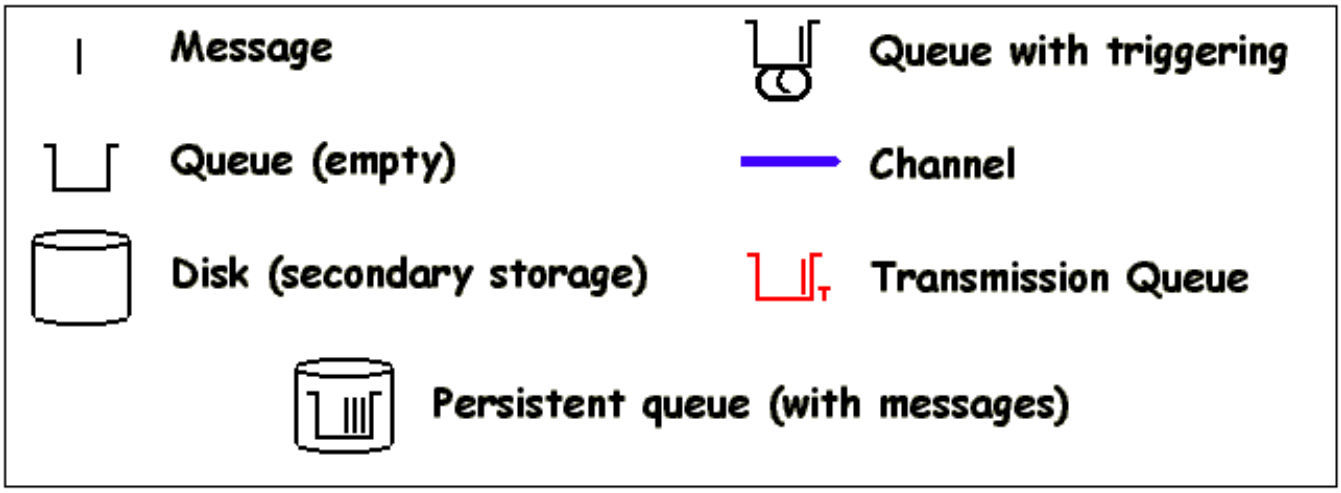
September 2006

MQ Use with ALCS

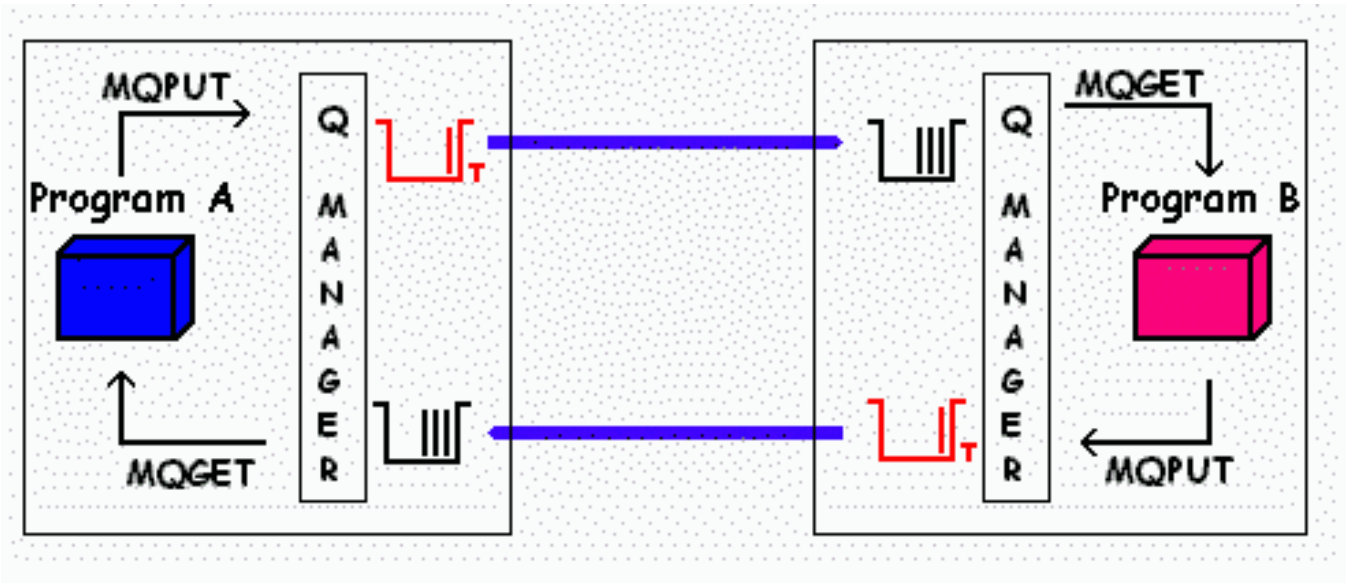
Introduction

This document describes the set up and use of Websphere MQ (MQ) with ALCS. It is aimed at being a quick start guide for ALCS systems programmers. It is **not** a substitute for a course in MQ, which IBM can provide.

ALCS utilises the services of 'Websphere MQ for z/OS'. This is prerequisite and must be active in the same system image as ALCS so that ALCS can connect to it and get MQ calls serviced.



MQ Basics



A message is any information that one application wishes to communicate to another.

A queue is a place to store messages until they can be processed.

A message channel is a one-way link between two queue managers for the transmission of messages.

A transmission queue is a local queue with a special role. Any message for a remote queue is placed by the queue manager on to a transmission queue.

MQ is a means of program to program communication using messages and queues. The communicating applications can be on the same system or distributed across a network.

MQ provides a common application programming interface, the MQI, that is consistent across all the supported platforms.

MQ can transfer data with assured delivery; messages don't get lost, even if there is a system failure, and just as important there is no duplicate delivery.

The communicating applications do not need to be active at the same time.

Benefits

In summary the benefits of MQ include:

- MQI is easy to use
- Some constraints of program-program relationships are removed
- Programs can be scheduled to make the best use of resources
- Fewer network sessions are needed
- Programs are less vulnerable to network failures
- Code is easier to move and reuse

Definitions

This section covers the definition of MQ in ALCS and the definition of MQ objects.

ALCS Support

ALCS monitor CSECT DXCMQI contains MQI support routines; it is provided as an object module for use. The source code is provided for reference only, the module is maintained as a object deck as you may not have all the macros to be able to assemble it. There is no need to link edit the ALCS monitor with MQ code. ALCS loads the MQ entry points dynamically at initialization, if MQI is supported by ALCS. The MQ load library must be available via STEPLIB or system LINKLST at ALCS execution time.

ALCS SCTGEN Definitions

The optional parameters on the SCTGEN generation macro that instruct ALCS to connect to MQ.

MQM=NO|(YES,CONNECT)|(YES,NOCONNECT)
Specifies if ALCS supports MQ and whether to connect at restart

MQMM=q_mgr_name
Message Queue Manager name (this is the local Websphere MQ for z/OS queue manager which will service the MQ requests for ALCS).

MQMI=(init_q_name|(init_q_name,system_state)
Initiation Queue name. Queue used by the Message Queue Manager to alert ALCS that a message has arrived on a triggered queue. See Message Driven Processing.

MQMQ=input_q_name|(input_q_name,system_state)
Input Queue name. ALCS can optionally open this queue when it connects to the Message Queue Manager. Messages sent to the input queue must be in PPMSG format (ROUTC) - ALCS will create an ECB and place the message text into a storage block on level 0. This method could be ideal for connecting ALCS to TPF or another ALCS, potentially replacing any LU6.1 links used for asynchronous processing.

Defining MQ Objects In z/OS

Queues and Processes are "MQ Objects" (as are Queue Managers and Channels) that need to be defined and named. Take care as the names are **case sensitive** so stick to a convention. The names can be upto 48 characters (except for channels which can have upto 20 characters). I suggest using only capitals with dots as separators so for example:

```
MY.IMPORTANT.QUEUE
MY.TRIGGER.PROCESS
```

could be typical names for MQ objects in this scheme.

Definitions of MQ objects are made by ‘Websphere MQ for z/OS’ commands, which can be issued from several places including:

1. The z/OS console (or equivalent, such as SDSF). When entering commands from the z/OS console, use the command prefix string (CPF) defined for your queue manager. When entering commands by any other means, the CPF must not be present.

   Note: Some commands, such as the DEFINE CHANNEL command for sender, server, and requester channels, might require too many mandatory parameters to be issued using SDSF. These commands have to be issued by another method.

2. The supplied batch utility (CSQUTIL) processing a list of commands in a sequential data set or member of a partitioned data set. Specify the destination queue manager with the TGTQMGR keyword; you also need to specify the queue manager to which you will connect via the EXEC PARM parameter of your JCL.
3. Much of the functionality of these commands is available in a user-friendly way from the MQ operations and control panels, described in the Websphere MQ for z/OS System Management Guide.

Changes made to the resource definitions of a queue manager using the commands (directly or indirectly) are preserved across restarts of the queue manager.

Defining a queue via the panels

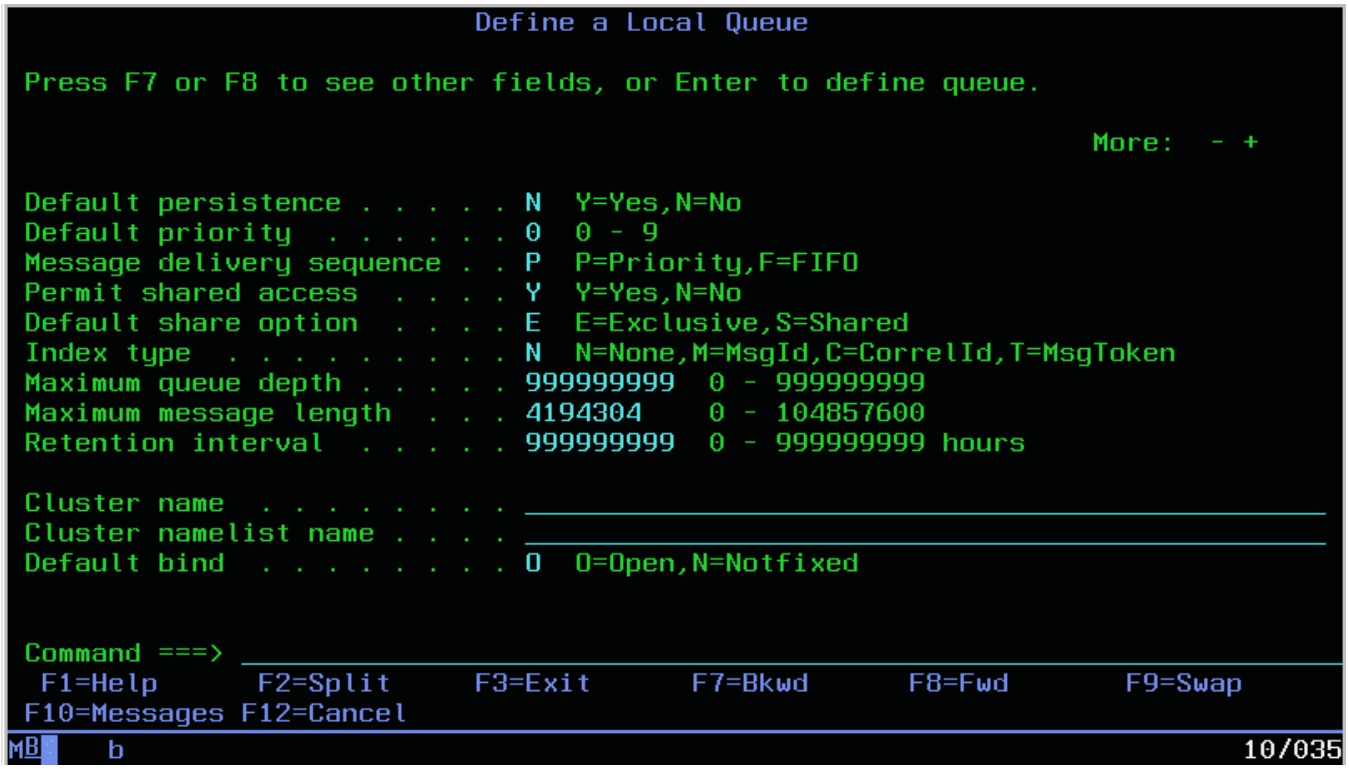
Here is an example of how to define a simple queue - one that is local to the MVS system and can be used for input and output. On the following panels the data entered is in pale blue (Cyan).

```
                    IBM MQSeries for OS/390 - Main Menu

Complete fields. Then press Enter.

 Action . . . . . 2      1. Display   5. Perform
                         2. Define    6. Start
                         3. Alter     7. Stop
                         4. Delete

 Object type  . . . . . QLOCAL        +
 Name . . . . . XAN.QUEUE.7
 Like . . . . . ____________________________________________________

 Connect to queue
 manager  . . . . . : CSQ1
 Target queue manager : CSQ1
 Response wait time . : 30   seconds

(C) Copyright IBM Corporation 1993,1999. All rights reserved.


Command ===> ____________________________________________________________
 F1=Help      F2=Split     F3=Exit      F4=Prompt    F6=QueueMgr  F9=Swap
F10=Messages F12=Cancel
```

