APP0* and the starting program are aware of actions that need to be taken when the transaction *APP0* is triggered by the trigger monitor application.

PROGRAM DESIGN

The trigger program **trigmon2** is a CICS program that is started by transaction *TRMN*. *TRMN* itself could be a long-running transaction that was started explicitly or a transaction included in the CICS start-up procedure. The trigger monitor program can be implemented in C or COBOL (we'll look at a C implementation).

As you can see in the set-up, the trigger monitor is expected to poll a non-default initiation queue. This is a significant improvement over the trigger monitor that's supplied with MQSeries V5.0 (**amqltmc0**). The name of the initiation queue can be manipulated in a more general fashion by means of the environment (that is, the CICS environment file). If you want the trigger monitor to be able to monitor several initiation queues, then one way to do this is to make the initiation queue names a transaction parameter. Another solution is to make the initiation queue names a delimited text string and parse it in the trigger monitor. However, in the current program, I assume that there is one initiation queue per region. We will pass the initiation queue name to the trigger monitor program via the CICS environment file.

The second important feature of the trigger monitor is its ability to work with a non-default queue manager. This means that, when you use the *MQCONN* call, you must be able to specify the name of the queue manager and you cannot 'hard-code' the queue manager name in the trigger monitor, as this puts an obvious constraint on the program. However, you can pass the queue manager name to the trigger monitor program via the CICS environment file, which is an acceptable solution. While this solution provides more flexibility, it can be cumbersome to manage the definitions in a dynamic and evolving environment, such as you'd encounter during prototyping or initial development. This is because, every time you decide that the *APPLID* region should work with a different queue manager, you have to change the queue manager name in several locations. A better solution, therefore, is to scan the stanza files and parse the *OpenString* parameter to establish the queue manager name. Given that there can be many XADefinitions in a CICS region, this approach also has its drawbacks. For the purpose of this article, we will take the approach of specifying the queue manager name via the CICS environment file. This solution offers us the flexibility we need and has a manageable overhead.

TRIGGER MONITOR PROGRAM

As stated earlier, the trigger monitor is a CICS program. It determines the names of the initiation queue and queue manager from its environment and then issues an *MQCONN* call for the queue manager and an *MQOPEN* call for the initiation queue. It then issues an *MQGET* call using the *WAIT* option. When there is a message on the initiation queue, the message contents are used to build an *MQTMC2* structure. You need to supply the trigger monitor only with the *StrucId*, *Version*, *QName*, and *QMgrName* parameters, as it doesn't reference any other fields. The program then determines the name of the transaction to be started from the message on the initiation queue and issues an *EXEC CICS START* call for the transaction, passing the *MQTMC2* structure as data. The trigger monitor is a long-running transaction; hence it continues to work with *MQGET* indefinitely. The transaction is terminated by CICS either during normal shutdown or when the queue manager is shutdown.

The program has three functions apart from *main*. The function *return_to_cics ()* issues a CICS log message and returns control to CICS. The function *write_CSMT* handles the issuing of messages to the CICS system log. Lastly the function *read_env* takes one parameter, which is the name of a variable. It searches for this environment variable in the CICS environment and, if finds it, it returns its value.

In *main*, the program completes following steps:

- Initialize CICS.
- Write a start message to the CICS log.
- Open a file for the program's own messages.
- Establish values for the variables *QMGR* and *INITQ*.
- Connect to the queue manager.
- Open the initiation queue
- Loop on 'get messages'.
- Set *MQMD* values as required.
- Get waiting message.
- Get the name of the transaction to be started.
- Issue an *EXEC CICS START* call with the transaction ID and trigger data in *MQTMC2* format.
- Close the initiation queue.
- Disconnect from the queue manager.
- Close the program trace.
- Write an end message to the CICS log.
- Return control to CICS.

TRIGMON2.CCS

```
/*******************************************************************/
/**                                                              **/
/** Program Name        : trigmon2.ccs                           **/
/** Program Author      : Ashish Joshi                           **/
/** Program Description : Trigger Monitor Program                **/
/** This program is a custom trigger monitor                     **/
/** INITQ monitored is specified to CICS region in environment file **/
/** The queue manager name is also picked up from CICS environment  **/
```



```
/**                                                                   **/
/***********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include <cmqc.h>

/***********************************************************************/
/**                                                                   **/
/**  Function prototypes                                              **/
/**                                                                   **/
/***********************************************************************/

int write_CSMT () ;
char * read_env ( );
int return_to_cics () ;

/***********************************************************************/
/**                                                                   **/
/** All the MQ interactions are in main                               **/
/**                                                                   **/
/***********************************************************************/
int main(int argc, char **argv)
{
  char     * QMName;              /** queue manager name     **/
  char     * InitQName;           /** initiation queue name  **/

  /** Declare MQI structures needed                            **/
  MQOD     od = {
    MQOD_DEFAULT    };    /**Object Descriptor   **/
  MQMD     md = {
    MQMD_DEFAULT    };    /**Message Descriptor  **/
  MQGMO   gmo = {
    MQGMO_DEFAULT   };    /**get message options **/

  MQTMC2  trig2={
    MQTMC2_DEFAULT  };    /**trigger message buff**/
  MQTM     trig={
    MQTM_DEFAULT    };    /**trigger message buff**/

  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  **/
MQLONG   buflen;                /**buffer length           **/
MQLONG   triglen;               /**message length received **/

MQCHAR   p1[4];                 /**ApplId insert           **/
MQCHAR   p2[900];               /**trigger insert          **/
MQCHAR   p3[600];               /**Environment insert      **/

/*** Other support variables for trace                     **/
int  i;                         /**auxiliary counter       **/
char     Blank_Line[80] = "                                 " ;

FILE    * errlog ;
char     filename [16] ;

/*********************************************************/
/**                                                      **/
/**  Initialize CICS, Tracefile, and Log starting point  **/
/**                                                      **/
/*********************************************************/

EXEC CICS ADDRESS EIB(dfheiptr);

strncpy ( Blank_Line , "Starting TRIGMON2 ....", 20  ) ;
write_CSMT ( Blank_Line, 20 );

QMName=  read_env ( "QMGR" );
InitQName= read_env ( "INITQ" ) ;

if ( InitQName [0] == NULL )
  strcpy (InitQName , "SYSTEM.CICS.INITIATION.QUEUE" ) ;

strcpy (filename, "/tmp/trigmon2.log" );
strcpy ( Blank_Line , filename );
write_CSMT ( Blank_Line, 20 );

errlog = fopen ( filename , "w+" ) ;

fprintf(errlog, " TRIGMON2 start\n");
fprintf(errlog, " QMName     is %s \n",QMName);
fprintf(errlog, " InitQName  is %s \n",InitQName);
fflush (errlog );

/************************************************************/
/**                                                        **/
/**   Initialize object descriptor for subject queue       **/
/**                                                        **/
/************************************************************/

strcpy(od.ObjectName, InitQName );
```

```
/************************************************************/
/**                                                        **/
/**   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(errlog,
      "MQCONN ended with reason code %ld\n",
      CReason);
  fflush (errlog );
  exit(CReason);
}
fprintf(errlog, " MQCONN Done.\n");
fflush (errlog );

/************************************************************/
/**                                                        **/
/** Open specified initiation queue for input. Exclusive or **/
/** shared use of the queue is controlled by the queue     **/
/** definition                                             **/
/**                                                        **/
/************************************************************/
O_options = MQOO_INPUT_AS_Q_DEF /** open queue for input    **/
+ MQOO_FAIL_IF_QUIESCING;       /** but not if MQM stopping **/
MQOPEN(Hcon,                    /** connection handle       **/
&od,                    /** object descriptor for queue     **/
O_options,              /** open options                    **/
&Hobj,                  /** object handle                   **/
&CompCode,              /** completion code                 **/
&Reason);               /** reason code                     **/

/**  report reason, if any; stop if failed                 **/
if (Reason != MQRC_NONE)
{
  fprintf(errlog, "MQOPEN (%s) ==> %ld\n", od.ObjectName, Reason);
  fflush (errlog );
}
fprintf(errlog, " MQOPEN Done.\n");
fflush (errlog );

OpenCode = CompCode;             /** keep for conditional close**/
```

```
/*****************************************************************/
/**                                                             **/
/**  Get messages from the message queue                        **/
/**  Loop until there is a failure, ask to fail if the queue   **/
/**  manager is quiescing                                       **/
/**                                                             **/
/*****************************************************************/
buflen = sizeof(trig);      /** size of all trigger messages  **/
while (CompCode != MQCC_FAILED)
{
  gmo.Options = MQGMO_WAIT     /** wait for new messages    **/
  + MQGMO_FAIL_IF_QUIESCING    /** or until MQM stopping    **/
  + MQGMO_ACCEPT_TRUNCATED_MSG /** remove long messages     **/
  + MQGMO_SYNCPOINT ;
  gmo.WaitInterval = MQWI_UNLIMITED; /** no time limit      **/

  /*******************************************************/
  /**                                                   **/
  /**  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));
  fprintf(errlog, " MQGET WAIT Starts.\n");
  fflush (errlog );

  /*******************************************************/
  /**                                                   **/
  /**  Wait for a trigger                               **/
  /**                                                   **/
  /*******************************************************/
  MQGET(Hcon,          /**  connection handle          **/
  Hobj,                /**  object handle              **/
  &md,                 /**  message descriptor         **/
  &gmo,                /**  get message options        **/
  buflen,              /**  buffer length              **/
  &trig,               /**  trigger message buffer     **/
  &triglen,            /**  message length             **/
  &CompCode,           /**  completion code            **/
  &Reason);            /**  reason code                **/

  /**  report reason, if any    **/
  if (Reason != MQRC_NONE)
  {
    fprintf(errlog, "MQGET ==> %ld\n", Reason);
    fflush (errlog );
  }
```

```
fprintf(errlog, " MQGET DONE .\n");
fflush (errlog );

/*********************************************************/
/**                                                   **/
/**  Process each message received                    **/
/**                                                   **/
/*********************************************************/
if (CompCode != MQCC_FAILED)
{
  if (triglen != buflen)
  {
    fprintf(errlog, "DataLength = %ld?\n", triglen);
    fprintf(errlog, "Buflen = %ld?\n", buflen);
    fflush (errlog );
  }
  else
  {
    /***************************************/
    /**                                 **/
    /**  Copy trigger message           **/
    /**                                 **/
    /***************************************/

    memcpy(p1, trig.ApplId, sizeof(trig.ApplId));
    memcpy(trig2.StrucId, MQTMC_STRUC_ID,  sizeof(trig2.StrucId));
    memcpy(trig2.Version, MQTMC_VERSION_2,  sizeof(trig2.Version));
    memcpy(trig2.QName,trig.QName , sizeof(trig.QName));
    memcpy(trig2.ProcessName,trig.ProcessName ,
                                     sizeof(trig.ProcessName));
    memcpy(trig2.TriggerData,trig.TriggerData ,
                                     sizeof(trig.TriggerData));
    memcpy(trig2.ApplId,trig.ApplId , sizeof(trig.ApplId));
    memcpy(trig2.EnvData,trig.EnvData , sizeof(trig.EnvData));
    memcpy(trig2.UserData,trig.UserData , sizeof(trig.UserData));
    memcpy(trig2.QMgrName, QMName, sizeof(trig2.QMgrName));

    memcpy(p2, &trig2, triglen);    /** copy modified trigger **/
    p2[triglen] = '\0';

    /**strip trailing blanks**/
    for (i=sizeof(trig2.ApplId)-1; i>=0; i--)
      if (p1[i] != ' ')
        break;
    p1[i+1] = '\0';

    /****************************************/
    /**                                  **/
    /**  Call the USER CICS transaction  **/
    /**                                  **/
```

```
      /****************************************/

      EXEC CICS START
          TRANSID ( p1 )
          FROM ( &trig2 )
          LENGTH ( sizeof (trig2))
          ;

      EXEC CICS SYNCPOINT ;

    }           /** end trigger processing        **/
  }                 /** end process for successful GET **/
}                       /** end message processing loop   **/

/****************************************************************/
/**                                                           **/
/**  Close the initiation 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(errlog, "MQCLOSE ==> %ld\n", Reason);
    fflush (errlog );
  }
  fprintf(errlog, " MQCLOSE DONE .\n");
  fflush (errlog );
}

/*****************************************************************/
/**                                                            **/
/**  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)
    {
      fprintf(errlog, "MQDISC ended with reason code %ld\n", Reason);
      fflush (errlog );
    }
    fprintf(errlog, " MQDISC DONE .\n");
    fflush (errlog );
  }

  fprintf(errlog, " TRIGMON2 end\n");
  fflush (errlog );

  return_to_cics();
}  /** end of main
**/

int  return_to_cics ()
{
  char blank_log[80] = "End of TRIGMON2" ;

  /**write blank line to log**/
  EXEC CICS WRITEQ TD QUEUE("CSMT")
      FROM(blank_log)
      LENGTH(80);

  EXEC CICS RETURN ;
}

int write_CSMT ( char * log_msg , int len ) {
  EXEC CICS WRITEQ TD QUEUE("CSMT")
    FROM(log_msg)
        LENGTH(len)
        ;
}   /** End of function write_CSMT                        **/

extern char ** environ ;
char * read_env ( char * environment_variable ) {

  char * env_var_value ;
  int length_of_varname ;
  int i ; /** Auxiliary counter **/
  char Blank_Line [ 80] ;

  strcat ( environment_variable, "=" );

  env_var_value = malloc  ( MQ_Q_NAME_LENGTH );
  env_var_value[0] = NULL ;

  length_of_varname = strlen ( environment_variable );
```

```
  for ( i=0; environ[i] != NULL; i++ )
  {
    if ( ! strncmp ( environ[i],
        environment_variable ,
        length_of_varname )
        )
    {
      strncpy ( Blank_Line, environ [i]+length_of_varname, 30  );
      strcat ( Blank_Line , "\n" );
      write_CSMT(  Blank_Line , 30);

      strcpy ( env_var_value , environ [i]+length_of_varname );
    }
  }
  return env_var_value ;
}   /** End of function read_env                              **/

/*****************************************************************/
/**                                                            **/
/**  end of trigmon2.ccs                                       **/
/**                                                            **/
/*****************************************************************/
```