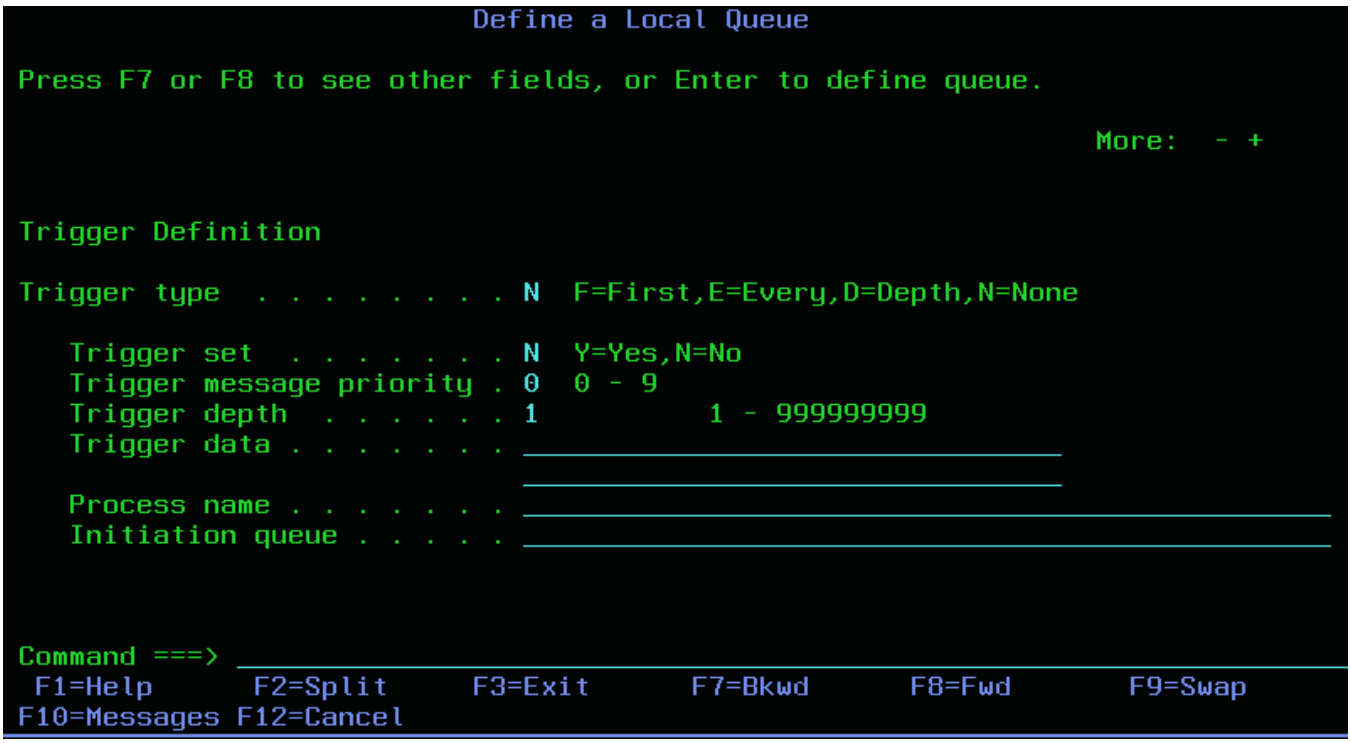
On pressing enter a continuation panel requires completing

```
                          Define a Local Queue

Complete fields, then press F8 for further fields, or Enter to define queue.

                                                              More:    +

Queue name  . . . . . . . . . XAN.QUEUE.7
Description . . . . . . . . . My new ALCS application queue___
                               ____________________________________

Put enabled . . . . . . . . . Y  Y=Yes,N=No
Get enabled . . . . . . . . . Y  Y=Yes,N=No
Usage . . . . . . . . . . . . N  N=Normal,X=XmitQ
Storage class . . . . . . . . DEFAULT









Command ===> ____________________________________________________________
 F1=Help      F2=Split     F3=Exit      F7=Bkwd      F8=Fwd       F9=Swap
F10=Messages F12=Cancel
```

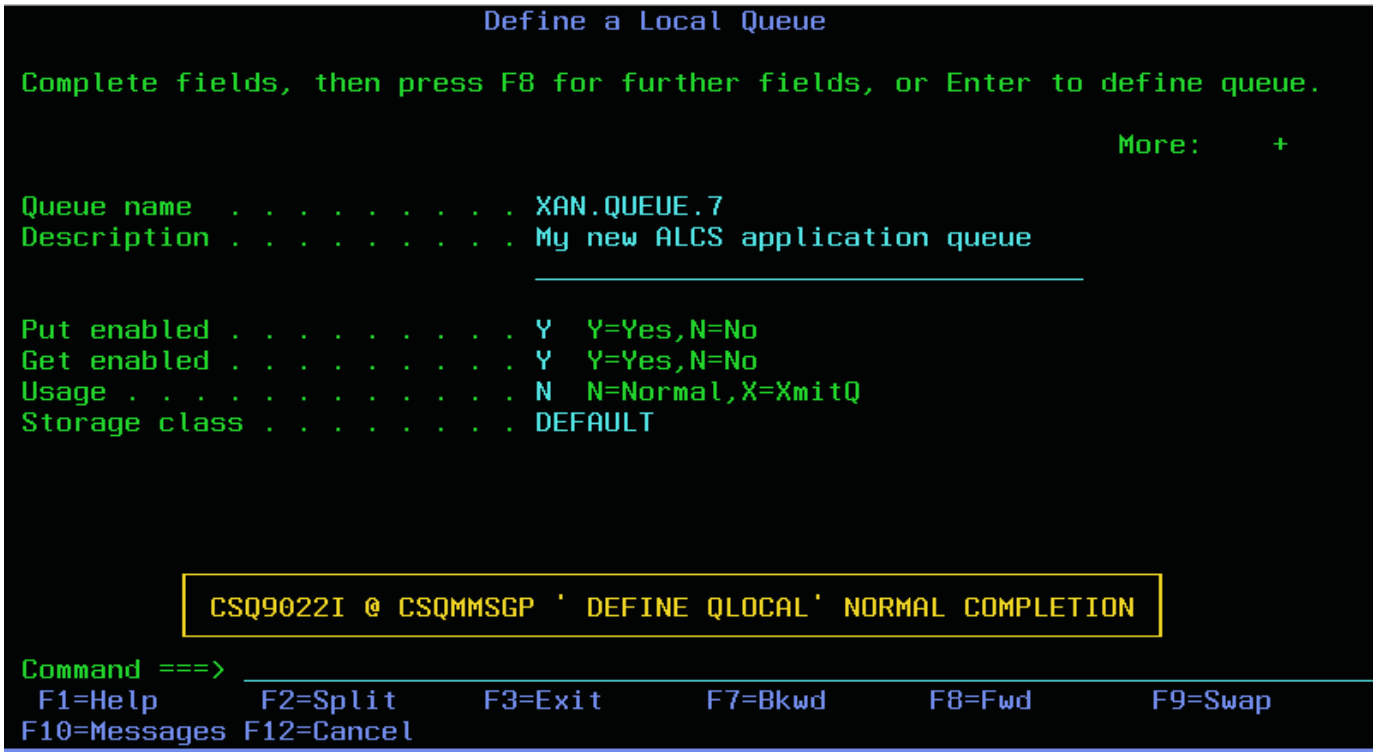
...and again...



...and again - for this queue we will not have triggering (Trigger type None)...



...finally we see the completion screen.



Defining a queue via the batch utility

Alternatively, a batch utility, CSQUTIL, could have been used, rather than the panels. An example of how to define the same queue as using the panels above is given below :

```
//MYJOB    JOB '0639222A,364161','M Hannaford',
//             CLASS=C,MSGCLASS=X,TIME=NOLIMIT,REGION=8M
//CSQUTIL  EXEC PGM=CSQUTIL,REGION=4096K,PARM='CSQ1'
//STEPLIB  DD DSN=SYS1.MQM.SCSQANLE,DISP=SHR
//         DD DSN=SYS1.MQM.SCSQAUTH,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
 COMMAND
/*
//CSQUCMD  DD *
 DELETE QL('XAN.QUEUE.7')
DEFINE -
 QLOCAL('XAN.QUEUE.7') -
 STGCLASS('DEFAULT') -
 DESCR('Re-definition of my queue in batch')
 PUT(ENABLED) -
 DEFPRTY(0) -
 DEFPSIST(NO) -
 NOTRIGGER -
 USAGE(NORMAL) -
 SHARE -
 DEFSOPT(EXCL) -
 MSGDLVSQ(PRIORITY) -
 TRIGTYPE(NONE) -
 NOHARDENBO -
 GET(ENABLED)
/*
//
```

Displaying the queue via system command

The Z/OS system command needs to be prefixed by the command prefix, the '@' symbol in this example:

```
 @DISPLAY QUEUE('XAN.QUEUE.7')
```

the output is much more verbose than the input, a lot of parameters take a default.

```
CSQM201I @ CSQMDMSG  DISPLAY QUEUE DETAILS 119
QUEUE(XAN.QUEUE.7)
TYPE(QLOCAL)
STGCLASS(DEFAULT)
CLUSTER( )
CLUSNL( )
DESCR(Re-definition of my queue in batch)
PUT(ENABLED)
DEFPRTY(0)
DEFPSIST(NO)
OPPROCS(0)
IPPROCS(0)
CURDEPTH(0)
MAXDEPTH(999999999)
PROCESS( )
NOTRIGGER
MAXMSGL(4194304)
BOTHRESH(0)
BOQNAME( )
INITQ( )
USAGE(NORMAL)
SHARE
DEFSOPT(EXCL)
MSGDLVSQ(PRIORITY)
RETINTVL(999999999)
TRIGTYPE(NONE)
TRIGDPTH(1)
TRIGMPRI(0)
TRIGDATA( )
DEFTYPE(PREDEFINED)
NOHARDENBO
CRDATE(2000-08-04)
CRTIME(14.10.32)
GET(ENABLED)
QDEPTHHI(80)
QDEPTHLO(40)
QDPMAXEV(ENABLED)
QDPHIEV(DISABLED)
QDPLOEV(DISABLED)
QSVCINT(999999999)
QSVCIEV(NONE)
INDXTYPE(NONE)
DEFBIND(OPEN)
ALTDATE(2000-08-04)
ALTTIME(14.10.32)
CSQMDMSG END QUEUE DETAILS
CSQ9022I @ CSQMDMSG ' DISPLAY QUEUE' NORMAL COMPLETION
```

Writing Applications

A first application

You can write your MQ application for ALCS in Assembler or a high level language. Here the samples are in Assembler or C. You may find that using C makes sense even if you have not used it before for ALCS applications as MQI provides structures in C (as well as in Assembler). These structures, groups of fields, are used to supply input to, and get output from, the calls. The MQI also provides a large set of named constants to help you supply options in the parameters of the calls. Also, as we have seen default values are set within the MQI calls.

Each object is identified by an object descriptor (MQOD), which you use when you write your application. These objects must be defined to the queue manager before you can work with them (see prior section Defining MQ Objects).

A MQ message is created by using a MQI call to put the message on a queue. As input to the call, you supply some control information in a message descriptor (MQMD) along with the message (data) that you want to send to another program.

For a program to work with an MQ object, the program must have a unique identifier by which it knows that object. This identifier is called an object handle. The handle is returned by the MQOPEN call when the program opens the object to work with it. Programs pass the object handle as an input parameter when they use subsequent MQPUT, MQGET, MQINQ, MQSET, or MQCLOSE calls. (Note: There is a connection handle specified on each call, a unique identifier by which the program knows the queue manager. For ALCS programs it is zero - actually ALCS takes care of it transparently to the application programs. So the applications do not issue the MQCONN/MQCONNX call).

Refer to The MQ Application Programming Guide for comprehensive information. For a quick start look at the sample applications and then refer to the above manual for more coding advice.

Assembling and Linking applications that contain MQ calls

To Assemble program MQA1 the MQ macro library must be added to the SYSLIB concatenation. The module then needs to be linked with the ALCS ECB stub program CDSN. This example creates load module member MQA1:

```
//MQA1      EXEC PGM=ASMA90,REGION=4M,
//     PARM=(OBJ,NODECK,'LC(55)',RENT,NORLD,'XREF(SHORT)',
//     'USING(MAP,WARN(15))','COMPAT(CASE,SYSLIST)')
//SYSLIB    DD DSN=DXC.V2R3M1.DXCMAC2,DISP=SHR     ALCS V231
//          DD DSN=DXC.V2R3M1.DXCMAC1,DISP=SHR     ALCS V231
//          DD DSN=DXC.V2R3M1.DXCMAC3,DISP=SHR     ALCS V231
//          DD DSN=DXC.V2R3M1.DXCMAC4,DISP=SHR     ALCS V231
//          DD DSN=SYS1.MQM.SCSQMACS,DISP=SHR      MQ
//          DD DSN=SYS1.MODGEN,DISP=SHR            Z/OS
//          DD DSN=SYS1.MACLIB,DISP=SHR            Z/OS
//SYSUT1    DD DSN=&&SYSUT1,UNIT=VIO,SPACE=(3000,(3000,300))
//SYSLIN    DD DSNAME=&&MQA1,UNIT=VIO,SPACE=(CYL,(1,1,0)),
//             DCB=(BLKSIZE=400),DISP=(,PASS)
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD DSN=MY.PROGRAM.SOURCE(MQA1),DISP=SHR
/*
//MQLINK    EXEC PGM=HEWL,COND=(4,LT,MQA1),
//             PARM=(XREF,LIST,LET,CALL,RENT)
//SYSPRINT  DD SYSOUT=*
//SYSUT1    DD DSN=&&SYSUT1,UNIT=VIO,SPACE=(1700,(600,100))
//SYSOBJ    DD DSN=DXC.V2R3M1.DXCMOD1,DISP=SHR
//SYSLMOD   DD DSN=MY.PROGRAM.LOAD(MQA1),DISP=SHR
//SYSLIN    DD DSN=&&MQA1,DISP=(OLD,DELETE)
//      DD DDNAME=SYSIN
//SYSIN DD *
 MODE RMODE(ANY),AMODE(31)
```

```
 INCLUDE SYSOBJ(CDSN)
 NAME MQA1(R)
/*
//
```

For C, the SYSLIB concatentation of the compile needs to include the MQM C Header library

```
//SYSLIB   ......
//         DD DSNAME=SYS1.MQM.SCSQC370,DISP=SHR
```

Sample Application

Sample application in Assembler

MQM0 is a simplified version of a program written by ALCS development to test the ALCS to MQ interface. It is a very good example to start with. It is clearly written and laid out. It allows you to put/get a user specified number of internally generated messages to the queue which you also specify. The simplest way to invoke the program is through ZDRIV but you will need to refer to the ROC for output messages. Alternatively if you set up a communication definition for a new input application program :

```
          COMDEF LDTYPE=ALCSAPPL,NAME=MQMQ,PROG=MQMA,                    -
                SYSSTATE=IDLE,ISTATUS=ACTIVE
```

then you can use the ZROUT MQMQ command to enter commands directly to the program. For example adding 5 messages and removing 2 from the queue using this program.

```
zrout mqmq
DXC8523I CMD M  03.30.09 ROUT  Terminal routing updated
PUT XAN.QUEUE.7 5
Program MQM0 finished
GET XAN.QUEUE.7 2
Program MQM0 -- Test application message number 0005+

Program MQM0 -- Test application message number 0004+

Program MQM0 finished
```

Sample application in C

MQMH is an application that allows the terminal user to add (MQPUT) messages to a queue and remove (MQGET) messages from a queue. It also will show how many messages there are on a queue (MQINQ) which will also show, information on the first ten messages on the queue. This is done by using an MQGET call with browse which allows an application to non-destructively retrieve a message. This final part of the application uses code that is 'ported' from the MVS sample application CSQ4TCH1-3 as described in the MQ Application Programming Guide. I have given the application a default queue of XAN.QUEUE.7 which you may wish to change. If you do not then this default can be overridden on each entry (as in the example run below to XAN.QUEUE.3).

The sample is driven by the terminal user routing to a new application. So again a new COMDEF will be needed :

```
          COMDEF LDTYPE=ALCSAPPL,NAME=MQMH,PROG=MQMH,                    -
                SYSSTATE=IDLE,ISTATUS=ACTIVE,SFORM=(,MIXED)
```

note that this definition defines the application as a mixed case application. This is important as message queue names are case sensitive.

Here is an example run of the program :

```
zrout mqmh
DXC8523I CMD M  17.12.48 ROUT  Terminal routing updated
MQINQ XAN.QUEUE.3
Queue set to  : XAN.QUEUE.3
Current depth : 000000000

No messages on queue.   CompCode: 2   Reason: 2033

MQPUT XAN.QUEUE.3
Put this test message on the queue
Queue set to  : XAN.QUEUE.3
Message length 34 placed.

MQINQ XAN.QUEUE.3
Queue set to  : XAN.QUEUE.3
Current depth : 000000001

Msg  Put Date  Put Time User     Put      Start of Msg
No  MM/DD/YYYY HH:MM:SS Id       Appl     Text
01  09/13/2000 13:55:21 IALMH    IALMHRUN Put this test message on the queue

MQGET XAN.QUEUE.3
Queue set to  : XAN.QUEUE.3
Destructive MQGET. Message length 34, text :
Put this test message on the queue
```

The MQPUT entry uses the newline key. You can type several lines of text with newline. Terminate the last line with the Enter key.

Try it yourself

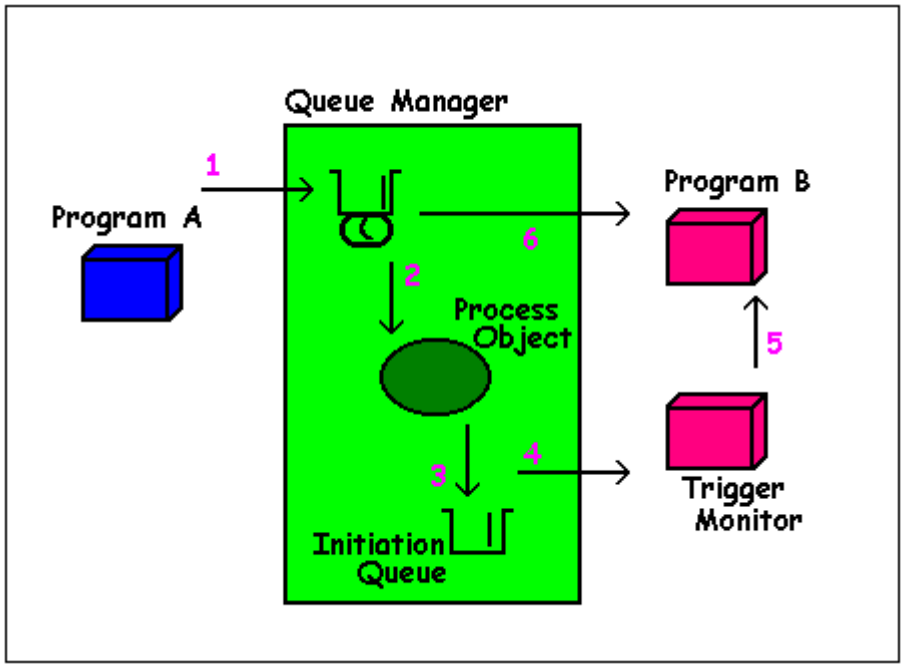
This application shows it is possible to code and port a MQ application written in C - it may be worth extending this application yourself to see how you find it compared to writing in assembler.
Personally, I found that because this application has no need to make use of the ECB it is relatively easy to write in C on ALCS. I have found that when trying to write an application in C that heavily uses the ALCS api and ECB then this ease is somewhat lost.

Message Driven Processing

Message Driven Processing is the name used to describe a processing model were complex business transactions can be broken down into discrete functional modules which communicate with each other by means of messages. The modules can execute on different systems and be scheduled at different times. To assist in this MQ has the concept of triggering.

Triggering

Triggering is an enhancement to the implementation of time-independent processing. The arrival of a message can, given the right conditions being met, result in the application that serves the application queue to be started.



The actions depicted are:

1. Program A puts a message on an application queue which is enabled for triggering.

2. If the conditions for triggering are met, a trigger event occurs, and the queue manager examines the process object referenced by the application queue.

3. The process object identifies the application to be started, in this case Program B. The queue manager creates a trigger message whose fields contain information copied from certain attributes of the process object and the application queue. The queue manager puts the message on the initiation queue.

4. A long running program called a trigger monitor gets the trigger message, examines the contents, and...

5. ...starts Program B, passing the entire trigger message as a parameter.

6. Program B opens the application queue and gets messages from it.

Type of Trigger

There are several parameters of the MQ command DEFINE QLOCAL which control triggering, for example **TRIGGER|NOTRIGGER** indicates whether triggering is active or not active respectively. One parameter influences the program design of the application that serves the application queue, it is **TRIGTYPE** :

TRIGTYPE(FIRST)

If enabled a trigger event occurs whenever the queue changes from being empty to having one message on it.

TRIGTYPE(DEPTH)

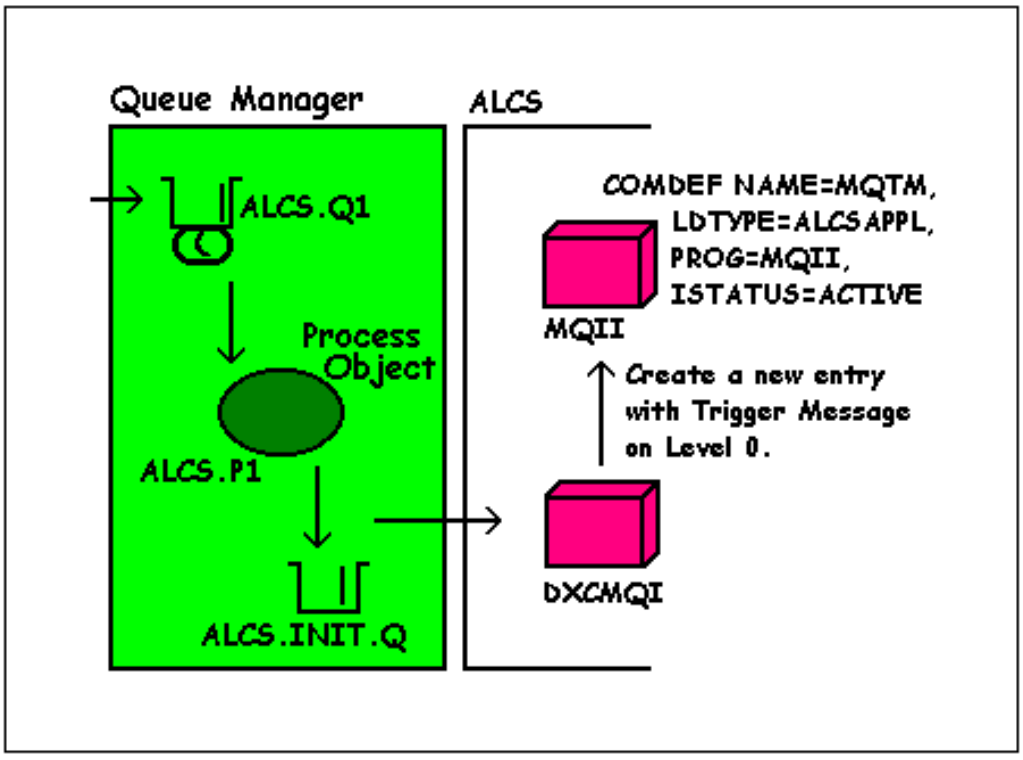
A trigger event occurs when the number of messages on the queue reaches the value indicated by the TRIGDPTH parameter. When a trigger message is created the queue manager disables triggering by setting the queue to **NOTRIGGER**. It is then the responsibility of the application to re-enable triggering by use of the MQSET call.

TRIGTYPE(EVERY)

A trigger event occurs whenever a message is placed on the queue.

Triggering in ALCS

You could have an ALCS application that uses MQGET with wait to implement message driven processing but then you have to manage a long running entry. Let ALCS do the hard work. ALCS has it's own implementation of triggering using the ALCS initiation queue.



Defining a COMDEF for the trigger monitor application

```
COMDEF LDTYPE=ALCSAPPL,NAME=MQTM,PROG=MQII,                    -
      SYSSTATE=IDLE,ISTATUS=ACTIVE
```

The application queue must be associated with an ALCS application that is to serve that queue. To do this, you must:

1. Create an MQ initiation queue for your application queue and specify its name in the **InitiationQName** attribute of the application queue.
2. Create an ALCS application to process trigger messages that arrive on the initiation queue.

3. Create an MQM Series process definition object and specify the CRN of your ALCS application in its **Applicid** attribute.
4. Name this process definition object in the **ProcessName** attribute of the application queue.
5. Set the trigger conditions that you require in the attributes of the application queue.

Whenever a trigger message arrives on the initiation queue that ALCS has opened, ALCS creates a new entry and copies the trigger message into a storage block attached to the ECB on level 0. Then it passes control to the ALCS application that is named in the trigger message.

An MQ queue manager can own more than one initiation queue, and each one is associated with one or more application queues. ALCS can only have one initiation queue open at a time. ALCS closes the initiation queue when it disconnects from the queue manager.

A Simple Example of ALCS Triggering

Firstly we need to define the queue which will have triggering enabled, the process and the initiation queue. Using the same names as in the previous section, this could be :

```
//MYJOB2   JOB '0639222A,364161','M Hannaford',
//             CLASS=C,MSGCLASS=X,TIME=NOLIMIT,REGION=8M
//CSQUTIL  EXEC PGM=CSQUTIL,REGION=4096K,PARM='CSQ1'
//STEPLIB  DD DSN=SYS1.MQM.SCSQANLE,DISP=SHR
//         DD DSN=SYS1.MQM.SCSQAUTH,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
 COMMAND
/*
//CSQUCMD  DD *
 DELETE PROCESS('ALCS.P1')
 DEFINE PROCESS('ALCS.P1') -
  DESCR('ALCS trigger process') -
  APPLTYPE(MVS) -
  APPLICID(MQTM) -
  USERDATA('ALCS TRIGGER MESSAGE HAS ARRIVED')
 DELETE QLOCAL('ALCS.INIT.Q')
 DEFINE QLOCAL('ALCS.INIT.Q') -
  DESCR('ALCS initiation queue') -
  MAXDEPTH(100)
 DELETE QL('ALCS.Q1')
 DEFINE QLOCAL('ALCS.Q1') -
  STGCLASS('DEFAULT') -
  DESCR('Queue with triggering') -
  PUT(ENABLED) -
  GET(ENABLED) -
  USAGE(NORMAL) -
  SHARE -
  DEFSOPT(EXCL) -
  TRIGGER -
  TRIGTYPE(EVERY) -
  INITQ('ALCS.INIT.Q') -
  PROCESS('ALCS.P1')
/*
//
```

The sample MQII application is very simple. Rather than automate the invocation of the application to go and retrieve the message from the application queue, it is instead reporting the fact there is a message on the queue to the RO CRAS. Then the operator (in this case you), can start the process to retrieve the message. It is not what you would do in normally but it demonstrates the principle and removes
some of the mystery of how it works, so hopefully provides you with the best understanding of what needs to be defined and what is actually happening.