ADDITIONAL SET-UP

Add the following to the environment file for CICS region *APPLID*:

```
QMGR=QMGR
INITQ=CICS.APPLID.INITIATION.QUEUE
```

The makefile to build this program on HP-UX 10.20 is as follows (note the use of the continuation character, '➤', in the code below to indicate that one line of code maps to several lines of print):

MAKEFILE

```
DCE_ROOT=/opt/dcelocal
ENCINA_ROOT=/opt/encina
CICS_ROOT=/opt/cics
SYSTEM_CCFLAGS=-Aa +z
DCE_CCFLAGS=-D_REENTRANT -D_HPUX_SOURCE -Dhpux -I/usr/include/reentrant
ENCINA_CCFLAGS=-I$(ENCINA_ROOT)/include
CICS_CCFLAGS=-I$(CICS_ROOT)/include
USER_CCFLAGS=-D_XPG4 -D_HPUX_SOURCE
CCFLAGS=$(SYSTEM_CCFLAGS) $(SAMPLE_CCFLAGS) $(USER_CCFLAGS)
SYSTEM_LDFLAGS=-Bimmediate -Bnonfatal +s +b$(CICS_ROOT)/lib
ENCINA_LDFLAGS=-L$(ENCINA_ROOT)/lib
DCE_LDFLAGS=-L$(DCE_ROOT)/lib
```



```
CICS_LDFLAGS=-L$(CICS_ROOT)/lib
USER_LDFLAGS=-L/opt/mqm/lib
LINKFLAGS=$(SYSTEM_LDFLAGS) $(SAMPLE_LDFLAGS) $(USER_LDFLAGS)
SYSTEM_LDOBJS=/lib/crt0.o
SYSTEM_LDLIBS=-lc
DCE_LDLIBS=-ldce -lndbm -lM -lc_r
ENCINA_LDLIBS=-lEncina
USER_LDLIBS=-lmqmxa_r -lmqm_r
LDLIBS=$(USER_LDLIBS) $(SAMPLE_LDLIBS) $(SYSTEM_LDLIBS) $(DCE_LDLIBS)
➤  $(COBLIBS)
CICSTRAN_ARGS=-e -d

trigmon2 :  trigmon2.ccs
cicstcl -e -d -lC trigmon2
```

CONCLUSION

This article lists some of the common problems faced in using the trigger monitor supplied with MQSeries Version 5.0. It then shows you how to customize your own trigger monitor. All the necessary definitions for the local queue, initiation queue, process for MQ, and environment file changes for CICS are described. Using this article as a guide for writing your own custom trigger monitor should make this a fairly straightforward and manageable task.

Ashish Joshi (USA)

Client/server messages with CICS/ESA

This article outlines some of the issues you should take into consideration when processing MQSeries messages in an OS/390 client/server environment using CICS as the transaction processing monitor. Most large IBM installations tend to use CICS to process their on-line transactions, regardless of whether the transactions are from terminal-based systems (LU 2) or client/server systems (LU 6.2 or TCP/IP). In such environments, it is natural to process MQSeries messages using CICS as well, thus combining the efficiency of CICS with MQSeries' platform and time independence and its relatively simple communications API.

CICS transactions have direct access to the MQI, and there's not much difference between the code for processing MQSeries transactions and an MQSeries batch application, the main ones being that the CICS region connects to the queue manager on your behalf and that syncpointing is coordinated by CICS. However, the CICS application still performs the underlying *MQOPEN*, *MQGET*, *MQPUT*, and *MQCLOSE* calls. It is, therefore, possible to allow application programmers to handle their own queues and messages without providing any 'wrappers' or 'message brokers'.

To simplify matters for application developers, and to add extra functions that many CICS applications need, it's a good idea to develop a CICS 'message broker'. This is not an application integrator in the same sense as the IBM/Neon MQSeries Integrator product, but is instead just a front-end that handles the initial processing of incoming messages.

One of the biggest problems with MQSeries-initiated CICS transactions is that by default they run with the RACF authority either of the CICS region's id or some other central user-id, and not under the RACF authority of the end-user. For on-line client/server systems this is rarely acceptable, as it prevents normal application security from operating at the CICS resource level.

What the message broker does is act as a combination of 'trigger monitor' and 'front-end'. As mentioned in previous articles, the use of the IBM trigger monitor is not appropriate for high-volume client/server applications as it adds an unnecessary overhead. The CICS message broker is a long-running application with an outstanding *MQGET* on the input message queue and is, therefore, always ready to deal with new messages, which means there is no requirement for MQSeries triggering in the traditional 'batch' sense.

Here's what an installation-written CICS message broker should do:

- Accept the name of the input message queue as a parameter for maximum flexibility.

- Retrieve the next incoming message with *MQGET* using convert under syncpoint (this would normally be a destructive get, not a browse). There would be an indefinite wait specified if no messages were available. The *FAIL_IF_QUIESCING* option should be coded.
- Read the message header. This will be in installation-standard layout and will contain (among other items) the name of the target CICS transaction and the caller's user-id and password or token and/or the new password (if changed by the user).
- Validate the user-id and password using CICS security calls. To avoid repeating this overhead, a token can be generated and stored in a table. Subsequent calls can pass the token in place of the password to speed authentication. The token should expire after an appropriate interval and must be passed back to the client program with the reply after it is generated.
- Invoke the desired CICS transaction, passing the message data and other relevant fields in the CICS comm area, starting this CICS transaction with the caller's own user-id.
- Commit the transaction so far and loop back to the *MQGET* logic for the next message.
- The started transaction performs its application logic and, when complete, issues an *MQPUT* to return the result to the reply queue specified in the incoming message data.