Here is an example run. First using PRC to disconnect / reconnect to the queue manager with our new initiation queue. Also, check the trigger application is loaded and active.

```
zcmqi disc
DXC8627I CMD M  18.14.33 CMQI
Now disconnected from MQSeries for MVS/ESA
Queue manager CSQ1
zcmqi conn,initq=ALCS.INIT.Q,INPUTQ=
DXC8626I CMD M  18.15.21 CMQI
Now connected to MQSeries for MVS/ESA
Queue manager    CSQ1
Initiation queue ALCS.INIT.Q
Input queue      not defined

zdcom n=mqtm
...
zdpgm mqii
...
```

Then a message is put on the application queue using the MQMH application.

```
zrout mqmh
DXC8523I CMD M  18.17.07 ROUT  Terminal routing updated
MQPUT ALCS.Q1
Hello there, can you hear me!
Queue set to  : ALCS.Q1
Message length 29 placed.
```

and if all has been set up then on RO Cras you will see the message

```
MQII  Trigger received for: ALCS.Q1
```

Then look on the ALCS.Q1 application queue the message will be seen.

Appendix A : Sample Assembler source

```
         BEGIN NAME=MQM0,VERSION=00,AMODE=31,XCL=NONE,SHR=NONE
         SPACE 1
         TRANV NAME=MQM0,ENTRY=MQM0
         TRANV NAME=MQMA,ENTRY=MQMA
         SPACE 1
*=====================================================================*
*        ALCS APPLICATION - TEST MQM MVS/ESA MESSAGE QUEUEING         *
*=====================================================================*
*                                                                     *
*        LOAD PROGRAM BY "ZPCTL L,S|T,MQM0"                           *
*                                                                     *
*        FORMAT IS "ZDRIV MQM0,GET,queue_name,N|WAIT|SIGNAL           *
*               OR "ZDRIV MQM0,PUT,queue_name,N                       *
*                                                                     *
*        MQM0 = DRIVER PROGRAM TO PUT/GET MESSAGES TO/FROM QUEUE      *
*                                                                     *
*        MQMA = INPUT EDIT PROGRAM FOR ALCS APPLICATION MQMA          *
*               (ALTERNATIVE ENTRY POINT TO ZDRIV)                    *
*                                                                     *
*---------------------------------------------------------------------*
         EJECT ,
******** MACROS AND EQUATES
         SPACE 1
         RC0PL REG=R00             ROUTING CONTROL PARAMETER LIST
         CM1CM REG=R01             INPUT MESSAGE
         CO0PR REG=R06             PARSED INPUT MESSAGE
         CO0IC REG=R14             COMIC RETURNED DATA AREA
         SPACE 1
         CMQA   ,                  MQM/ESA API EQUATES
         SPACE 1
         CMQGMOA DSECT=YES,LIST=YES MQM/ESA GET MESSAGE OPTIONS
         CMQMDA  DSECT=YES,LIST=YES MQM/ESA MSG DESCRIPTOR STRUCTURE
         CMQODA  DSECT=YES,LIST=YES MQM/ESA OBJECT DESCRIPTOR STRUCTURE
         CMQPMOA DSECT=YES,LIST=YES MQM/ESA PUT MSG OPTIONS STRUCTURE
         SPACE 1
         RSECT ,                   RESTORE PROGRAM CSECT BASE
         SPACE 1
         USING MQGMO,R02           GET MESSAGE OPTIONS
         USING MQMD,R03            MESSAGE DESCRIPTOR
         USING MQOD,R04            OBJECT DESCRIPTOR
         USING MQPMO,R05           PUT MESSAGE OPTIONS
         SPACE 1
COMPCODE EQU   EBW000               COMPLETION CODE
REASON   EQU   EBW004               REASON CODE
MQHCONN  EQU   EBW008               CONNECTION HANDLE
MQHOBJ   EQU   EBW012               OBJECT HANDLE
OPTIONS  EQU   EBW016               OPTIONS
USEREVCB EQU   EBW020               SIGNAL RETRY EVCB
DATALEN  EQU   EBW024               LENGTH OF DATA
BUFFERL  EQU   EBW028               LENGTH OF DATA AREA
BUFFER   EQU   EBW032               START OF DATA AREA
         SPACE 1
******** EBSW01 VALUES
         SPACE 1
MANYMSG  EQU   0
GETWAIT  EQU   1
GETSIG   EQU   2
         EJECT ,
*=====================================================================*
*        MQMA -- ENTRY POINT                                          *
```

```
*====================================================================*
         SPACE 1
******** CHECK IF SCREEN FORMATTING NEEDED
         SPACE 1
MQMA     DC    0H'0'               APPLICATION INPUT ENTRY POINT
         L     R01,CE1CR0          LOAD INPUT MESSAGE BASE
         LH    R14,CM1CCT          LOAD MESSAGE LENGTH
         LA    R15,CM1CRI-2(R14)   GET ADDRESS OF EOM CHARACTER - 1
         CLI   1(R15),#EOU         SCREEN FORMAT REQUESTED
         BE    MQMA0010            YES - BRANCH TO FORMAT
         SPACE 1
         CLI   0(R15),C'>'         SIMULATED PF12
         BE    MQMA0010            YES - BRANCH TO FORMAT
         SPACE 1
         CLI   CM1TXI,X'FF'        PF12 AID BYTE
         BNE   MQM0                NO - BYPASS FORMAT
         SPACE 1
MQMA0010 DC    0H'0'
         ENTNC CFMT                FORMAT SCREEN
         EJECT ,
*====================================================================*
*        MQM0 -- ENTRY POINT                                         *
*====================================================================*
         SPACE 1
MQM0     DC    0H'0'               ZDRIV ENTRY POINT
         DEFRC ,                   WAIT FOR TRACE
         SPACE 1
         SLIMC ECBLIFE=NONE,GFS=NONE   EXTEND ECB LIFE
         SPACE 1
******** REFORMAT INPUT MESSAGE FOR CFMP
         SPACE 1
         L     R01,CE1CR0          LOAD INPUT MESSAGE BASE
         LH    R07,CM1CCT          LOAD CCT
         CH    R07,=H'3'           ENOUGH PARAMETERS
         BNH   MQM0E020            NO - BRANCH
         SPACE 1
         SH    R07,=Y(3)           DECREMENT FOR CRI
         LA    R05,CM1TXI-1(R07)   POINT TO END OF MESSAGE
         LA    R06,CM1TXI-1+6(R07) POINT TO NEW END OF MESSAGE
         SPACE 1
MQM00005 DC    0H'0'
         MVC   0(1,R06),0(R05)     MOVE MESSAGE TEXT
         BCTR  R05,0               DECREMENT MOVE-FROM POINTER
         BCTR  R06,0               DECREMENT MOVE-TO POINTER
         BCT   R07,MQM00005        MOVE ALL MESSAGE TEXT
         SPACE 1
         MVC   CM1TXI,=CL6'ZZZZZ ' MOVE IN DUMMY COMMAND NAME
         LH    R07,CM1CCT          LOAD CCT
         LA    R07,6(,R07)         INCREMENT FOR DUMMY COMMAND NAME
         STH   R07,CM1CCT          MOVE IN NEW CCT
         SPACE 1
         ENTRC CFMP                PARSE INPUT MESSAGE
         SPACE 1
         FLIPC D2,D1               MOVE PARSED MESSAGE TO LEVEL D1
         L     R06,CE1CR1          LOAD PARSED MESSAGE BASE
         SPACE 1
         GETCC LEVEL=D2,SIZE=L3    GET A NEW STORAGE BLOCK
         GETCC LEVEL=D3,SIZE=L3    GET A NEW STORAGE BLOCK
         GETCC LEVEL=D4,SIZE=L3    GET A NEW STORAGE BLOCK
         GETCC LEVEL=D5,SIZE=L3    GET A NEW STORAGE BLOCK
         SPACE 1
         CLC   PRSW1,=H'0'         ANY ERRORS REPORTED
         BNE   MQM0E010            YES - BRANCH
```

```
         SPACE 1
         CLC   PRNUM,=H'3'         RIGHT NUMBER OF PARAMETERS
         BNE   MQM0E020            NO - BRANCH
         SPACE 1
         L     R04,=A(TRTAB1)      LOAD ADDRESS OF TRANSLATE TABLE
         CLC   =C'GET',PRVAL       IS IT GET
         BE    MQM00010            YES - BRANCH
         SPACE 1
         CLC   =C'PUT',PRVAL       IS IT PUT
         BE    MQM00020            YES - BRANCH
         SPACE 1
         B     MQM0E030            OTHERWISE TAKE ERROR BRANCH
         SPACE 1
MQM00010 DC    0H'0'
         MVI   EBSW01,GETWAIT
         CLC   =C'WAIT',PRVAL+L'PRENT+L'PRENT IS IT GET,,WAIT
         BE    MQM00030            YES - BRANCH
         SPACE 1
         MVI   EBSW01,GETSIG
         CLC   =C'SIGNAL',PRVAL+L'PRENT+L'PRENT IS IT GET,,SIGNAL
         BE    MQM00030            YES - BRANCH
         SPACE 1
MQM00020 DC    0H'0'
         MVI   EBSW01,MANYMSG
         XC    EBX000(8),EBX000    CLEAR WORK AREA
         MVC   EBX000(5),=C'00000' SET NUMBER OF MESSAGES TO ZERO
         SPACE 1
         CLI   PRVLN+L'PRENT+L'PRENT,X'4' IS NNN > 9999
         BH    MQM0E040            YES - TAKE ERROR BRANCH
         BE    MQM00024            BRANCH IF NNN IS > 999
         SPACE 1
         CLI   PRVLN+L'PRENT+L'PRENT,X'3' IS NNN > 99
         BE    MQM00023            YES - BRANCH
         SPACE 1
         CLI   PRVLN+L'PRENT+L'PRENT,X'2' IS NNN > 9
         BE    MQM00022            YES - BRANCH
         SPACE 1
         TRT   PRVAL+L'PRENT+L'PRENT(1),0(R04) IS NNN VALID
         BNZ   MQM0E040            NO - TAKE ERROR BRANCH
         SPACE 1
         MVC   EBX004(1),PRVAL+L'PRENT+L'PRENT
         B     MQM00029
         SPACE 1
MQM00022 DC    0H'0'
         TRT   PRVAL+L'PRENT+L'PRENT(2),0(R04) IS NNN VALID
         BNZ   MQM0E040            NO - TAKE ERROR BRANCH
         SPACE 1
         MVC   EBX003(2),PRVAL+L'PRENT+L'PRENT
         B     MQM00029
         SPACE 1
MQM00023 DC    0H'0'
         TRT   PRVAL+L'PRENT+L'PRENT(3),0(R04) IS IT GET|PUT,,NNN
         BNZ   MQM0E040            NO - TAKE ERROR BRANCH
         SPACE 1
         MVC   EBX002(3),PRVAL+L'PRENT+L'PRENT
         SPACE 1
         B     MQM00029
         SPACE 1
MQM00024 DC    0H'0'
         TRT   PRVAL+L'PRENT+L'PRENT(4),0(R04) IS IT GET|PUT,,NNN
         BNZ   MQM0E040            NO - TAKE ERROR BRANCH
         SPACE 1
         MVC   EBX001(4),PRVAL+L'PRENT+L'PRENT
```

```
         SPACE 1
MQM00029 DC    0H'0'
         PACK  EBX005(3),EBX000(5) PACK NNN
         XC    EBX000(5),EBX000    CLEAR UNUSED BYTES
         CVB   R07,EBX000          CONVERT TO BINARY
         ST    R07,EBX000          SAVE NUMBER OF MESSAGES FOR GET|PUT
         EJECT ,
*--------------------------------------------------------------------*
*        OPEN QUEUE                                                  *
*--------------------------------------------------------------------*
         SPACE 1
MQM00030 DC    0H'0'
         L     R04,CE1CR4          LOAD MQOD BASE
         LR    R07,R05             SAVE R05 OVER MVCL
         LA    R05,MQOD_LENGTH     LOAD MOVE TO LENGTH
         LA    R14,MQOD01          LOAD MOVE FROM ADDRESS
         LR    R15,R05             LOAD MOVE FROM LENGTH
         MVCL  R04,R14             SET UP OBJECT DESCRITPOR
         LR    R05,R07             RESTORE R05
         L     R04,CE1CR4          LOAD MQOD BASE AS MVCL HAS DESTROYED
         SPACE 1
         SR    R07,R07             CLEAR REGISTER
         IC    R07,PRVLN+L'PRENT   INSERT LENGTH OF QUEUE NAME
         BCTR  R07,0               DECREMENT FOR EXECUTE
         MVC   MQOD_OBJECTNAME(0),PRVAL+L'PRENT (EXECUTED INSTRUCTION)
         EX    R07,*-6             MOVE IN QUEUE NAME
         SPACE 1
         XC    EBW000(28),EBW000   CLEAR ECB WORK AREA
         L     R00,=A(MQOO_INPUT_SHARED) OPEN QUEUE FOR INPUT
         ST    R00,OPTIONS         SET UP OPTIONS
         SPACE 1
         CLC   =C'GET',PRVAL       IS IT GET
         BE    MQM00040            YES - BRANCH
         SPACE 1
         L     R00,=A(MQOO_OUTPUT) OPEN QUEUE FOR OUTPUT
         ST    R00,OPTIONS         SET UP OPTIONS
         SPACE 1
MQM00040 DC    0H'0'
         CALL  MQOPEN,                                              -
               (MQHCONN,MQOD,OPTIONS,MQHOBJ,COMPCODE,REASON),       -
               VL,MF=(E,EBX008)
         SPACE 1
         CLC   =A(MQCC_OK),COMPCODE  WAS IT SUCCESSFUL
         BE    MQM00050              YES - BRANCH
         SPACE 1
         B     MQM0E060            OTHERWISE TAKE ERROR BRANCH
         SPACE 1
MQM00050 DC    0H'0'
         XC    COMPCODE(4),COMPCODE  CLEAR COMPLETION CODE
         XC    REASON(4),REASON    CLEAR REASON CODE
         SPACE 1
         LA    R07,1               SET NUMBER OF MESSAGES = 1
         CLI   EBSW01,MANYMSG      IS IT GET|PUT,,NNN
         BNE   MQM00055            NO - BRANCH
         SPACE 1
         L     R07,EBX000          LOAD NUMBER OF MESSAGES FOR GET|PUT
         SPACE 1
MQM00055 DC    0H'0'
         CLC   =C'GET',PRVAL       IS IT GET
         BNE   MQM00080            NO - BRANCH FOR PUT
         EJECT ,
*--------------------------------------------------------------------*
*        GET MESSAGE(S) FROM QUEUE                                   *
```

```
*-------------------------------------------------------------------*
         SPACE 1
MQM00060 DC    0H'0'
         CLI   EBSW01,GETWAIT      IS IT GET,,WAIT
         BE    MQM00073            YES - BRANCH
         SPACE 1
         CLI   EBSW01,GETSIG       IS IT GET,,SIGNAL
         BE    MQM00076            YES - BRANCH
         SPACE 1
         XC    BUFFER(72),BUFFER   CLEAR GET MESSAGE BUFFER
         MVC   BUFFERL(4),=A(72)   LENGTH OF BUFFER AREA FOR MESSAGE
         SPACE 1
         L     R03,CE1CR3          LOAD MQMD BASE
         MVC   MQMD(256),MQMD01    SET UP MESSAGE DESCRIPTOR
         MVC   MQMD+256(MQMD_LENGTH-256),MQMD01+256
         SPACE 1
         L     R02,CE1CR2          LOAD MQGMO BASE
         MVC   MQGMO(MQGMO_LENGTH),MQGMO01 SET UP GET MESSAGE OPTIONS
         SPACE 1
         CALL  MQGET,                                              -
               (MQHCONN,MQHOBJ,MQMD,MQGMO,BUFFERL,BUFFER,DATALEN,  -
               COMPCODE,REASON),                                   -
               VL,MF=(E,EBX008)
         SPACE 1
         CLC   =A(MQCC_OK),COMPCODE  WAS IT SUCCESSFUL
         BE    MQM00070              YES - BRANCH
         SPACE 1
         CLC   =A(MQRC_TRUNCATED_MSG_ACCEPTED),REASON
         BE    MQM00070
         SPACE 1
         B     MQM0E070            OTHERWISE TAKE ERROR BRANCH
         EJECT ,
MQM00070 DC    0H'0'
         L     R14,DATALEN         GET LENGTH OF MESSAGE
         LTR   R14,R14             IS THERE ANY DATA
         BZ    MQM00072            NO - BRANCH
         SPACE 1
         C     R14,BUFFERL         IS IT LONGER THAN BUFFER
         BNH   MQM00071            NO - BRANCH
         SPACE 1
         L     R14,BUFFERL         GET LENGTH OF MESSAGE
         SPACE 1
MQM00071 DC    0H'0'
         LA    R01,BUFFER          LOAD ADDRESS OF MESSAGE TEXT
         LR    R02,R14             LOAD LENGTH OF MESSAGE TEXT
         BAS   R03,MQM0MSG         SEND MESSAGE
         SPACE 1
MQM00072 DC    0H'0'
         BCT   R07,MQM00060        LOOP FOR ALL MESSAGES
         SPACE 1
         B     MQM00100            GO TO CLOSE QUEUE
         EJECT ,
*-------------------------------------------------------------------*
*        GET WITH WAIT                                              *
*-------------------------------------------------------------------*
         SPACE 1
MQM00073 DC    0H'0'
         XC    BUFFER(72),BUFFER   CLEAR GET MESSAGE BUFFER
         MVC   BUFFERL(4),=A(72)   LENGTH OF BUFFER AREA FOR MESSAGE
         SPACE 1
         L     R03,CE1CR3          LOAD MQMD BASE
         MVC   MQMD(256),MQMD01    SET UP MESSAGE DESCRIPTOR
         MVC   MQMD+256(MQMD_LENGTH-256),MQMD01+256
```

```
         SPACE 1
         L     R02,CE1CR2          LOAD MQGMO BASE
         MVC   MQGMO(MQGMO_LENGTH),MQGMO02 SET UP GET MESSAGE OPTIONS
         SPACE 1
         L     R00,=A(MQWI_UNLIMITED)  UNLIMITED WAIT
         ST    R00,MQGMO_WAITINTERVAL  SET WAIT INTERVAL
         SPACE 1
         CALL  MQGET,                                                  -
               (MQHCONN,MQHOBJ,MQMD,MQGMO,BUFFERL,BUFFER,DATALEN,      -
               COMPCODE,REASON),                                       -
               VL,MF=(E,EBX008)
         SPACE 1
         CLC   =A(MQCC_OK),COMPCODE  WAS IT SUCCESSFUL
         BE    MQM00074              YES - BRANCH
         SPACE 1
         CLC   =A(MQRC_TRUNCATED_MSG_ACCEPTED),REASON
         BE    MQM00074
         SPACE 1
         B     MQM0E074            OTHERWISE TAKE ERROR BRANCH
         SPACE 1
MQM00074 DC    0H'0'
         L     R14,DATALEN         GET LENGTH OF MESSAGE
         LTR   R14,R14             IS THERE ANY DATA
         BZ    MQM00100            NO - BRANCH
         SPACE 1
         C     R14,BUFFERL         IS IT LONGER THAN BUFFER
         BNH   MQM00075            NO - BRANCH
         SPACE 1
         L     R14,BUFFERL         GET LENGTH OF MESSAGE
         SPACE 1
MQM00075 DC    0H'0'
         LA    R01,BUFFER          LOAD ADDRESS OF MESSAGE TEXT
         LR    R02,R14             LOAD LENGTH OF MESSAGE TEXT
         BAS   R03,MQM0MSG         SEND MESSAGE
         SPACE 1
         B     MQM00100            GO TO CLOSE QUEUE
         EJECT ,
*--------------------------------------------------------------------*
*        GET WITH SIGNAL                                             *
*--------------------------------------------------------------------*
         SPACE 1
MQM00076 DC    0H'0'
         XC    BUFFER(72),BUFFER   CLEAR GET MESSAGE BUFFER
         MVC   BUFFERL(4),=A(72)   LENGTH OF BUFFER AREA FOR MESSAGE
         SPACE 1
         L     R03,CE1CR3          LOAD MQMD BASE
         MVC   MQMD(256),MQMD01    SET UP MESSAGE DESCRIPTOR
         MVC   MQMD+256(MQMD_LENGTH-256),MQMD01+256
         SPACE 1
         L     R02,CE1CR2          LOAD MQGMO BASE
         MVC   MQGMO(MQGMO_LENGTH),MQGMO03 SET UP GET MESSAGE OPTIONS
         SPACE 1
         L     R00,=A(MQWI_UNLIMITED)  UNLIMITED WAIT
         ST    R00,MQGMO_WAITINTERVAL  SET WAIT INTERVAL
         SPACE 1
         LA    R00,USEREVCB          POINT TO SIGNAL RETRY EVCB
         ST    R00,MQGMO_SIGNAL1     SET POINTER TO SIGNAL RETRY EVCB
         SPACE 1
         CALL  MQGET,                                                  -
               (MQHCONN,MQHOBJ,MQMD,MQGMO,BUFFERL,BUFFER,DATALEN,      -
               COMPCODE,REASON),                                       -
               VL,MF=(E,EBX008)
         SPACE 1
```

```
         CLC   =A(MQCC_OK),COMPCODE  WAS IT SUCCESSFUL
         BE    MQM00077              YES - BRANCH
         SPACE 1
         CLC   =A(MQRC_TRUNCATED_MSG_ACCEPTED),REASON
         BE    MQM00077
         SPACE 1
         CLC   =A(MQRC_SIGNAL_REQUEST_ACCEPTED),REASON
         BE    MQM00079
         SPACE 1
         B     MQM0E078            OTHERWISE TAKE ERROR BRANCH
         SPACE 1
MQM00077 DC    0H'0'
         L     R14,DATALEN         GET LENGTH OF MESSAGE
         LTR   R14,R14             IS THERE ANY DATA
         BZ    MQM00100            NO - BRANCH
         SPACE 1
         C     R14,BUFFERL         IS IT LONGER THAN BUFFER
         BNH   MQM00078            NO - BRANCH
         SPACE 1
         L     R14,BUFFERL         GET LENGTH OF MESSAGE
         SPACE 1
MQM00078 DC    0H'0'
         LA    R01,BUFFER          LOAD ADDRESS OF MESSAGE TEXT
         LR    R02,R14             LOAD LENGTH OF MESSAGE TEXT
         BAS   R03,MQM0MSG         SEND MESSAGE
         SPACE 1
         B     MQM00100            GO TO CLOSE QUEUE
         SPACE 1
MQM00079 DC    0H'0'
         CALL  MQAWAIT
         SPACE 1
         B     MQM00076            GO TO RE-ISSUE MQGET
         EJECT ,
*-------------------------------------------------------------------*
*        PUT MESSAGE(S) ON QUEUE                                    *
*-------------------------------------------------------------------*
         SPACE 1
MQM00080 DC    0H'0'
         L     R03,CE1CR3          LOAD MQMD BASE
         MVC   MQMD(256),MQMD01    SET UP MESSAGE DESCRIPTOR
         MVC   MQMD+256(MQMD_LENGTH-256),MQMD01+256
         SPACE 1
         L     R05,CE1CR5          LOAD MQPMO BASE
         MVC   MQPMO(MQPMO_LENGTH),MQPMO01 SET UP PUT MESSAGE OPTIONS
         SPACE 1
******** PUT APPLICATION MESSAGE (ON APPLICATION QUEUE)
         SPACE 1
MQM00084 DC    0H'0'
         XC    BUFFER(72),BUFFER           CLEAR MESSAGE AREA
         MVC   BUFFER(MSG01S),MSG01        SET UP OUTPUT MESSAGE
         CVD   R07,EBX000          CONVERT MESSAGE NUMBER TO DECIMAL
         UNPK  EBX000(5),EBX005(3) CONVERT TO ZONED DECIMAL
         MVC   BUFFER+(MSG01N-MSG01)(4),EBX001 MOVE INTO MESSAGE
         OI    BUFFER+(MSG01N-MSG01)+3,X'F0'   SET ON ZONE BITS
         SPACE 1
         CALL  MQPUT,                                              -
               (MQHCONN,MQHOBJ,MQMD,MQPMO,MSG01L,BUFFER,           -
               COMPCODE,REASON),                                   -
               VL,MF=(E,EBX008)
         SPACE 1
MQM00086 DC    0H'0'
         CLC   =A(MQCC_OK),COMPCODE  WAS IT SUCCESSFUL
         BE    MQM00090              YES - BRANCH
```

```
         SPACE 1
         B     MQM0E080            OTHERWISE TAKE ERROR BRANCH
         SPACE 1
MQM00090 DC    0H'0'
         BCT   R07,MQM00080        LOOP FOR ALL MESSAGES
         SPACE 1
         B     MQM00100            GO TO CLOSE QUEUE
         EJECT ,
*------------------------------------------------------------------*
*        CLOSE QUEUE                                               *
*------------------------------------------------------------------*
         SPACE 1
MQM00100 DC    0H'0'
         XC    COMPCODE(4),COMPCODE  CLEAR COMPLETION CODE
         XC    REASON(4),REASON    CLEAR REASON CODE
         SPACE 1
         LA    R00,MQCO_NONE
         ST    R00,OPTIONS         SET UP OPTIONS
         SPACE 1
         CALL  MQCLOSE,                                           -
               (MQHCONN,MQHOBJ,OPTIONS,COMPCODE,REASON),          -
               VL,MF=(E,EBX008)
         SPACE 1
         CLC   =A(MQCC_OK),COMPCODE  WAS IT SUCCESSFUL
         BE    MQM00110              YES - BRANCH
         SPACE 1
         B     MQM0E090            OTHERWISE TAKE ERROR BRANCH
         SPACE 1
******** FINISHED
         SPACE 1
MQM00110 DC    0H'0'
         L     R01,=A(MSG03)       LOAD MESSAGE ADDRESS
         LA    R02,MSG03L          LOAD MESSAGE LENGTH
         BAS   R03,MQM0MSG         SEND MESSAGE
         SPACE 1
         BACKC ,                   RETURN OR EXIT
         SPACE 1
         DROP  R01                 DROP CM1CM BASE
         DROP  R02                 DROP MQGMO BASE
         DROP  R03                 DROP MQMD  BASE
         DROP  R04                 DROP MQOD  BASE
         DROP  R05                 DROP MQPMO BASE
         DROP  R06                 DROP CO0PR BASE
         DROP  R14                 DROP CO0IC BASE
         EJECT ,
*------------------------------------------------------------------*
*        ERROR ROUTINES                                            *
*------------------------------------------------------------------*
         SPACE 1
MQM0E010 DC    0H'0'
         L     R01,=A(MQM0E01M)
         B     MQM0DMP1
         SPACE 1
MQM0E020 DC    0H'0'
         L     R01,=A(MQM0E02M)
         B     MQM0DMP1
         SPACE 1
MQM0E030 DC    0H'0'
         L     R01,=A(MQM0E03M)
         B     MQM0DMP1
         SPACE 1
MQM0E040 DC    0H'0'
         L     R01,=A(MQM0E04M)
```

```
         B     MQM0DMP1
         SPACE 1
MQM0E050 DC    0H'0'
         L     R01,=A(MQM0E05M)
         B     MQM0DMP1
         SPACE 1
MQM0E060 DC    0H'0'
         LA    R01,MQM0OPEN
         B     MQM0DMP
         SPACE 1
MQM0E070 DC    0H'0'
         LA    R01,MQM0GET
         B     MQM0DMP
         SPACE 1
MQM0E074 DC    0H'0'
         LA    R01,MQM0GETW
         B     MQM0DMP
         SPACE 1
MQM0E078 DC    0H'0'
         LA    R01,MQM0GETS
         B     MQM0DMP
         SPACE 1
MQM0E080 DC    0H'0'
         LA    R01,MQM0PUT
         B     MQM0DMP
         SPACE 1
MQM0E090 DC    0H'0'
         LA    R01,MQM0CLOS
         B     MQM0DMP
         EJECT ,
MQM0DMP  DC    0H'0'
         L     R02,=A(DUMPMSG)     POINT TO DUMP MESSAGE
         MVC   EBX016(DUMPMSGL),0(R02)                       DUMP MSG
         MVC   EBX016+(DUMPMSGT-DUMPMSG)(L'DUMPMSGT),0(R01) CALL TYPE
         SPACE 1
         ICM   R00,B'1111',COMPCODE  INSERT COMPLETION CODE
         CVD   R00,EBX008          CONVERT TO PACKED DECIMAL
         UNPK  EBX008(4),EBX012(4) CONVERT TO ZONED DECIMAL
         OI    EBX011,X'F0'        TURN ON ZONE BITS
         MVC   EBX016+(DUMPMSGR-DUMPMSG)(L'DUMPMSGR),EBX008 COMP CODE
         SPACE 1
         ICM   R00,B'1111',REASON  INSERT REASON CODE
         CVD   R00,EBX008          CONVERT TO PACKED DECIMAL
         UNPK  EBX008(4),EBX012(4) CONVERT TO ZONED DECIMAL
         OI    EBX011,X'F0'        TURN ON ZONE BITS
         MVC   EBX016+(DUMPMSGS-DUMPMSG)(L'DUMPMSGS),EBX008 REASON CODE
         SPACE 1
         LA    R01,EBX016          POINT TO DUMP MESSAGE
         SPACE 1
MQM0DMP1 DC    0H'0'
         SR    R02,R02             CLEAR REGISTER
         IC    R02,0(R01)          INSERT MESSAGE LENGTH
         LA    R01,1(R01)          POINT TO MESSAGE TEXT
         SPACE 1
         BAS   R03,MQM0MSG         SEND MESSAGE
         SPACE 1
         EXITC ,                   EXIT
         EJECT ,
*------------------------------------------------------------------*
*        SUBROUTINE TO BUILD AND SEND MESSAGE                      *
*------------------------------------------------------------------*
*        ON ENTRY, R01 -> MESSAGE TEXT                             *
*                  R02 =  MESSAGE LENGTH                           *
```

```
*                 R03 -> RETURN POINT                                 *
*-------------------------------------------------------------------*
         SPACE 1
         CM1CM REG=R01             INPUT MESSAGE
         CO0IC REG=R14             COMIC RETURNED DATA AREA
         SPACE 1
MQM0MSG  DC    0H'0'
         RELCC LEVEL=D0,TYPE=COND  RELEASE ANY ATTACHED BLOCK
         GETCC LEVEL=D0,SIZE=L1    GET BLOCK FOR OUTPUT MESSAGE
         LR    R04,R01             LOAD ADDRESS OF MESSAGE TEXT
         L     R01,CE1CR0          LOAD OUTPUT MESSAGE BASE
         SPACE 1
         BCTR  R02,0               DECREMENT LENGTH FOR EXECUTE
         MVC   CM1TXT(0),0(R04)    (EXECUTED INSTRUCTION)
         EX    R02,*-6             MOVE IN MESSAGE TEXT
         LA    R02,CM1TXT+1(R02)   POINT PAST MESSAGE TEXT
         MVC   CM1CRI,EBROUT       MOVE IN DESTINATION CRI
         SPACE 1
         COMIC CRI=EBROUT,         GET COMMS DATA                     -
               DATA=SYS,                                              -
               AREA=(0,ICELEN)
         SPACE 1
         TM    ICESTPP,L'ICESTPP   IS IT A PRINTER
         BO    MQM0MSG1            YES - BRANCH
         SPACE 1
         MVC   CM1CMW(2),=AL1(#WEW,#CAR) MOVE IN CONTROL CHARACTERS
         MVC   0(3,R02),=AL1(#CAR,#SOM,#EOM) INSERT ENDING SEQUENCE
         LA    R04,CM1CRI-3        POINT TO START OF MESSAGE
         SR    R02,R04             CALCULATE CHARACTER COUNT
         STH   R02,CM1CCT          SAVE CHARACTER COUNT
         SPACE 1
         SENDC D0,A                SEND MESSAGE TO DISPLAY
         SPACE 1
         B     MQM0MSG2            CONTINUE
         SPACE 1
MQM0MSG1 DC    0H'0'
         MVC   CM1CMW(2),=AL1(#NOP,#CAR) MOVE IN CONTROL CHARS
         MVI   0(R02),#EOM         INSERT ENDING SEQUENCE
         LA    R04,CM1CRI-1        POINT TO START OF MESSAGE
         SR    R02,R04             CALCULATE CHARACTER COUNT
         STH   R02,CM1CCT          SAVE CHARACTER COUNT
         SPACE 1
         SENDC D0,L                SEND MESSAGE TO PRINTER
         SPACE 1
MQM0MSG2 DC    0H'0'
         BR    R03                 RETURN TO CALLER
         SPACE 1
         DROP  R01                 DROP CM1CM BASE
         DROP  R14                 DROP CO0IC BASE
         EJECT ,
*-------------------------------------------------------------------*
*        PROGRAM CONSTANTS                                          *
*-------------------------------------------------------------------*
         SPACE 1
******** BUILD GET MESSAGE OPTIONS
         SPACE 1
MQGMO01  CMQGMOA LIST=YES,                                           -
               OPTIONS=MQGMO_ACCEPT_TRUNCATED_MSG+MQGMO_NO_SYNCPOINT
         SPACE 1
MQGMO02  CMQGMOA LIST=YES,                                           -
               OPTIONS=MQGMO_ACCEPT_TRUNCATED_MSG+MQGMO_NO_SYNCPOINT+MQ-
               GMO_WAIT
         SPACE 1
```

```
MQGMO03  CMQGMOA LIST=YES,                                              -
               OPTIONS=MQGMO_ACCEPT_TRUNCATED_MSG+MQGMO_NO_SYNCPOINT+MQ-
               GMO_SET_SIGNAL
         SPACE 1
******** BUILD MESSAGE DESCRIPTOR
         SPACE 1
MQMD01   CMQMDA LIST=YES
         SPACE 1
******** BUILD OBJECT DESCRIPTOR
         SPACE 1
MQOD01   CMQODA LIST=YES,                                               -
               OBJECTTYPE=MQOT_Q
         SPACE 1
******** BUILD PUT MESSAGE OPTIONS
         SPACE 1
MQPMO01  CMQPMOA LIST=YES,OPTIONS=MQPMO_NO_SYNCPOINT
         SPACE 1
******** APPLICATION TEST MESSAGE
         SPACE 1
MSG01L   DC    A(MSG01S)
MSG01    DC    C'Program MQM0 -- Test application message number '
MSG01N   DC    CL4' '
         DC    AL1(#EOM)
MSG01S   EQU   *-MSG01
         SPACE 1
MQM0OPEN DC    CL5'OPEN'
MQM0GET  DC    CL5'GET'
MQM0GETW DC    CL5'GET/W'
MQM0GETS DC    CL5'GET/S'
MQM0PUT  DC    CL5'PUT'
MQM0CLOS DC    CL5'CLOSE'
         EJECT ,
******** LITERAL POOL
         SPACE 1
         LTORG ,
         EJECT ,
*--------------------------------------------------------------------*
*        INDIRECTLY ADDRESSED CONSTANTS                              *
*--------------------------------------------------------------------*
         SPACE 1
******** MQM0 ENDING MESSAGE
         SPACE 1
MSG03    DC    C'Program MQM0 finished'
MSG03L   EQU   *-MSG03
         SPACE 1
******** ERROR MESSAGES
         SPACE 1
DUMPMSG  DC    AL1(DUMPMSGL)
DUMPMSGI DC    C'MQM0 -- '
DUMPMSGT DC    CL5' '
         DC    C' -- COMPCODE '
DUMPMSGR DC    CL4' '
         DC    C' -- REASON '
DUMPMSGS DC    CL4' '
DUMPMSGL EQU   *-DUMPMSGI
         SPACE 1
******** ERROR MESSAGES
         SPACE 1
MQM0E01M DC    AL1(L'MQM0E01T)
MQM0E01T DC    C'MQM0 -- Errors reported by CFMP'
         SPACE 1
MQM0E02M DC    AL1(MQM0E02L)
MQM0E02T DC    C'MQM0 -- Format is "PUT queue_name nnnn"'
```

```
         DC    AL1(#CAR)
         DC    C'               or "GET queue_name nnnn|WAIT|SIGNAL"'
MQM0E02L EQU   *-MQM0E02T
         SPACE 1
MQM0E03M DC    AL1(L'MQM0E03T)
MQM0E03T DC    C'MQM0 -- Parameter 1 not GET|PUT'
         SPACE 1
MQM0E04M DC    AL1(L'MQM0E04T)
MQM0E04T DC    C'MQM0 -- Parameter 3 not nnnn'
         SPACE 1
MQM0E05M DC    AL1(L'MQM0E05T)
MQM0E05T DC    C'MQM0 -- Parameter 3 not numeric 1 to 9999'
         SPACE 1
******** TRANSLATE AND TEST TABLE
         SPACE 1
TRTAB1   DC    256X'FF'
         ORG   TRTAB1+C'0'
         DC    10X'00'
         ORG   ,
         SPACE 1
         FINIS ,
         SPACE 1
         END   ,
```