Observant readers will detect a problem with persistent messages being lost if the target CICS transaction abends. This can be avoided if the broker copies persistent messages to a second input queue (passing all context) and indicates to the started transaction that it is to read that queue under syncpoint instead of looking in the comm area. For the majority of non-persistent messages, the above method is very efficient, as it requires only one *MQGET* and *MQPUT*, while the trigger monitor method requires two *MQGET*s and two *MQPUT*s for every message received. This alternative (of using a second queue) is also required if the message length exceeds the CICS comm area 32 KB limit.

There is no reason why multiple instances of the message broker can't be executed concurrently, either in the same region or in different regions. Another alternative is to schedule the started CICS transactions to run in different regions, possibly reserving the first region for MQSeries message processing in order to improve resilience.

If a token is to be used for authentication the next time a message requires processing, then the target transaction has to return the token in its reply message. However, there are occasions when the message broker itself should reply to the request, for instance to notify the rejection of the user-id or password, or to notify the user of password expiry. The application needs to be able to handle these return codes and must be able to supply a new password from the user when requested (on expiry or when the user decides to change the password).

This begs the question: what if there isn't a user sitting at the PC? For batch programs to be able to interact with the broker requires that special batch user-ids be validated. One way to do this is to pass information from RACF user data to the broker to validate the batch user-id. The batch job could then be allowed to access its own user data for use as a pseudo-password.

It is also worth noting that CICS/ESA 4.1 is necessary to issue an *EXEC CICS START TRAN* with a user-id parameter, which allows the application to run as a non-terminal CICS task with the authority environment of the real user. The real user's RACF id is defined to RACF as a surrogate of the CICS region id, allowing a task running under the CICS region id to start a task on behalf of the user. The required RACF profile takes the form shown below. Note the use of the continuation character, '➤', to indicate that one line of code maps to several lines of print.

```
RDEFINE SURROGAT *.DFHSTART UACC(NONE) OWNER(*)
PERMIT *.DFHSTART CLASS(SURROGAT) ID(CICS region name)
➤  ACCESS(READ)
```

MQSeries Specialist (UK) © Xephon 1999

Monitoring MQ events from the mainframe

The utility in this article provides a cheap way of monitoring and logging MQ events on non-mainframe servers from a mainframe. The utility is quick to set up and can be modified easily.

BACKGROUND

We have several MQ servers running on platforms other than the mainframe (principally OS/2 systems). When we first set up these servers, we had no means of reporting on MQ events being generated on them. A quick solution to this problem was to port events from our OS/2 MQ servers to the mainframe, where we run most of our automation and problem management software.

We already had an MQ server on the mainframe (called *INM1*) that could be used to receive messages from the OS/2 servers.

I decided to tackle the problem using REXX, and so downloaded a copy of the REXX/MQ support pack from the IBM Web site. This allowed me quickly to develop an application that would be able to retrieve messages from *INM1*'s queues and format the output to present to our automation product, NetView/AOC.

HOW IT WORKS

MQ events are written to system queues, so I've altered these queues so they're aliases of a remote queue definition. Any message that is put on the system event queue is ported across to the mainframe MQ server. The following definitions are an example of how I connected an OS/2 server (*SL028457*) to our mainframe MQ server (*INM1*).

MQ DEFINITIONS ON SL028457

```
DELETE QLOCAL('SYSTEM.ADMIN.QMGR.EVENT')

DEFINE QALIAS('SYSTEM.ADMIN.QMGR.EVENT') REPLACE +
       DESCR('Alias name for system queue manager events') +
       TARGQ('SL028457.EVENT.REMOTEQ')
```

```
DELETE QLOCAL('SYSTEM.ADMIN.PERFM.EVENT')

DEFINE QALIAS('SYSTEM.ADMIN.PERFM.EVENT') REPLACE +
       DESCR('Alias name for system performance events') +
       TARGQ('SL028457.EVENT.REMOTEQ')

ALTER QMGR PERFMEV(ENABLED)
ALTER QMGR STRSTPEV(ENABLED)

DEFINE CHANNEL(SL028457.TO.INM1) chltypE(SDR) TRPTYPE(TCP) +
        REPLACE CONNAME(10.1.1.85) XMITQ(SL028457.TO.INM1.XMITQ) +
        DESCR('Channel to remote qmgr')

DEFINE QLOCAL(SL028457.TO.INM1.XMITQ) USAGE(XMITQ) +
        REPLACE PROCESS(SL028457.TO.INM1.SEND.PROCESS) +
        DEFPSIST(NO) +
        TRIGGER TRIGTYPE(EVERY) TRIGDPTH(1) +
         INITQ(SYSTEM.CHANNEL.INITQ) +
        GET(ENABLED) PUT(ENABLED) +
        DESCR('Transmission queue')

DEFINE QREMOTE(SL028457.EVENT.REMOTEQ) +
        RNAME(SL028457.EVENT.LOCALQ) +
        REPLACE RQMNAME(INM1) XMITQ(SL028457.TO.INM1.XMITQ) +
        DEFPSIST(NO) +
        PUT(ENABLED) +
        DESCR('Points to local queue on remote qmgr')

DEFINE PROCESS(SL028457.TO.INM1.SEND.PROCESS) +
        REPLACE APPLTYPE(OS2) USERDATA(SL028457.TO.INM1)
```

MQ DEFINITIONS ON INM1

```
DEFINE CHANNEL('SL028457.TO.INM1') +
       CHLTYPE(RCVR) +
       REPLACE +
       DESCR('Channel for receiving messages from SL028457')

DEFINE QLOCAL('SL028457.EVENT.LOCALQ') +
      REPLACE +
      PUT(ENABLED) +
      GET(ENABLED) +
      SHARE +
      DEFSOPT(EXCL) +
      MSGDLVSQ(FIFO) +
      DESCR('Local Queue for SL028457 events')
```

GETTING CODE FROM INM1

As mentioned previously, I’ve used REXX and the REXX support pack to develop an application that can get messages from the local MQ queues. I’ve set the following options in the ‘get options’ section of the REXX code:

- *mqgmo_wait*
  GET waits for messages to appear in the MQ queue.
- *mqgmo_fail_if_quiescing*
  GET completes even if MQ server is shutting down.
- *mqgmo_convert*
  Convert ASCII to EBCDIC.
- *mqwi_unlimited*
  No wait time set.

The main function of the code is to parse event messages and issue a console WTO for AOC to trap. AOC then passes the console message to Solve, our problem management system, to record an ‘incident’ against the server.

Multiple servers can pass events to the same MVS *localq*. The code handles this by parsing the server name from the message and including it in the MVS WTO. If you do this, I recommend setting the local queue name to reflect the fact that multiple servers are putting event messages on the queue.

Note the use of the continuation character, ‘➤’, in the code below to indicate that one line of code maps to several lines of print.

REXX CODE FOR APPLICATION

```
/* REXX
*/

arg mq_server mq_queue .

/* Initialize the interface */

RXMQVTRACE = ''
return_code= RXMQV('INIT')
say 'init rc='word(return_code,1)
```

```
/* Connect to Queue Manager */

RXMQVTRACE = ''
return_code = RXMQV('CONN',mq_server)
say 'conn rc='word(return_code,1)

/* Open localq */

RXMQVTRACE = ''
return_code = RXMQV('OPEN',mq_queue,mqoo_input_shared,'h2','ood.')
say 'open rc='word(return_code,1)

/* Get messages from queue (loop) */

do i=1
  message.0 = 250
  message.1 = ""
  igmo.opt  = mqgmo_wait+mqgmo_fail_if_quiescing+mqgmo_convert
  igmo.wait = mqwi_unlimited
  RXMQVTRACE = ''
  return_code =
  ➤  RXMQV('GET',h2,'message.','igmd.','ogmd.','igmo.','ogmo.')
  say 'get  rc='word(return_code,1)
  if (word(return_code,1) <> 0) then leave

  server = substr(message.1,57,8)

  localq = substr(message.1,125,44)

  alert_char = substr(message.1,31,2)

  alert_hex  = c2x(alert_char)

  select
   when (alert_hex = '08B0') then do
                                alert  = 'Queue depth HIGH'
                                msg_id = 'SLMQ0051'
                           end
   when (alert_hex = '0805') then do
                                alert = 'Queue FULL'
                                msg_id = 'SLMQ0053'
                           end
   when (alert_hex = '08B1') then iterate
   when (alert_hex = '08AE') then iterate
   otherwise do
             alert = 'Unknown('||alert_hex||')'
             msg_id = 'SLMQ0059'
             end
  end
```

```
  wto_message = msg_id||' MQmgr='||server||', Alert='||alert||,
                ', Time='ogmd.pt||,
                ', Date='ogmd.pd||,
                ', Queue name='||localq

  address linkmvs ldmwto1 'wto_message'

  say wto_message

  /* Use for diagnostics
  say 'ogmd.pd    ' ogmd.pd
  say 'ogmd.pt    ' ogmd.pt
  say 'ogmd.form  ' ogmd.form
  say 'ogmd.pan   ' ogmd.pan
  say 'ogmd.rtoqm ' ogmd.rtoqm

  do loop_zlist = 1 to words(ogmo.zlist)
     ts = word(ogmo.zlist,loop_zlist)
     ogmo.ts = translate(ogmo.ts,'','00'x)
     ogmo.ts = Strip(ogmo.ts,'B')
     say 'ogmo.'ts ogmo.ts
  end
  */

end

/* Close localq */

RXMQVTRACE = ''
return_code = RXMQV('CLOSE',h2,mqco_none)
say 'close rc='word(return_code,1)

/* Disconnect queue manager  */

RXMQVTRACE = ''
return_code = RXMQV('DISC',queue_manager)
say 'disc rc='word(return_code,1)

/* Remove the interface functions from the REXX workspace ... */

RXMQVTRACE = ''
return_code = RXMQV('TERM',)
say 'term rc='word(return_code,1)

return
```

I've created the started task to run the REXX code for which the JCL is listed below.