Appendix B : Sample C source

```
/* C language MQ API Exerciser
*
* Note the trigraph sequence ??( represents open square bracket and
* ??) represents close square bracket. These replacements are done
* in the first phase of preprocessing by the compilation.
*/

#include <tpfeq.h>
#include <tpfapi.h>
#include <tpfio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>

static char usage??(??) = {"\n input is case sensitive:\
\nMQINQ Q.NAME  -  Query number of messages on queue\
\nMQGET Q.NAME  -  Get message from queue\
\nMQPUT Q.NAME (newline)\nMessage text \
in mixed case..  -  Put message on queue"};
#define MQ_MSG_LENGTH 2048
#define MSG_LENGTH 79
#define SPACE " "
#define NEWLINE "\n"

/*****************************************************************/
/*                     Global Variables                          */
/*****************************************************************/
MQCHAR48 qName = "XAN.QUEUE.7";           /* queue name default  */
MQHCONN  HConn;                           /* qmgr connection hdl */
MQHOBJ   HObj;                            /* queue object handle */
MQBYTE24 MsgId, CorrelId;
static MQBYTE   verb??(5??);              /* action verb         */
char   numMsgs??(9 + 1??);
char   message??(MSG_LENGTH??);
char   scrMsg??(MQ_MSG_LENGTH??);

/*****************************************************************/
/*                  Function prototypes                          */
/*****************************************************************/
MQLONG openQ( MQHCONN HConn, MQCHAR48 qName, MQHOBJ *openHObj );
MQLONG closeQ( MQHCONN HConn, MQHOBJ *closeHObj );
MQLONG currentQDepth( MQHCONN HConn, MQHOBJ HObj, char* qDepth );
MQLONG createMsgList( MQHCONN HConn, MQHOBJ HObj, char* list);
MQLONG inquireQ( MQHCONN HConn, MQCHAR48 qName, MQHOBJ *openHObj );
MQLONG buildPut( char* putText );
MQLONG putQ( MQHCONN HConn, MQCHAR48 qName,
             MQBYTE *Buffer, MQLONG BufferLength);
MQLONG getQ( MQHCONN HConn, MQCHAR48 qName);
void   errorMessage( char *msgStr, MQLONG CompCode, MQLONG Reason );
void   printMessage( char *msgStr );

/*****************************************************************/
/*                           Main                                */
/*****************************************************************/
void MQMH()
{
   static char line1 ??(MSG_LENGTH??);
   char *ptr1, *ptrqName, *ptrverb;
                                                 /* prolog code     */
```

```
   gets(line1);                                  /* Any input?      */
   ptr1=&line1??(0??);                           /* No, clear & exit*/
   if (*ptr1 == 0) entdc("CFMT",NULL);           /* Does not return */

/*****************************************************************/
/* Test if MQ entry. If not show usage and leave                 */
/*****************************************************************/
   if ( !( (*ptr1) == 'M' && *(ptr1+1) =='Q' ) ) {
       printf("Non MQ entry; use ZROUT to change application");
       puts(usage);
       exit(0);
   }

/*****************************************************************/
/* Determine action requested and Queue Name                     */
/*****************************************************************/
   ptrverb=&verb??(0??);                    /* Isolate the action */
   strncpy(ptrverb,ptr1,5);                 /* verb. Move pointer */
   ptr1 = ptr1 + 6;                         /* to queue name.     */
   ptrqName=&qName??(0??);                  /*                    */
   if (*ptr1 == 0) {                        /* If not present     */
   printf("Queue defaults: %s", qName);     /* show default       */
   } else {                                 /*                    */
   strcpy( ptrqName, ptr1 );                /* If present show    */
   printf("Queue set to  : %s", qName);     /* new name set       */
   }                                        /*                    */

/*****************************************************************/
/* Conditional processing for entry type                         */
/*****************************************************************/
   if (strcmp(verb,"MQPUT") == 0) {
      if (MQRC_NONE == buildPut( scrMsg )) {
         if(MQRC_NONE == putQ( HConn, qName,
                               scrMsg, strlen(scrMsg)) ) {
            printf("\nMessage length %u placed.", strlen(scrMsg));
         } else {
            printf("\nMessage not placed on queue.");
         }
      } else {
      printf("\nNo message text found on subsequent line");
      }
   } else if (strcmp(verb,"MQINQ") == 0) {
      if( (MQRC_NONE == openQ( HConn, qName, &HObj )) ) {
         currentQDepth( HConn, HObj, numMsgs );
         printf("\nCurrent depth : %s\n", numMsgs);
         createMsgList( HConn, HObj, scrMsg );
         puts(scrMsg);
         closeQ( HConn, &HObj );
      }
   } else if (strcmp(verb,"MQGET") == 0) {
          getQ( HConn, qName );
   } else { printf("\nMQ verb not recognised.");
            puts(usage);
   }

/*****************************************************************/
/* Exit here                                                     */
/*****************************************************************/

   exit(0);
} /*end MQMH*/
/*****************************************************************/
```

```
/******************************************************************/
/*    This function inquires on the current queue depth for the   */
/*    specified queue and returns the depth if successful.        */
/******************************************************************/
MQLONG currentQDepth( MQHCONN HConn, MQHOBJ HObj, char* qDepth )
{
MQLONG Selectors = MQIA_CURRENT_Q_DEPTH;
MQLONG IntAttrs;
MQCHAR CharAttrs;
MQLONG SelectorCount  = 1;
MQLONG IntAttrCount   = 1;
MQLONG CharAttrLength = 0;
MQLONG CompCode, Reason;

/*                                                                */
/*    Call MQINQ with variables set to inquire the queue depth    */
/*                                                                */
MQINQ( HConn,
HObj,
SelectorCount,
&Selectors,
IntAttrCount,
&IntAttrs,
CharAttrLength,
&CharAttrs,
&CompCode,
&Reason );

/*                                                                */
/*      A failure will cause an error message to be displayed.    */
/*                                                                */
if( (MQCC_OK == CompCode) )                  /* Print the returned  */
     sprintf( qDepth, "%9.9ld", IntAttrs );  /* value to a string   */
else {
  errorMessage( "Error finding queue depth.", CompCode, Reason );
  IntAttrs = -1;
  }
return IntAttrs;                             /* Return depth        */
} /*end currentQDepth*/
/******************************************************************/


/******************************************************************/
/*   This function closes the specified queue.                    */
/*   The object handle passed in is used in the call to MQCLOSE and  */
/*   is returned to the call function.                            */
/*   A failure will cause an error message to be displayed.       */
/******************************************************************/
MQLONG closeQ( MQHCONN HConn, MQHOBJ *closeHObj )
{
MQLONG CompCode, Reason;

MQCLOSE( HConn,
closeHObj,
MQRC_NONE,
&CompCode,
&Reason );

if( (MQCC_OK != CompCode) )
errorMessage( "Failed when closing queue.", CompCode, Reason );

return Reason;
} /*end closeQ*/
```

```
/*********************************************************************/
/*   This function inquires on the specified queue to verify that    */
/*   it is a local queue. If so, the function tries to open the      */
/*   queue and return the object handle created.                     */
/*   An error will cause an appropriate message to be displayed.     */
/*********************************************************************/
MQLONG openQ( MQHCONN HConn, MQCHAR48 qName, MQHOBJ *openHObj )
{
MQLONG OpenOptions;
MQLONG CompCode, Reason;
MQOD   ObjDesc = { MQOD_DEFAULT };

Reason = inquireQ( HConn, qName, &HObj );
if ( MQRC_NONE == Reason )
{
/*                                                                   */
/*     Set MQOPEN options for a queue and set the queue name         */
/*                                                                   */
ObjDesc.ObjectType = MQOT_Q;
strncpy( ObjDesc.ObjectName, qName, MQ_Q_NAME_LENGTH );

/*                                                                   */
/*      The specified queue is set for inquire, browse and           */
/*      exclusive input. The context of any message taken            */
/*      from the queue must also be available if that                */
/*      message is to be forwarded to another queue later            */
/*      on.                                                          */
/*                                                                   */
OpenOptions = MQOO_INQUIRE         +
MQOO_BROWSE          +
MQOO_INPUT_EXCLUSIVE +
MQOO_SAVE_ALL_CONTEXT;

MQOPEN( HConn,
&ObjDesc,
OpenOptions,
openHObj,
&CompCode,
&Reason );

if( (MQCC_OK != CompCode) )
errorMessage( "Unable to open queue.", CompCode, Reason );

return Reason;
}
else
return Reason;
} /*end openQ*/
/*********************************************************************/



/*********************************************************************/
/*    This function opens the specified queue to inquire on the      */
/*    queue and definition type. The definition type is required     */
/*    to distinguish local queues from dynamic queues created when   */
/*    a model queue is opened for inquiry. If the queue is unable to */
/*    be opened or the inquiry shows the queue is not a local queue, */
/*    an appropriate error message is issued.                        */
/*********************************************************************/
MQLONG inquireQ( MQHCONN HConn, MQCHAR48 qName, MQHOBJ *openHObj )
{
MQHOBJ   HObj;
```

```
MQLONG   OpenOptions;
MQOD     ObjDesc = { MQOD_DEFAULT };
MQLONG   Selectors??( 2 ??);
MQLONG   IntAttrs??( 2 ??);
MQCHAR48 CharAttrs;
MQLONG   SelectorCount  = 2;
MQLONG   IntAttrCount   = 2;
MQLONG   CharAttrLength = 0;
MQLONG   CompCode, Reason, retCode;
char     errorMsg??(MSG_LENGTH??);

/*                                                                  */
/*      Set the open options to inquire on the queue.               */
/*                                                                  */
ObjDesc.ObjectType = MQOT_Q;
strncpy( ObjDesc.ObjectName, qName, MQ_Q_NAME_LENGTH );

OpenOptions = MQOO_INQUIRE;

MQOPEN( HConn,
&ObjDesc,
OpenOptions,
&HObj,
&CompCode,
&Reason );

/*                                                                  */
/*    If the open was unsuccessful, then issue an appropriate       */
/*    error message and return the reason code.                     */
/*                                                                  */
if( (MQCC_OK != CompCode) )
{
errorMessage( "Unable to open queue.", CompCode, Reason );
return Reason;
}

/*                                                                  */
/*       Set the inquire selectors for queue type and queue         */
/*       definition type.                                           */
/*                                                                  */
Selectors??( 0 ??) = MQIA_Q_TYPE;
Selectors??( 1 ??) = MQIA_DEFINITION_TYPE;

MQINQ( HConn,
HObj,
SelectorCount,
Selectors,
IntAttrCount,
IntAttrs,
CharAttrLength,
CharAttrs,
&CompCode,
&Reason );

/*                                                                  */
/*      If the inquire was successful then check whether the        */
/*      queue is local. If not, issue an error message and set      */
/*      W02-REASON to a negative value for return to caller.        */
/*      If the inquire is unsucessful, then determine whether       */
/*      the reason for failure was because we inquired on an        */
/*      attribute not applicable to local queues or some other      */
/*      reason, and issue an appropriate error message.             */
/*                                                                  */
```

```
if( (MQRC_NONE == Reason) )
{
if ( ( IntAttrs??( 0 ??) != MQQT_LOCAL ) ||
( IntAttrs??( 1 ??) != MQQDT_PREDEFINED ) )
{
sprintf( errorMsg, "Queue is not a local queue.");
printMessage( errorMsg );
retCode = -1;
return retCode;
}
}
else if ( (MQRC_SELECTOR_NOT_FOR_TYPE == Reason) )
{
sprintf( errorMsg, "Queue is not a local queue.");
printMessage( errorMsg );
retCode = -1;
return retCode;
}
else
errorMessage( "Unable to inquire whether queue is local.",
CompCode, Reason );

retCode = Reason;

/*                                                               */
/*               Close the queue.                                */
/*                                                               */
MQCLOSE( HConn,
&HObj,
MQCO_NONE,
&CompCode,
&Reason );

return retCode;
} /*end inquireQ*/
/*****************************************************************/


/*****************************************************************
*     This function adds the message to the chosen queue          *
*****************************************************************/
MQLONG putQ( MQHCONN HConn, MQCHAR48 qName,
             MQBYTE *Buffer, MQLONG BufferLength)
{
MQOD    ObjDesc  = { MQOD_DEFAULT };    /* Object  descriptor     */
MQMD    MsgDesc  = { MQMD_DEFAULT };    /* Message descriptor     */
MQPMO   scrMsgOpts = { MQPMO_DEFAULT }; /* Options which control  */
                                        /* the MQPUT call         */
MQHOBJ  HObj;                           /* Object Handle          */
MQLONG  Open_Options;                   /*                        */
MQLONG  CompCode, Reason;

/*  Initialize the object descriptor (MQOD) control block        */
ObjDesc.ObjectType = MQOT_Q;
strncpy( ObjDesc.ObjectName, qName, MQ_Q_NAME_LENGTH );

/*  Initialize the message descriptor and get message options     */
/*  control blocks                                               */
/*     No need to change the message descriptor (MQMD) control   */
/*     block because initialization default sets all the fields  */
/*                                                               */
/*     Initialize the put message options (MQPMO) control block  */