JCL FOR STARTED TASK TO RUN REXX CODE

```
//MVMQMP    PROC
//MONITOR   EXEC PGM=IRXJCL,
//                     PARM='MQMON INM1 SL028457.EVENT.LOCALQ'
//STEPLIB   DD DISP=SHR,DSN=SGMX.MASTER.LOAD    * REXX loadlib *
//          DD DISP=SHR,DSN=IN.MTMQSZDX.SCSQANLE
//          DD DISP=SHR,DSN=IN.MTMQSZDX.SCSQAUTH
//SYSEXEC   DD DISP=SHR,DSN=SG.DEVT.EXEC                * REXX execs */
//SYSTSPRT DD SYSOUT=*
```

'Parms' passed to the REXX code are set as follows:

- *MQMON*
  REXX code (held in //SYSEXEC)

- *INM1*
  MVS MQ server name

- *SL028457.EVENT.LOCALQ*
  MQ local queue for events.

Calum Reid
Systems Programmer (UK)

An MQSeries batch trigger monitor for MVS/ESA

While a trigger monitor for CICS is supplied with MQSeries for MVS/ESA, one is not provided for the batch environment. The Support Pac's MA12, however, comprises two sample programs that can be used as a batch trigger monitor (for further information on them, see *www.software.ibm/ts/mqseries/txppacs/ma12.html*). I've written a batch trigger monitor that is similar to the Support Pac versions but differs from them in three significant ways. Firstly, the programming language of the main routine is PL/I (not COBOL). Secondly, the monitor is stopped by the operator via the *MODIFY* command (the Support Pac version is stopped by placing a special message on the initiation queue). Thirdly, the submit process is handled by TSO's

SUBMIT command (as opposed to reading JCL cards and passing them to INTRDR).

The monitor is started as a task and comprises two programs. The main program, MQSBAT1 (PL/I), runs continuously, monitoring an initiation queue for incoming trigger messages. When a message arrives on the queue, the program retrieves a dataset name from the trigger data and generates a *TSO SUBMIT(dsn)* job that's passed to JES via INTRDR. The name of the initiation queue and the wait interval for the *MQGET* (*MQGMO_WAIT*) are passed to the program via the JCL *PARM* parameter.

Program MQSBAT2 (ASSEMBLER) is called periodically by the main program to check the console's command input buffer for a *MODIFY* shutdown request (*F jobname,SHUTDOWN*) from the operator. WTO messages are also written by MQSBAT2 if the batch trigger monitior terminates processing abnormally.

IMPLEMENTATION REQUIREMENTS

1 Define one initiation queue (for instance, *BATCH.INITQ*) as set out below:

```
Queue name . . . . . . . . . . : BATCH.INITQ
  Description  . . . . . . . . : Batch trigger initiation queue
  Put enabled  . . . . . . . . : Y  Y=Yes,N=No
  Default persistence  . . . . : Y  Y=Yes,N=No
  Default priority . . . . . . : 0  0 - 9
  Get enabled  . . . . . . . . : Y  Y=Yes,N=No
 Message delivery sequence . . : F  P=Priority,F=FIFO
 Permit shared access  . . . . : Y  Y=Yes,N=No
 Default share option  . . . . : E  E=Exclusive,S=Shared
 Maximum queue depth . . . . . : 100        0 - 999999999
 Maximum message length  . . . : 4096       0 - 4194304
 Retention interval  . . . . . : 999999999  0 - 999999999 hours
 Usage . . . . . . . . . . . . : N  N=Normal,X=XmitQ
 Storage class . . . . . . . . : SYSTEM
 Trigger type  . . . . . . . . : N  F=First,E=Every,D=Depth,N=Non
    Trigger set  . . . . . . . : N  Y=Yes,N=No
    Trigger message priority : 0  0 - 9
    Trigger depth  . . . . . . : 1          1 - 999999999
    Trigger data . . . . . . . :
    Process name . . . . . . . :
    Initiation queue . . . . . :
```

2 Define one process (for instance *BATCH.PROCESS*), as laid out below:

```
Process name . . . . . . . . BATCH.PROCESS
Description  . . . . . . . : Universal process for batch trigger
Application type . . . . . : MVS
Application ID . . . . . . :
User data  . . . . . . . . :
Environment data . . . . . :
```

3 Generate JCL for the started task:

```
//MQSBTMO EXEC PGM=MQSBAT1,PARM=('BATCH.INITQ,5000')       (1)
//STEPLIB  DD  DSN=user.LOADLIB,DISP=SHR                   (2)
//                   DD  DSN=thq.SCSQAUTH,DISP=SHR         (3)
//SYSPRINT DD  SYSOUT=A                                    (4)
//SYSUDUMP DD  SYSOUT=*
//INTRDR   DD  SYSOUT=(*,INTRDR)
```

1 PARM values for intiation queue name and wait interval in msec

2 Library containing MQSBAT1 and MQSBAT2

3 Library containing MQSeries product load modules and CSQBDEFV for QMGR default connect

4 Output dataset for batch trigger monitor submit and error log messages.

The way to request the submission of a batch job via the trigger monitor is relatively straightforward. Only the definition trigger attributes of the local application queue are necessary:

```
Trigger Definition

Trigger type  . . . . . . . : F  F=First,E=Every,D=Depth,N=None

   Trigger set  . . . . . . : Y  Y=Yes,N=No
   Trigger message priority : 0  0 - 9
   Trigger depth  . . . . . : 1          1 - 999999999
   Trigger data . . . . . . : BATCH.JCL(TESTJOB)           (1)
   Process name . . . . . . : BATCH.PROCESS                (2)
   Initiation queue . . . . : BATCH.INITQ                  (3)
```

1 Job TESTJOB is submitted using the TSO *SUBMIT(dsn)* command

2 Application-related process definitions are not required, so one global process definition is sufficient

3 The trigger message is generated on initiation queue BATCH.INITQ.

Protecting the batch trigger monitor from jobs submitted by unauthorized users can be done in one of two ways. Firstly, the program MQSBAT1 can be extended to retrieve context information from the application queue and hence obtain the *UserIdentifier* (which identifies the user that originated the message). The TSO *SUBMIT* command job can then be submitted under this userid (*USER=*). The second method, which requires no modification to MQSBAT1, is to define MQQUEUE security for the local application queue, so that only authorized users are allowed to put messages on the local application queue and therefore to request a job to be submitted.

Thanks to Klaus Langnau from SCIC Consulting (*www.scic.de/*) for his contribution to this MQSeries batch trigger monitor.

MQSBAT1

```
MQSBAT1: PROC(PARM_DATA) OPTIONS(MAIN) REORDER;
 /***************************************************************/
 /*       PROGRAM-HISTORY                                       */
 /*SOURCE MEMBER: MQSBAT1                                       */
 /*AUTHOR       : KLEEBAUR/LANGNAU                              */
 /*DATE         : 05.02.1999                                    */
 /*FUNCTION     : MQ/SERIES BATCH TRIGGER MONITIOR.             */
 /*                                                             */
 /*               THIS CONTINUOUSLY RUNNING PROGRAM SERVES AN   */
 /*               INITIATION QUEUE FOR BATCH JOB SUBMITS.       */
 /*               WHEN A TRIGGER MESSAGE ARRIVES ON THE QUEUE,  */
 /*               THE PROGRAM RETRIEVES THE DATA SET NAME FROM  */
 /*               THE TRIGGER DATA AND GENERATES A TSO SUBMIT JOB, */
 /*               WHICH IS PASSED TO JES VIA INTRDR.            */
 /*                                                             */
 /*               PARAMETER PASSED:                             */
 /*               THE NAME OF THE INIT QUEUE AND THE WAIT INTERVAL */
 /*               VALUE (MQGMO_WAIT) ARE PASSED VIA THE JCL PARM   */
 /*               PARAMETER TO THE PROGRAM.                     */
 /*                                                             */
 /*               EXTERNAL REFERENCES:                          */
 /*               THE CALLED PGM MQSBAT2 CHECKS THE CONSOLE     */
```

```
/*               COMMAND INPUT BUFFER IF THERE IS A SHUTDOWN      */
/*               REQUEST FROM THE OPERATOR. WTO MESSAGES ARE ALSO */
/*               WRITTEN BY MQSBAT2 IF THE BATCH TRIGGER MONITOR  */
/*               TERMINATES PROCESSING ABNORMALLY.                */
/******************************************************************/

/******************************************************************/
/*                   E X T E R N A L  R E F E R E N C E S        */
/******************************************************************/
        DCL MQSBAT2 OPTIONS(ASSEMBLER) EXTERNAL ENTRY(POINTER);

/******************************************************************/
/*                   J C L  P A R M  PARAMETER DATA               */
/******************************************************************/
        DCL PARM_DATA          CHAR(*) VAR;
        DCL PARM_QUEUENAME     CHAR(48)            INIT('');
        DCL PARM_WAITINTERVAL  FIXED BIN(31)       INIT(0);

/******************************************************************/
/*                   F I L E  D E C L A R A T I O N S             */
/******************************************************************/
        DCL SYSPRINT FILE STREAM OUTPUT PRINT;
        DCL INTRDR   FILE RECORD OUTPUT SEQL UNBUF
                                 ENV (F RECSIZE(80));
/******************************************************************/
/*                   B U I L T I N S                              */
/******************************************************************/
        DCL  ADDR            BUILTIN;
        DCL  DATETIME        BUILTIN;
        DCL  BINARY          BUILTIN;
        DCL  INDEX           BUILTIN;
        DCL  SUBSTR          BUILTIN;
        DCL  REPEAT          BUILTIN;
        DCL  NULL            BUILTIN;

/******************************************************************/
/*                   W O R K I N G  STORAGE                       */
/******************************************************************/
        DCL  SYS_DATETIME    CHAR(20)             INIT('');
        DCL  INT_YYYY        CHAR(04)             INIT('');
        DCL  INT_MM          CHAR(02)             INIT('');
        DCL  INT_DD          CHAR(02)             INIT('');
        DCL  INT_HOUR        CHAR(02)             INIT('');
        DCL  INT_MIN         CHAR(02)             INIT('');
        DCL  INT_SEC         CHAR(02)             INIT('');
        DCL SUBMIT_COUNTER   FIXED BIN(15)        INIT(0);
        DCL ERROR_COUNTER    FIXED BIN(15)        INIT(0);
        DCL OFF              CHAR(01)             INIT('0');
        DCL ON               CHAR(01)             INIT('1');
```



```
/****************************************************************/
/*                  M Q S  I N C L U D E S                      */
/****************************************************************/
  %INCLUDE SYSLIB(CMQP);
  %INCLUDE SYSLIB(CMQEPP);

/****************************************************************/
/*                   M Q S  WORKING STORAGE                     */
/****************************************************************/
        DCL HCONN         FIXED BIN(31)    INIT(0);
        DCL OBJDESC       LIKE MQOD;
        DCL OPEN_OPTIONS  FIXED BIN(31)    INIT(0);
        DCL CLOSE_OPTIONS FIXED BIN(31)    INIT(0);
        DCL HOBJ          FIXED BIN(31)    INIT(0);
        DCL COMPCODE      FIXED BIN(31)    INIT(0);
        DCL REASON        FIXED BIN(31)    INIT(0);
        DCL MGR_NAME      CHAR(48)         INIT('');
        DCL MSGDESC       LIKE MQMD;
        DCL GETMSGOPTS    LIKE MQGMO;
        DCL DATALENGTH    FIXED BIN(31)    INIT(0);
        DCL BUFFERLENGTH  FIXED BIN(31)    INIT(4096);
        DCL BUFFER        CHAR(4096);
        DCL TMPTR         POINTER;

/****************************************************************/
/*                  C O M M A R E A  FOR MQSBAT2                */
/****************************************************************/
        DCL   MQSBAT2_PTR POINTER;
        DCL 1 MQSBAT2_PARM ALIGNED,
             2  MQSBAT2_WORKAREA,
              3  SAVEAREA     (18) FIXED BIN(31)  INIT((18)0),
              3  COMMADDR     FIXED BIN(31)       INIT(0),
              3  RETB         CHAR(28)            INIT(''),
              3  ZRETB        CHAR(24)            INIT(''),
             2  MQSBAT2_FCODE CHAR(01)            INIT(''),
             2  CON_FLAG      CHAR(01)            INIT('0');

/*****************************************************************/
/*                   J C L  FOR TSO SUBMIT COMMAND               */
/*****************************************************************/
     DCL 1 JOB_CARDS,
          2 JOB_CARD01    CHAR(80)
            INIT('//MQSUBMIT JOB  ,CLASS=S,MSGCLASS=Z'),
          2 JOB_CARD02    CHAR(80)
            INIT('//SUBMIT   EXEC PGM=IKJEFT01,DYNAMNBR=20'),
          2 JOB_CARD03    CHAR(80)
            INIT('//SYSPRINT    DD   SYSOUT=*'),
          2 JOB_CARD04    CHAR(80)
            INIT('//SYSTSPRT    DD   SYSOUT=*'),
          2 JOB_CARD05    CHAR(80)
```

```
         INIT('//SYSUDUMP    DD   SYSOUT=*'),
        2 JOB_CARD06    CHAR(80)
         INIT('//SYSTSIN     DD *'),
        2 JOB_CARD07,
         3 JOB_SUBMIT   CHAR(08) INIT(' SUBMIT '),
         3 JOB_DSN      CHAR(72) INIT(''),
        2 JOB_CARD08    CHAR(80)
         INIT('/*'),
        2 JOB_CARD09    CHAR(80)
         INIT('/*EOF');

  DCL 1 JOB_TAB(09) CHAR(80) BASED(ADDR(JOB_CARDS));
  DCL   JOB_MAX     FIXED(3) INIT(09);
  DCL   JOB_I       FIXED(3) INIT(0);
  DCL   JOB_CARD    CHAR(80) INIT('');

/*********************************************************************/
/*         PROGRAM MAINCONTROL                                       */
/*********************************************************************/
PROLOG:
        OPEN FILE(SYSPRINT);
        OPEN FILE(INTRDR);

        CALL GET_PARM;                  /* PARAMETERS FROM JCL      */

        TMPTR = ADDR(BUFFER);           /* ADDR MQS IO-BUFFER       */

        FETCH MQSBAT2;                  /* LOAD SERVICE PROGRAM     */

        MQSBAT2_PTR = ADDR(MQSBAT2_PARM);  /* ADDR COMMAREA POINTER */

        MQSBAT2_FCODE = '1';            /* OBTAIN CIB ADDRESS        */
        CALL MQSBAT2(MQSBAT2_PTR);      /* PASS PTR TO COMMAREA PTR */

BODY:
        CALL MQ_CONNECT;                /* CONNECT TO QMGR           */
        CALL MQ_OPEN;                   /* OPEN INIT QUEUE           */
        DO WHILE (CON_FLAG = OFF);      /* LOOP UNTIL SHUTDOWN CMD  */
           CALL MQ_GET;                 /* GET MQ MESSAGE            */
           IF COMPCODE = MQCC_OK THEN CALL SUBMIT_JOB;
           CALL MQ_COMMIT;
           MQSBAT2_FCODE = '2';         /* LOOK FOR CONSOLE COMMAND */
           CALL MQSBAT2(MQSBAT2_PTR);   /* PASS PTR TO COMMAREA PTR */
        END;
        CALL MQ_CLOSE;                  /* CLOSE INIT QUEUE          */
        CALL MQ_DISCON;                 /* DISCONNECT FROM QMGR      */
        GOTO EPILOG;
ERROR:
        MQSBAT2_FCODE = '3';            /* REQUEST WTOR MESSAGE      */
        CALL MQSBAT2(MQSBAT2_PTR);      /* PASS PTR TO COMMAREA PTR */
```



```
EPILOG:
        RELEASE MQSBAT2;                /* RELEASE SERVICE PROGRAM  */
        PUT SKIP FILE(SYSPRINT)
            ('JOBS SUBMITTED             = ' || SUBMIT_COUNTER);
        PUT SKIP FILE(SYSPRINT)
            ('DSN MISSING FOR JOB SUBMIT = ' || ERROR_COUNTER);
        CLOSE FILE(SYSPRINT),
              FILE(INTRDR);

/*********************************************************************/
/*         SUBROUTINES                                              */
/*********************************************************************/

/*********************************************************************/
/*         GET QUEUE NAME AND WAIT INTERVAL FROM JCL PARM           */
/*********************************************************************/
GET_PARM: PROC REORDER;

        PARM_QUEUENAME =
        SUBSTR(PARM_DATA,1,INDEX(PARM_DATA,',')-1);
        PARM_WAITINTERVAL =
        BINARY(SUBSTR(PARM_DATA,INDEX(PARM_DATA,',')+1),31,0);


END GET_PARM;
/*********************************************************************/
/*         CONNECT TO THE DEFAULT QUEUEMANAGER (VIA CSQBDEFV)       */
/*********************************************************************/
MQ_CONNECT: PROC REORDER;

        MGR_NAME = '';
        CALL MQCONN (MGR_NAME,
                     HCONN,
                     COMPCODE,
                     REASON);

        IF COMPCODE ¬= MQCC_OK THEN
          DO;
             PUT SKIP FILE(SYSPRINT)
                 ('** MQCONN ** ERROR' || COMPCODE || REASON);
             GOTO ERROR;
          END;

END MQ_CONNECT;

/*********************************************************************/
/*         DISCONNECT FROM THE DEFAULT QUEUEMANAGER                 */
/*********************************************************************/
MQ_DISCON: PROC REORDER;
```



```
        CALL MQDISC(HCONN,
                    COMPCODE,
                    REASON);

        IF COMPCODE ¬= MQCC_OK THEN
         DO;
             PUT SKIP FILE(SYSPRINT)
                 ('** MQDISC ** ERROR' || COMPCODE || REASON);
             GOTO ERROR;
         END;

END MQ_DISCON;

/*******************************************************************/
/*        OPEN THE INITQUEUE                                       */
/*******************************************************************/
MQ_OPEN: PROC REORDER;

        OBJDESC.OBJECTNAME = PARM_QUEUENAME;
        OPEN_OPTIONS = MQOO_INPUT_SHARED
                     + MQOO_FAIL_IF_QUIESCING;

        CALL MQOPEN(HCONN,
                    OBJDESC,
                    OPEN_OPTIONS,
                    HOBJ,
                    COMPCODE,
                    REASON);

        IF COMPCODE ¬= MQCC_OK THEN
         DO;
             PUT SKIP FILE(SYSPRINT)
                 ('** MQOPEN ** ERROR' || COMPCODE || REASON);
             GOTO ERROR;
         END;

END MQ_OPEN;

/*******************************************************************/
/*        CLOSE THE INITQUEUE                                      */
/*******************************************************************/
MQ_CLOSE: PROC REORDER;

        CLOSE_OPTIONS = MQCO_NONE;
        CALL MQCLOSE (HCONN,
                    HOBJ,
                    CLOSE_OPTIONS,
                    COMPCODE,
                    REASON);
```



```
        IF COMPCODE ¬= MQCC_OK THEN
         DO;
             PUT SKIP FILE(SYSPRINT)
                 ('** MQCLOSE** ERROR' || COMPCODE || REASON);
             GOTO ERROR;
         END;

END MQ_CLOSE;

/*********************************************************************/
/*         READ MESSAGES FROM INIT QUEUE                             */
/*         WITH WAIT INTERVAL AND QMGR QUIESCING OPTION              */
/*********************************************************************/
MQ_GET: PROC REORDER;

        GETMSGOPTS.OPTIONS = MQGMO_WAIT
                           + MQGMO_FAIL_IF_QUIESCING;
        GETMSGOPTS.WAITINTERVAL = PARM_WAITINTERVAL;

        MSGDESC.MSGID    = MQMI_NONE;
        MSGDESC.CORRELID = MQCI_NONE;

        CALL MQGET(HCONN,
                   HOBJ,
                   MSGDESC,
                   GETMSGOPTS,
                   BUFFERLENGTH,
                   BUFFER,
                   DATALENGTH,
                   COMPCODE,
                   REASON);

        IF COMPCODE = MQCC_OK |
           (COMPCODE = MQCC_FAILED &
            REASON = MQRC_NO_MSG_AVAILABLE) THEN;
        ELSE
           DO;
             PUT SKIP FILE(SYSPRINT)
                 ('** MQGET** ERROR' || COMPCODE || REASON);
             GOTO ERROR;
           END;

END MQ_GET;

/********************************************************************/
/*               SUBMIT JOB VIA IKJEFT01                            */
/*                                                                  */
/*   FOLLOWING TRIGGER MESSAGE FIELDS ARE AVAILABLE:                */
/*               TMPTR->MQTM.QNAME                                  */
/*               TMPTR->MQTM.PROCESSNAME                            */
```

```
/*               TMPTR->MQTM.TRIGGERDATA                               */
/*               TMPTR->MQTM.APPLTYPE                                  */
/*               TMPTR->MQTM.APPLID                                    */
/*               TMPTR->MQTM.ENVDATA                                   */
/*               TMPTR->MQTM.USERDATA                                  */
/*   THE SUBMIT DATA SET NAME WILL BE PAST IN FIELD                    */
/*                                         MQMT.TRIGGERDATA            */
/*********************************************************************/
SUBMIT_JOB: PROC REORDER;

        IF TMPTR->MQTM.TRIGGERDATA  = '' THEN
           DO;
              ERROR_COUNTER = ERROR_COUNTER + 1;
           END;
        ELSE
           DO;
            JOB_DSN = '';
            JOB_DSN = '(''' ||
                      SUBSTR(TMPTR->MQTM.TRIGGERDATA,1,
                      INDEX(TMPTR->MQTM.TRIGGERDATA,' ')-1)
                            || ''')';

            DO JOB_I = 1 TO JOB_MAX;
               JOB_CARD = JOB_TAB(JOB_I);
               WRITE FILE (INTRDR) FROM (JOB_CARD);
            END;
            SUBMIT_COUNTER = SUBMIT_COUNTER + 1;
            CALL LOG_SUBMIT;
          END;

END SUBMIT_JOB;

/**********************************************************************/
/*         LOG JOBSUBMIT ON SYSPRINT                                  */
/**********************************************************************/
LOG_SUBMIT: PROC REORDER;

        SYS_DATETIME  = DATETIME;
        INT_YYYY      = SUBSTR(SYS_DATETIME,01,04);
        INT_MM        = SUBSTR(SYS_DATETIME,05,02);
        INT_DD        = SUBSTR(SYS_DATETIME,07,02);
        INT_HOUR      = SUBSTR(SYS_DATETIME,09,02);
        INT_MIN       = SUBSTR(SYS_DATETIME,11,02);
        INT_SEC       = SUBSTR(SYS_DATETIME,13,02);

        PUT SKIP FILE(SYSPRINT)
            ('SUBMIT ' || SUBSTR(JOB_DSN,1,46) || ' ON '
            || INT_DD || '.' || INT_MM || '.' || INT_YYYY ||
            ' AT ' || INT_HOUR || ':' || INT_MIN  || ':' || INT_SEC
            || ' TRIGGERED BY QUEUE ' || TMPTR-> MQTM.QNAME);
```

```
END LOG_SUBMIT;

/******************************************************************/
/*        COMMIT MQ_GET                                           */
/******************************************************************/
MQ_COMMIT: PROC REORDER;

        CALL MQCMIT (HCONN,
                    COMPCODE,
                    REASON);

        IF COMPCODE ¬= MQCC_OK THEN
         DO;
             PUT SKIP FILE(SYSPRINT)
                 ('** MQCOMIT** ERROR' || COMPCODE || REASON);
             GOTO ERROR;
         END;

END MQ_COMMIT;

END MQSBAT1;
```