/*                                                               */
```

```
scrMsgOpts.Options = MQPMO_NO_SYNCPOINT ;

/* Put the message                                                  */
/*                                                                  */
MQPUT1(HConn,
      &ObjDesc,
      &MsgDesc,
      &scrMsgOpts,
      BufferLength,
      Buffer,
      &CompCode,
      &Reason);
if ( (CompCode != MQCC_OK) ) {
   errorMessage( "Error placing msg.", CompCode, Reason );
   }
return Reason;
} /*end putQ*/
/******************************************************************/


/******************************************************************/
/*    This function assembles the MQPUT message data              */
/******************************************************************/
MQLONG buildPut( char* putText )
{
static char line ??(MSG_LENGTH??);
char *ptr;
MQLONG retCode;

retCode = 0;
gets(line);
ptr=&line??(0??);
if (*ptr == 0) retCode = -1;                    /* No data          */
else {
   strncpy ( putText, line, strlen(line));      /* First msg line   */
   strcpy( line, "\0");                         /* Clear input area */
   gets(line);                                  /* Is there more?   */
   while (*ptr != 0) {
      strcat ( putText, SPACE );                /* add a space      */
      strncat ( putText, line, strlen(line));   /* concatenate next */
      strcpy( line, "\0");                      /* Clear input area */
      gets(line);                               /* Is there more?   */
   }
}
return retCode;
} /*end buildPut*/
/******************************************************************/


/*****************************************************************
*   This function does a destructive GET to a queue              *
*****************************************************************/
MQLONG getQ( MQHCONN HConn, MQCHAR48 qName)
{

MQLONG  CompCode, Reason ;              /* Completion code        */
MQOD    ObjDesc = { MQOD_DEFAULT };     /* Object descriptor      */
MQHOBJ  HObj;                           /* Object handle          */
MQMD    MsgDesc  = { MQMD_DEFAULT };    /* Message descriptor     */
MQLONG  DataLength ;                    /* Length of the message  */
MQLONG  OpenOptions;                    /*                        */
MQGMO   GetMsgOpts = { MQGMO_DEFAULT }; /*                        */
MQLONG  BufferLength = MQ_MSG_LENGTH;   /* Length of buffer       */
```

```
/* Initialize storage for buffer                                    */
/*                                                                  */
char  *buffer, *p;                        /* declare pointers       */
buffer = malloc(MQ_MSG_LENGTH);           /* allocate storage       */
p = buffer;                               /* set initial value to   */
*p = '\0';                                /* NULL                   */

/*  Initialize the object descriptor (MQOD) control block.          */
/*  (The initialization default sets StrucId, Version, ObjectType,  */
/*  ObjectQMgrName, DynamicQName, and AlternateUserid fields)       */
/*                                                                  */
strncpy(ObjDesc.ObjectName,qName,MQ_Q_NAME_LENGTH);

/*  Initialize the other fields required for the open call          */
/*                                                                  */
OpenOptions  = MQOO_INPUT_SHARED;

/* Open the queue                                                   */
/*   Test the output of the open call.  If the call failed,         */
/*   print an error message showing the completion code and         */
/*   reason code, then bypass processing, leave the program         */
/*                                                                  */
MQOPEN(HConn,
       &ObjDesc,
       OpenOptions,
       &HObj,
       &CompCode,
       &Reason);
if ((CompCode != MQCC_OK) || (Reason != MQRC_NONE)) {
   printf("\nMQOPEN error CC %d, RC %d ", CompCode, Reason);
   exit(0);
   }

/*  Initialize the message descriptor and get message options       */
/*  control blocks                                                  */
/*     No need to change the message descriptor (MQMD) control      */
/*     block because initialization default sets all the fields     */
/*                                                                  */
/*     Initialize the get message options (MQGMO) control block     */
/*     (The copy file initializes all the other fields)             */
/*                                                                  */
GetMsgOpts.Options = MQGMO_NO_WAIT      +
                     MQGMO_NO_SYNCPOINT +
                     MQGMO_ACCEPT_TRUNCATED_MSG;

/* Get the first message                                            */
/*                                                                  */
MQGET(HConn,
      HObj,
      &MsgDesc,
      &GetMsgOpts,
      BufferLength,
      buffer,
      &DataLength,
      &CompCode,
      &Reason);
if ( !( (CompCode == MQCC_OK) ||
     ((CompCode == MQCC_WARNING) &&
        (Reason == MQRC_TRUNCATED_MSG_ACCEPTED)) ) )
   {
   printf("\nMQGET  error CC %d, RC %d ", CompCode, Reason);
```

```
   exit(0);
   }

/* Tidy up buffer to terminate string and print the message        */

p = p + DataLength + 1;                    /* point to after msg  */
*p = '\0';                                 /* terminate with null */

printf("\nDestructive MQGET. Message length %u, text : \n%s",
           DataLength, buffer);
/*                                                                 */
/*   Close the queue                                               */
/*    Test the output of the close call.  If the call failed,      */
/*    print an error message showing the completion code and       */
/*    reason code                                                  */
/*                                                                 */
MQCLOSE(HConn,
        &HObj,
        MQCO_NONE,
        &CompCode,
        &Reason);
if ((CompCode != MQCC_OK) || (Reason != MQRC_NONE))
   {
   printf("\nMQCLOSE error CC %d, RC %d ", CompCode, Reason);
   exit(0);
   }
return;
} /*end getQ*/
/*****************************************************************/


/*****************************************************************/
/*    This function compiles the error message                   */
/*    The message consists of some text message, a completion code   */
/*    and a reason code.                                         */
/*****************************************************************/
void errorMessage( char *msgStr, MQLONG CompCode, MQLONG Reason )
{
sprintf( message, "%s   CompCode: %ld   Reason: %ld",
msgStr, CompCode, Reason );
printMessage( message );
} /*end errorMessage*/
/*****************************************************************/


/*****************************************************************/
/*    This function places a message onto the screen             */
/*****************************************************************/
void printMessage( char *msgStr )
{
printf("\n%s", msgStr);
} /*end printMessage*/
/*****************************************************************/


/*****************************************************************/
/*    This function creates a list of messages containing details    */
/*    of the first few messages browsed in the specified queue.      */
/*****************************************************************/
MQLONG createMsgList( MQHCONN HConn, MQHOBJ HObj, char* list)
{
#define maxSize   10                        /* maximum list size    */
MQGMO    GetMsgOpts = { MQGMO_DEFAULT };
```

```
MQMD     MsgDesc    = { MQMD_DEFAULT };
MQBYTE   Buffer??(MQ_MSG_LENGTH??);
MQLONG   DataLength;
MQLONG   CompCode;
MQLONG   Reason;
MQLONG   retCode = MQRC_NONE;
MQLONG   tableSpaceRemaining = maxSize;
char     lineNum??(2 + 1??);
MQCHAR8  formatName;
MQCHAR   putTime??(8 + 1??);
MQCHAR   putDate??(12 + 1??);
MQCHAR12 userId;
MQCHAR   putApplType??(8 + 1??);
MQCHAR28 putApplName;
char   msgText??(34 + 1??);
char   blank??(??) = "                                    ";
char   *year  = &MsgDesc.PutDate??( 0 ??);
char   *month = &MsgDesc.PutDate??( 4 ??);
char   *day   = &MsgDesc.PutDate??( 6 ??);
char   *hour  = &MsgDesc.PutTime??( 0 ??);
char   *min   = &MsgDesc.PutTime??( 2 ??);
char   *sec   = &MsgDesc.PutTime??( 4 ??);
char    message??(MSG_LENGTH??);
int     tableElement = 0;

/*                                                                  */
/*      Set up MQGET variables and browse the first message.        */
/*                                                                  */
memcpy( MsgDesc.MsgId,    MQMI_NONE, sizeof(MsgDesc.MsgId)    );
memcpy( MsgDesc.CorrelId, MQCI_NONE, sizeof(MsgDesc.CorrelId) );
GetMsgOpts.Options = MQGMO_BROWSE_FIRST         +
MQGMO_NO_WAIT              +
MQGMO_ACCEPT_TRUNCATED_MSG +
MQGMO_NO_SYNCPOINT;
MQGET( HConn,
HObj,
&MsgDesc,
&GetMsgOpts,
MQ_MSG_LENGTH,
Buffer,
&DataLength,
&CompCode,
&Reason );

/*                                                                  */
/*    If MQGET failed displayed an appropriate error message        */
/*    and return to the calling function.                           */
/*                                                                  */
/*    If successful set the MQGET browse options for further        */
/*    calls to MQGET.                                               */
/*                                                                  */
if( (MQCC_FAILED != CompCode) )
GetMsgOpts.Options = MQGMO_BROWSE_NEXT          +
MQGMO_NO_WAIT              +
MQGMO_ACCEPT_TRUNCATED_MSG +
MQGMO_NO_SYNCPOINT;
else
{
switch( Reason )
{
case MQRC_GET_INHIBITED    :
errorMessage( "Get Inhibited set on queue.",
CompCode, Reason );
```

```
break;
case MQRC_NO_MSG_AVAILABLE :
errorMessage( "No messages on queue.",
CompCode, Reason );
break;
default :
errorMessage( "Get from queue failed.",
CompCode, Reason );
break;
} /*end switch*/

return Reason;
}


/*                                                                  */
/*     Copy title lines to message list                             */
/*                                                                  */
#define TITLE1 "Msg  Put Date  Put Time User     Put      Start of Msg"
#define TITLE2 "No  MM/DD/YYYY HH:MM:SS Id       Appl     Text"
strncat(list, NEWLINE,1);
strncat(list, TITLE1, sizeof(TITLE1) );
strncat(list, NEWLINE,1);
strncat(list, TITLE2, sizeof(TITLE2) );


/*                                                                  */
/*     While there is still space remaining in the list             */
/*     place the message details into the list and read the         */
/*     next message from the queue.                                 */
/*                                                                  */
while( tableSpaceRemaining )
{
tableElement++;
tableSpaceRemaining--;

/*                                                               */
/*   Copy details read from queue into variables.                */
/*                                                               */
sprintf( lineNum, "%2.2d", tableElement );
memcpy( MsgId,    MsgDesc.MsgId,    sizeof(MsgId)    );
memcpy( CorrelId, MsgDesc.CorrelId, sizeof(CorrelId) );
sprintf( putTime, "%2.2s:%2.2s:%2.2s", hour, min,   sec  );
sprintf( putDate, "%2.2s/%2.2s/%4.4s", month, day, year );
memcpy( formatName, MsgDesc.Format, sizeof(formatName) );
memcpy( userId, MsgDesc.UserIdentifier, sizeof(userId) );
sprintf( putApplType, "%8.8ld", MsgDesc.PutApplType );
memcpy( putApplName, MsgDesc.PutApplName, sizeof(putApplName) );
strcpy( msgText, blank );                        /* 34 blanks  */
if ( DataLength < 34 ) {
   strncpy( msgText, Buffer, DataLength );
} else {
   strncpy( msgText, Buffer, 34 );
}

/*                                                               */
/*    Add a new line of message details to the message list      */
/*                                                               */
strncat(list, NEWLINE,       1              );
strncat(list, lineNum,       3              );
strncat(list, SPACE  ,       1              );
strncat(list, SPACE  ,       1              );
strncat(list, putDate,      10              );
```

```
strncat(list, SPACE  ,       1              );
strncat(list, putTime,       8              );
strncat(list, SPACE  ,       1              );
strncat(list, userId,        8              );
strncat(list, SPACE  ,       1              );
strncat(list, putApplName,   8              );
strncat(list, SPACE  ,       1              );
strncat(list, msgText, strlen(msgText)      );

/*                                                                  */
/*    Blank the MsgId and CorrelId so that any message on           */
/*    the queue will qualify on the next call to MQGET.             */
/*                                                                  */
memcpy( MsgDesc.MsgId,    MQMI_NONE, sizeof(MsgDesc.MsgId)    );
memcpy( MsgDesc.CorrelId, MQCI_NONE, sizeof(MsgDesc.CorrelId) );
MQGET( HConn,
HObj,
&MsgDesc,
&GetMsgOpts,
MQ_MSG_LENGTH,
Buffer,
&DataLength,
&CompCode,
&Reason );

/*                                                                  */
/*      Check for a failure with the previous MQGET call.           */
/*      If the failure was caused by having no more                 */
/*      messages on the queue then reset the error codes            */
/*      and break from while loop.                                  */
/*      Otherwise delete the message table, display an              */
/*      error message and return from function.                     */
/*                                                                  */
/*      MQCC_WARNINGs are in relation to truncated                  */
/*      messages which are accepted and hence ignored.              */
/*                                                                  */
if( (MQCC_FAILED == CompCode) )
if( (MQRC_NO_MSG_AVAILABLE == Reason) )
{
tableSpaceRemaining = 0;
Reason = MQRC_NONE; CompCode = MQCC_OK;
}
else
{
errorMessage( "Get from queue failed.", CompCode, Reason );
return Reason;
}
else
{
Reason = MQRC_NONE; CompCode = MQCC_OK;
}

} /*end while*/

return Reason;
} /*end createMsgTable*/
/*********************************************************************/
```