MQSBAT2

```
**********************************************************************
*        PROGRAM-HISTORY                                             *
*                                                                    *
* SOURCE MEMBER: MQSBAT2                                             *
* AUTHOR       : KLEEBAUR                                            *
* DATE         : 02.02.1999                                          *
* FUNCTION     : CONSOLE COMMUNICATION FOR MQ/SERIES BATCH TRIGGER.  *
*                THE OPERATOR CAN PASS SHUTDOWN INFORMATION FOR THE  *
*                STARTED BATCH TRIGGER MONITOR BY ISSUING A MODIFY   *
*                COMMAND.                                            *
*                                                                    *
**********************************************************************
         EJECT
**********************************************************************
*        MACROS                                                      *
*                                                                    *
**********************************************************************
         MACRO
**********************************************************************
* MACRO        : ENTUP                                               *
* FUNCTION     : SAVING BAL REGISTER WHEN ENTERING SUBROUTINE.       *
**********************************************************************
&NAME    ENTUP &REG
&NAME    MVC   $ZRETB(L'$RETB-4),$RETB
         MVC   $RETB+4(L'$RETB-4),$ZRETB
```

```
         ST    &REG,$RETB
         MEND
         MACRO
**********************************************************************
* MACRO         : RETUP                                              *
* FUNCTION      : RESTORING BAL REGISTER WHEN LEAVING SUBROUTINE.    *
**********************************************************************
&NAME   RETUP &REG
&NAME   L     &REG,$RETB
        MVC   $RETB(L'$RETB-4),$RETB+4
        BR    &REG
        MEND
**********************************************************************
*        REGISTER EQUATES                                            *
*                                                                    *
**********************************************************************
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3                   BASE REGISTER
R4       EQU   4
R5       EQU   5
R6       EQU   6                   ADDR OF CALLER'S PARAMETER AREA
R7       EQU   7                   ADDR OF COMMAND INPUT BUFFER AREA
R8       EQU   8
R9       EQU   9                   ADDR OF COMMUNICATION AREA
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         EJECT
         PRINT GEN
**********************************************************************
*        DSECTS                                                      *
*                                                                    *
**********************************************************************
PARMAREA DSECT                     PARAMETER PASSED BY CALLER
SAVE     DS    18F                 SAVE AREA
COMADDR  DS    F                   ADDR(COMAREA) FROM EXTRACT
$RETB    DS    CL28                FIELDS FOR ENTUP/RETUP-MACROS
$ZRETB   DS    CL24                7 LEVELS POSSIBLE
FCODE    DS    CL01                '1'  = OBTAIN CIB ADDR
*                                  '2'  = WATCH CONSOLE
*                                  '3'  = WTO
CONFLAG  DS    CL01                '0'  = NO COMMAND ENTERED
*                                  '1'  = COMMAND ENTERED
COM      DSECT
         IEZCOM   ,                 COMAREA
```

```
*
CIB      DSECT
         IEZCIB   ,                 CIB
**********************************************************************
*        PROGRAM-MAINCONTROL                                         *
*                                                                    *
**********************************************************************
MQSBAT2 CSECT
MQSBAT2 AMODE 31
MQSBAT2 RMODE ANY
A000     EQU *
         SAVE  (14,12)             SAVE CALLER'S REGS IN CALLER-
*                                  PROVIDED R13 SAVE AREA
         BALR  R3,R0               INIT BASE REGISTER
         USING *,R3                ESTABLISH ADDRESSABILITY
*
         B     A100                PAST EYECATCHER
         DC    CL8'MQSBAT2'        PROGRAM NAME
         DC    CL8'&SYSDATE'       DATE ASSEMBLED
         DC    CL6'&SYSTIME'       TIME ASSEMBLED
A100     EQU *
         L     R6,0(R1)            R1 POINTS TO ADDR OF COMMAREA PTR
         L     R6,0(R6)            LOAD ADDR OF COMMAREA
         USING PARMAREA,R6         ESTABLISH ADDRESSABILITY
*
         ST    R13,SAVE+4          SAVE ADDR OF CALLER'S SAVE AREA
*                                  IN MY SAVE AREA (BACKWARD CHAIN)
         LR    R11,R13
         LA    R13,SAVE            PUT MY SAVE AREA ADDR IN R13
         ST    R13,8(R11)          SAVE MY SAVE AREA IN CALLER' S
*                                  SAVE AREA (FORWARD CHAIN)
A200     EQU *                     FUNCTION ROUTING
         CLI   FCODE,C'1'
         BNE   A300
         BAL   R2,ADCIB00          ADDRESS THE CIB
         B     A900
A300     CLI   FCODE,C'2'
         BNE   A400
         BAL   R2,LKCMD00          LOOK FOR CONSOLE COMMAND
         B     A900
A400     CLI   FCODE,C'3'
         BNE   A900
         BAL   R2,WTOMS00          WRITE WTO ERROR MESSAGE
A900     EQU *
         XR    R15,R15
         L     R13,SAVE+4          RESTORE CALLER'S SAVE AREA ADDR
         RETURN (14,12),RC=(15)    RESTORE CALLER'S REGS AND RETURN
         EJECT
**********************************************************************
*        OBTAIN ADDRESS OF THE CIB AND                               *
```

```
*        ALLOW CIBS TO BE CHAINED                                    *
**********************************************************************
ADCIB00  EQU *
         ENTUP R2
         LA    R9,COMADDR         GET COMMUNICATION AREA ADDR
         EXTRACT (R9),FIELDS=COMM,MF=(E,EXTRACT)
*                                 EXTRACT THE COMMUNICATIONS AREA
         L     R9,COMADDR         GET ADDRESS OF THE AREA
         USING COM,R9             USE R9 AS BASE ADDRESS OF COMM AREA
         ICM   R7,15,COMCIBPT     GET CIB ADDRESS FROM COM AREA
         BZ    ADCIB50            NO CIB, START COMMAND NOT PRESENT
         BAL   R2,DLCIB00         START COMMAND PRESENT,FREE CIB
ADCIB50  QEDIT ORIGIN=COMCIBPT,CIBCTR=5   SET MODIFY LIMIT TO 5
ADCIB99  RETUP R2
         EJECT
**********************************************************************
*        CHECK THE COMMUNICATION ECB FOR COMMAND INPUT               *
*        DETERMINE WHETHER A STOP OR MODIFY HAS BEEN ENTERED         *
**********************************************************************
LKCMD00  EQU *
         ENTUP R2
         L     R9,COMADDR            GET ADDR OF CIB FROM COMMAREA
         USING COM,R9                ESTABLISH ADDRESSABILITY
         L     R1,COMECBPT           GET ADDR OF THE COMMUNICATION ECB
         TM    0(R1),B'01000000'     ECB POSTED ?
         BO    LKCMD10               YES, PROCESS MODIFY COMMAND
         B     LKCMD99               NO, RETURN
LKCMD10  EQU *
         ICM   R7,15,COMCIBPT        GET CIB ADDRESS FROM COM AREA
         USING CIB,R7                BASE CIB MAPPING
         CLI   CIBVERB,CIBMODFY      WAS IT A MODIFY?
         BNE   LKCMD80               NO, GO FREE CIB
         CLC   CIBDATA,=C'SHUTDOWN'  ALLOWED MODIFY CMD ENTERED ?
         BNE   LKCMD80               NO, GO FREE CIB
         BAL   R2,DLCIB00            FREE CIB
         MVI   CONFLAG,C'1'          SIGNAL COMMAND ENTERED
         B     LKCMD99               GOTO RETURN
LKCMD80  EQU *                       UNKNOWN COMMAND ENTERED
         BAL   R2,DLCIB00            FREE CIB
         WTO   MF=(E,WTOMSG1)        EXECUTE WTO
         CLI   CIBVERB,CIBSTOP       WAS IT A STOP ?
         BNE   LKCMD99               NO, RETURN
         QEDIT ORIGIN=COMCIBPT,CIBCTR=5   RESET MODIFY LIMIT TO 5
LKCMD99  RETUP R2
         EJECT
**********************************************************************
*        WRITE MQ/SERIES ERROR MESSAGE TO CONSOLE                    *
**********************************************************************
WTOMS00  EQU *
         ENTUP R2
```

```
         WTO   MF=(E,WTOMSG2)        EXECUTE WTO
WTOMS99  RETUP R2
         EJECT
**********************************************************************
*        SUB ROUTINES                                                *
*                                                                    *
**********************************************************************
**********************************************************************
*        USE QEDIT TO FREE THE CIB                                   *
*        QEDIT WILL ALSO CLEAR THE ECB                               *
**********************************************************************
DLCIB00  EQU *
         ENTUP R2
         QEDIT ORIGIN=COMCIBPT,BLOCK=(R7)   FREE THE CIB
DLCIB99  RETUP R2
         EJECT
**********************************************************************
*        DATA DEFINITION                                             *
*                                                                    *
**********************************************************************
         EJECT
**********************************************************************
*        LITERALPOOL                                                 *
*                                                                    *
**********************************************************************
         LTORG
         EJECT
**********************************************************************
*        DC - DEFINITIONS                                            *
*                                                                    *
**********************************************************************
         DS    0F
EXTRACT  EXTRACT MF=L              EXTRACT PARAMETER LIST
WTOMSG1  WTO   'INVALID COMMAND ENTERED |
               ,ROUTCDE=(2),DESC=(12),MF=L
WTOMSG2  WTO   'ERROR - MQ/SERIES BATCH TRIGGER MONITOR ABNORMAL ENDED
                CHECK JOBLOG FOR ERROR DESCRIPTION |
               ,ROUTCDE=(2),DESC=(12),MF=L
         END
```


Raimund Kleebaur
DB2/CICS/MQS Systems Programmer
Hugo Boss AG (Germany)

Client/server MQSeries with REXX

INTRODUCTION

This article provides a simple way for readers to familiarize themselves with MQSeries application programming and to try out the MQI using a REXX client/server application. The client portion is designed to work under Windows NT using IBM's object REXX package (or using OS/2 Warp 4). The MVS server portion runs under TSO/ISPF REXX (it could also run under TSO BATCH).

One of the many benefits of using REXX is that the same language can be used for both the client (PC) and server (mainframe) portion of the application. So a complete end-to-end client/server environment can be set up with knowledge of just one simple interpreted language. Changes can be made quickly, and various client/server scenarios can be replicated or modelled with ease.

Many MQSeries administrators (especially those with a mainframe background) are familiar with REXX, which is also one of the easiest programming languages to learn. IBM provides support packs to enable REXX MQSeries applications, and these need to be downloaded from IBM's Web site and installed before you proceed any further. The MVS part of the support pack does not require any special facilities and can be placed in your ordinary ISPLLIB 'ddname' library.

After installing the IBM support packs, you need to get the channels and queues defined on your PC (which I am assuming runs the MQSeries Queue manager, though it could also be a MQSeries client) to access the mainframe queue manager. Start the server REXX under ISPF, which causes it to issue a series of MQGETs waiting for requests to arrive. Run the PC REXX program to generate requests, and switch between your PC and host session to see the messages exchanged.

Having successfully configured MQSeries to permit this simple application to operate, you are now in a position to experiment with different MQI options and different ways of client/server programming without having to become an expert in 'C' or COBOL and so forth.

Changes to the REXX code can be made in seconds and the results tested without recompilation or the need for CICS etc.

Note the use of the continuation character, '➤', in the code below to indicate that one line of code maps to several lines of print.

THE PC-BASED CLIENT APPLICATION

```
/* REXX BASIC MQ REQUESTOR ON NT */

CALL RXFUNCADD 'SYSLOADFUNCS', 'REXXUTIL', 'SYSLOADFUNCS'
CALL SYSLOADFUNCS

SAY 'MQREXX STARTED'

QMGRL = 'MQM.MYQM'              /* LOCAL QUEUE MANAGER NAME */
QNAM2 = 'TEST.REQUEST'          /* QUEUE FOR OUTGOING REQ */
QNAM1 = 'TEST.REPLY'            /* QUEUE FOR IN REPLY   */

RET = RXFUNCADD('RXMQNINIT', 'RXMQN', 'RXMQNINIT')

RET = RXMQNINIT()
CALL CHECK 'INIT'

RET = RXMQNCONN(QMGRL)
CALL CHECK 'CONN'

RET = RXMQNOPEN(QNAM1, MQOO_INPUT_SHARED, 'QHAN1', 'MQOD1.')
CALL CHECK 'OPEN INPUT Q'
RET = RXMQNOPEN(QNAM2, MQOO_OUTPUT, 'QHAN2', 'MQOD2.')
CALL CHECK 'OPEN OUTPUT Q'

DO I = 1 TO 2
   DROP IPO.  IPD.
   IPO.OPT = MQPMO_NO_SYNCPOINT + MQPMO_FAIL_IF_QUIESCING
   IPD.MSG = MQMT_DATAGRAM
   IPD.PER = MQPER_NOT_PERSISTENT
   IPD.FORM = 'MQSTR'

   MSG.1 = 'THIS IS A TEST MESSAGE HERE' TIME()
   MSG.0 = LENGTH(MSG.1)                  /* THE MSG LEN */

   RET = RXMQNPUT(QHAN2, 'MSG.','IPD.','OMD.','IPO.','OMO.')
   RCP = WORD(RET,1)
   SAY 'MQREXX PUT ISSUED ***'

   IF RCP = 0 THEN
      DO
```

```
         DO J = 1 TO 10 UNTIL (RCG \= 2033)
            MSG.0 = 200               /* MAX MSG LENGTH*/
            MSG.1 = ''
            IGO.OPT = MQGMO_WAIT + MQGMO_SYNCPOINT
                      + MQGMO_FAIL_IF_QUIESCING,
                      + MQGMO_CONVERT
            IGO.WAIT = 30 * 1000      /* 10 SECONDS */
            SAY 'MQREXX ABOUT TO ISSUE GET WITH WAIT FOR MILLISECS'
            ➤  IGO.WAIT
            RET = RXMQNGET(QHAN1, 'MSG.','IGD.','OMD.','IGO.','OMO.')
            RCG = WORD(RET,1)
            SELECT
               WHEN (RCG = 0) THEN
                  SAY 'MQREXX INCOMING MSG IS' MSG.1
               WHEN (RCG = 2033) THEN
                  SAY 'MQREXX NO REPLY ON QUEUE'
               OTHERWISE
                  CALL CHECK 'GET'
            END
         END
      END
   ELSE
      CALL CHECK 'PUT'

CALL DUMP1

END

RET = RXMQNCLOSE(QHAN1, MQCO_NONE)
CALL CHECK 'CLOSE'
RET = RXMQNDISC()
CALL CHECK 'DISC'
RET = RXMQNTERM()
CALL CHECK 'TERM'

SAY 'MQREXX ENDED'

EXIT 0

CHECK:

ARG TYPE

LASTRC = WORD(RET,1)
IF LASTRC = 0 THEN
   RETURN

INTERPRET 'TEXT = RXMQV.RCMAP'LASTRC

SAY 'MQREXX CALL' TYPE  'FAILED, RETN   COMP   REAS   FUNC'
```



```
SAY 'MQREXX RET = ' RET
SAY 'MQREXX ERROR IS' TEXT

EXIT 4

RETURN

DUMP1:

DO J = 1 TO WORDS(OMO.ZLIST)
   K = WORD(OMO.ZLIST,J)
   IF K \= OMO.K THEN
      SAY 'MQREXX MSGOPT' K '=' OMO.K
END
SAY ' '

DO J = 1 TO WORDS(OMD.ZLIST)
   K = WORD(OMD.ZLIST,J)
   IF K \= OMD.K THEN
      SAY 'MQREXX DESC' K '=' OMD.K
END
SAY ' '

RETURN
```