Appendix C : Sample source for the simple trigger monitor

```
         BEGIN NAME=MQII,VERSION=00,AMODE=31,XCL=NONE,SHR=NONE
         SPACE 1
*=====================================================================*
*        ALCS APPLICATION - TRIGGER MONITOR ECB BACKEND               *
*=====================================================================*
*        MQII -- INPUT EDIT PROGRAM FOR APPLICATION MQTM              *
*---------------------------------------------------------------------*
*        TRIGGER MESSAGE QNAME IS COPIED TO RO CRAS                   *
*=====================================================================*
         SPACE 1
         CMQTMA DSECT=YES,LIST=YES   TRIGGER MESSAGE STRUCTURE
         RSECT ,                     RESTORE CSECT BASE
         USING MQTM,R01              TRIGGER MESSAGE
         CM1CM REG=R02,SUFFIX=O      OUTPUT MESSAGE
         SPACE 1
******** COPY QNAME TO RO CRAS
         SPACE 1
MQII     DC    0H'0'
         L     R01,CE1CR0            LOAD BASE OF TRIGGER MESSAGE
         LA    R01,8(R01)            POINT TO QUEUE NAME
         GETCC LEVEL=D1,SIZE=L3      GET A STORAGE BLOCK
         L     R02,CE1CR1            LOAD BASE OF OUTPUT MESSAGE
         SPACE 1
         L     R04,=A(MSGH)          MOVE IN MESSAGE HEADER
         MVC   CM1TXTO(MSGHL),0(R04) MOVE IN MESSAGE HEADER
         LA    R15,48                LOAD INPUT LENGTH (QNAME)
         LR    R14,R1                LOAD MOVE-FROM ADDRESS
         LA    R04,CM1TXTO+(MSGHQNAM-MSGH)     TO ADDRESS
         LR    R05,R15               COPY MOVE LENGTH
         MVCL  R04,R14               COPY TEXT TO OUTPUT MESSAGE
         SPACE 1
MQII0100 DC    0H'0'
         LA    R04,CM1TXTO+MSGHL     LOAD END OF TEXT ADDRESS
         MVI   0(R04),#EOM           MOVE IN EOM CHARACTER
         LA    R05,CM1CRIO-1         POINT TO START OF ADDRESS - 1
         SR    R04,R05               CALCULATE MESSAGE CCT
         STH   R04,CM1CCTO           SAVE IN OUTPUT MESSAGE
         MVC   CM1CMWO(2),=AL1(#NOP,#CAR) MOVE IN CONTROL CHARS
         MVC   CM1CRIO,=X'000000'    MOVE IN CRI OF RO CRAS
         SPACE 1
         CRASC D1                    SEND MESSAGE ON LEVEL D1
         SPACE 1
         EXITC ,                     EXIT
         EJECT ,
*---------------------------------------------------------------------*
*        INDIRECTLY ADDRESSED CONSTANTS                               *
*---------------------------------------------------------------------*
         SPACE 1
******** MQB3 MESSAGE
         SPACE 1
MSGH     DC    C'MQII  Trigger received for: '
MSGHQNAM DC    CL48' '
MSGHL    EQU   *-MSGH
         SPACE 1
******** ERROR MESSAGES
         SPACE 1
         FINIS ,
         SPACE 1
         END   ,
```