THE MVS-BASED SERVER APPLICATION

```
/* REXX  BASIC MQ SERVER ON MVS */

QMGRL = 'MVS1'                /* LOCAL QUEUE MANAGER NAME */
QNAM1 = 'TEST.REQUEST'        /* QUEUE FOR INCOMING REQ   */
QNAM2 = 'TEST.REPLY'          /* QUEUE FOR REPLY (FIXED)  */
MSG.1 = ''

RET = RXMQV('INIT')
CALL CHECK 'INIT'
RET = RXMQV('CONN', QMGRL)
CALL CHECK 'CONN'

RET = RXMQV('OPEN', QNAM1, MQOO_INPUT_SHARED,  'QHAN1', 'MQOD1.')
CALL CHECK 'OPEN INPUT Q'
RET = RXMQV('OPEN', QNAM2, MQOO_OUTPUT, 'QHAN2', 'MQOD2.')
CALL CHECK 'OPEN OUTPUT Q'

DO I = 1 TO 2

   MSGL = MSG.1

   DO J = 1 TO 10 UNTIL (RCG \= 2033)

      MSG.0 = 4096              /* MAX MSG LENGTH*/
```

```
      MSG.1 = ''
      IGO.OPT = MQGMO_WAIT + MQGMO_SYNCPOINT + MQGMO_FAIL_IF_QUIESCING,
                + MQGMO_CONVERT

      IGO.WAIT = 20 * 1000      /* 20 SECONDS */

      ADDRESS ISPEXEC "CONTROL DISPLAY LOCK"
      STAT = 'MQREXX ABOUT TO ISSUE GET FOR MILLSECS' IGO.WAIT
      ADDRESS ISPEXEC "DISPLAY PANEL(MQREXX3A)"

      RET = RXMQV('GET', QHAN1, 'MSG.','IGD.','OGD.','IGO.','OGO.')
      RCG = WORD(RET,1)

      IF RCG = 0 THEN
         DO
            IPO.OPT = MQPMO_NO_SYNCPOINT + MQPMO_FAIL_IF_QUIESCING

            IPD.MSG = MQMT_DATAGRAM
            IPD.PER = MQPER_NOT_PERSISTENT
            IPD.FORM = 'MQSTR'

            MSG.1 = MSG.1 '**RETURNED**' TIME()
            MSG.0 = LENGTH(MSG.1)            /* THE MSG LEN */

            RET = RXMQV('PUT',QHAN2,'MSG.','IPD.','OPD.','IPO.','OPO.')
            RCP = WORD(RET,1)
            CALL CHECK 'PUT'
         END
      ELSE
         SELECT
            WHEN (RCG = 2033) THEN  /* NO MESSAGE ON QUEUE */
               NOP
            OTHERWISE
               CALL CHECK 'GET'
         END
   END

/* CALL DUMP1 */

END

RET = RXMQV('CLOSE', QHAN1, MQCO_NONE)
CALL CHECK 'CLOSE'
RET = RXMQV('DISC')
CALL CHECK 'DISC'
RET = RXMQV('TERM')
CALL CHECK 'TERM'

STAT = 'MQREXX ENDED'
ADDRESS ISPEXEC "DISPLAY PANEL(MQREXX3A)"
```



```
EXIT 0

CHECK:

ARG TYPE

LASTRC = WORD(RET,1)
IF LASTRC = 0 THEN
   RETURN

/* INTERPRET 'TEXT = RXMQV.RCMAP'LASTRC */

SAY 'MQ' TYPE 'CALL FAILED, RETN   COMP   REAS   FUNC'
SAY RET
SAY TEXT

EXIT 4

RETURN

DUMP1:

DO J = 1 TO WORDS(OGO.ZLIST)
   K = WORD(OGO.ZLIST,J)
   SAY 'OGO.'K '=' OGO.K
END
SAY ' '
DO J = 1 TO WORDS(OGD.ZLIST)
   K = WORD(OGD.ZLIST,J)
   SAY 'OGD.'K '=' OMD.K
END

RETURN
```

ISPF PANEL MQREXX3A

```
%------------------------- MQ REXX -------------------------
%COMMAND ===>_ZCMD                                          +
+
+&STAT
+
+READ  COUNT  + &I
+RETRY COUNT  + &J
+
+LAST MSG    + &MSGL
+
+LAST RETURN + &RET
+
+
)INIT
```

```
)PROC
)END
```

RUNMQSC DEFINITIONS FOR WINDOWS NT

```
*******************************************************
*    Queue manager definitions for MQM.MYQM
*
*    Define Channels & XMIT queues for access to MVS1
*    (assumes channel initiator running against
*     default channel initiation queue)
*******************************************************

DEFINE PROCESS(MVS1.P1) +
       APPLTYPE(DEF) +
       USERDATA(MYQM.MVS1.C1) REPLACE

DEFINE CHANNEL(MYQM.MVS1.C1) CHLTYPE(SDR) REPLACE +
       TRPTYPE(TCP) +
       XMITQ(MVS1) CONNAME(MVSIPNAME) DISCINT(600)

DEFINE CHANNEL(MVS1.MYQM.C1) CHLTYPE(RCVR) REPLACE +
       TRPTYPE(TCP)

DEFINE QLOCAL(MVS1) REPLACE USAGE(XMITQ) PROCESS(MVS1.P1) +
       TRIGGER TRIGTYPE(FIRST) +
       INITQ(SYSTEM.CHANNEL.INITQ)

****************************************************
*    Define queues for TEST application
****************************************************
DEFINE QALIAS(TEST.REQUEST) REPLACE +
       TARGQ(TEST.REQUEST.QR)

DEFINE QALIAS(TEST.REPLY)   REPLACE +
       TARGQ(TEST.REPLY.QL)

DEFINE QLOCAL (TEST.REPLY.QL) REPLACE

DEFINE QREMOTE (TEST.REQUEST.QR) REPLACE +
       RNAME (TEST.REQUEST) +
       RQMNAME(MVS1) XMITQ(MVS1)

****************************************************
*    End of file
****************************************************
```

MQ news

MQSoftware has announced that QPasa!, the company's MQSeries monitoring and management product, now supports IBM's Commerce Integrator, IBM's add-on product for the company's own Net.Commerce e-commerce suite. Earlier this year MQSoftware announced that QPasa! Version 2.0 (available in 1Q99) added full support for monitoring and managing IBM's MQSeries Integrator and MQSeries Workflow.

Pricing for QPasa! starts at US$20,000 for monitoring and managing up to 15 queue managers. The product is available now.

For further information contact:
MQSoftware Inc, 7575 Golden Valley Road, Suite 140, Minneapolis, MN 55427, USA
Tel: +1 612 546 9080
Fax: +1 612 546 9082
Web: http://www.mqsoftware.com

MQSoftware Europe Ltd, The Surrey Technology Centre, 40 Occam Road, Surrey Research Park, Guildford, Surrey GU2 5YH, UK
Tel: +44 1483 295400
Fax: +44 1483 573704

* * *

Landmark Systems has announced The Monitor for MQSeries Version 1.1, the latest version of the company's performance monitoring system for MQSeries across multiple platforms. Among the improvements is a rules-driven Automatic Dead Letter Queue Processor, which enables critical messages to be handled differently from non-critical ones.

Also new is better integration with The Monitor for CICS/ESA, allowing information and views from both mainframe and distributed applications to be presented together, and the MVS Queue Manager Security Management feature now displays MQSeries security switches and provides the ability to alter user IDs, time-out intervals, and refresh security.

The Monitor for MQSeries requires MVS/ESA 4.3 or higher and supports all Level 2 MQ Managers. Out now, prices weren't announced.

For further details contact:
Landmark Systems Corporation, 12700 Sunrise Valley Drive, Reston, VA 20191-5804, USA
Tel: +1 703 464 1300
Fax: +1 703 464 4918
Web: http://www.landmark.com

Landmark Systems, Eastlands Court Business Centre, St. Peters Road, Rugby, Warwickshire CV21 3QP, UK
Tel: +44 8700 744 755
Fax: +44 8700 755 766

* * *

IBM has announced that MQSeries is to be one of the products to benefit from its new Parallel Sysplex pricing scheme. The new Level D Pricing includes a reduced price structure 25% below the previously lowest price available.

For further details contact your local IBM representative.