MQSeries Client for VM and VSE

Session #1C2

Richard F. Lewis
IBM Washington System Center
rflewis@us.ibm.com
The information contained in this document has not been submitted to any formal IBM test and is distributed on an "AS IS" basis without any warranty either express or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

In this document, any references made to an IBM licensed program are not intended to state or imply that only IBM's licensed program may be used; any functionally equivalent program may be used instead.

Any performance data contained in this document was determined in a controlled environment and, therefore, the results which may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environments.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming or services in your country.
The following are trademarks of International Business Machines Corporation. Those identified with an (*) are registered trademarks of International Business Machines:

- Enterprise Systems Architecture/370
- Enterprise Systems Architecture/390
- ESA/370
- ESA/390
- IBM*
- System/370
- VM/ESA*
- OpenEdition
- MQSeries

The following are trademarks of the corporations identified:

- JAVA - is a trademark of Sun Microsystems
Introduction
Installation related topics
Usage related topics
Summary
Appendix: Sample Applications
Introduction
What Is MQSeries?

- A single, multi-platform API
- Assured message delivery
- Faster application development
- Time independent processing
- Application parallelism
Application Integration Styles
MQSeries
Application Programming Interface

MQI
MQCONN MQDISC MQOPEN MQCLOSE MQGET MQPUT MQPUT1 MQCMIT MQBACK MQNO MQSET

Application Program

Queue Manager
Process Definition Object
Queue Object
Queue Manager Object
Component of MQSeries product
- Installed on a separate machine from base product and server
- Supports execution of MQSeries applications
- Interacts using communications protocol with MQSeries servers
  ▶ Connects to queue managers through connecting with servers

Benefits of using MQSeries Client
- Eliminates need for full MQSeries implementation on client machine
- MQSeries applications can connect to multiple queue managers on different systems.
- Reduction of hardware requirements on client systems

Note: MQSeries client is not connected in any way with CMS Multitasking queues, or distributed queues
Servers that support client connections:
- Sun Solaris
- SINIX+ Pyramid DC/OSx
- AIX/6000
- Windows NT
- OS/400
- OS/2
- SunOS
- MVS/ESA
- HP-UX
- AT&T GIS NCR
- Digital VAX/Alpha
- Tandem NSK
- VSE/ESA as of Jan 1999
Message
- String of bits and bytes
  - Control information used by MQSeries
  - Application data

Message Queue
- Named object in which messages accumulate
- Belongs to one queue manager
- Has a name
  - Unique within owning queue manager
- Can have more than one per queue manager
- Not a stack, true queue
  - FIFO queue
  - Priority queue
Queue Manager
- System service providing message queuing facilities to applications
- Applications use MQ calls to request services
- Known by a name
  - Must be unique within network of queue managers
- More than one can be defined per system (only 1 on VSE)

MQI Channel
- Logical bi-directional communication link
- Connects client system to an MQ server
- Used only for transfer of MQI calls and responses
- Created at both client and server ends
- Two types
  - Client connection
  - Server connection
MQI Channel (cont.)
- Created by MQCONN call from application
- Destroyed by MQDISC call from application
- Application can connect to more than one server at a time
  ▶ Queue handle returned from MQCONN call
Installation
Installation

- Delivered on VM/ESA V2R3 system DDR
  - All parts located on MAINT 193 minidisk
  - Identified in CMS build lists for 193 minidisk
  - Serviced via CMS service stream

- Parts
  - AMQTEXT TXTLIB - main MQ VM client code
  - AMQTEXTC TXTLIB - C language bindings
  - AMQTEXTL TXTLIB - Cobol and PL/I language bindings
  - AMQTEXTA TXTLIB - Assembler language bindings
  - RXMQV MODULE - REXX interface to MQ calls (nucleus extension)
  - CMQC H - C header file
  - AMQOM MACLIB - Cobol, PL/I, Assembler include files
  - Sample programs for C, PL/I, Cobol, REXX
MQSeries Client Software Requirements

- VM/ESA V2 R2.0 or higher
- LE/370 Release 1.6
- C for VM Release 3.1 or compatible product
- TCP/IP Release 2.4 or VTAM
Following application language bindings are supplied
- IBM C for VM Release 3.1
- IBM VS Cobol II
- IBM OS/PLI Release 2.3
- REXX
- IBM Assembler
Using MQSeries Client
Environment Variables

- **MQCHLTAB**
  - Specifies name of client channel definition table
    - Default name is amqclchl.tab

- **MQSERVER**
  - Defines a minimal channel
  - Specifies location of MQSeries server and communication method
    - TCPIP port names specified in parenthesis
    - E.G. 9.82.1.101(2414)

- **MQCCSID**
  - Specifies coded character set number to be used

- **MQ_User_ID**
  - User name for authentication on server channel exit

- **MQ_PASSWORD**
  - Password for authentication on server channel exit
MQSeries does no security verification of its own
- Uses native facilities such as SAF on MVS to talk to ESM, or OS/400 security
- Addresses security issues with distributed queueing, and clients through provision of MCAUSER channel attribute, and security exits
- Channel security exits must be written by customer. Samples provided that interface with DCE on AIX, HP-UX, OS/2 WARP, SUN SOLARIS, and Windows NT

On VM/ESA
- Can set userid and password environment variables
  - Passed to channel security exit on server for validation
  - Supply value for MCAUserIdentifier at server for use in access control decisions on server
- Can write a security exit (in C, COBOL, PL1, Assembler) to interface with channel security exit at server. Allows for verification of server when connection is made, and passing of userid and password values from client
  - Can’t use globalv method to define client connection channel
- Access to client runtime code minidisk can be limited by standard VM security
- No server objects on VM (queues, queue managers, lists, …) to be protected from unauthorized access
- Link and access MAINT 193 for header file, maclibs, and txtlibs
- Make sure you also have access to:
  - LE/370 libraries
  - C libraries
- Create source program
  - Use C, REXX, Cobol, PL/I, or Assembler
- Global MQ client code library AMQTEXT TXTLIB
- Global MQ binding library appropriate to application language
- Global AMQOM MACLIB if using Cobol, PL/I, or Assembler
- Compile/assemble application, and GENMOD
- Define MQI channel for server and client communication
  - Use MQSERVER variable on client or
  - Create channel table on server and use MQCHLTAB variable on client to point to copy of table
    - Best alternative when defining multiple channels and MQSeries client machines at once
      - `GLOBALV SELECT CENV SETLP MQSERVER chnlname/trnstype/conname`
      - `GLOBALV SELECT CENV SETLP MQSERVER SYSTEM.DEF.SRVCONN/TCP/9.82.1.101`

- On server use MQSC DEFINE CHANNEL command
  - Ensure channel name matches channel name defined on client

- On server define and start queue manager, and listener
  - Default port for listener is 1414

- Execute application module
  - Nucxload RXMQV prior to invoking REXX MQ application
5686-A06 MQSeries for VSE/ESA V2R1

Software requirements
- VSE/ESA 2.3.1 or later
- CICS/VSE 2.3 (5686-026)
- VTAM for VSE/ESA 4.2 (5666-363) or later 4.x
- TCP/IP for VSE/ESA V1.3 (5686-A04) with PTFs
  - NFS (from CSI) - PQ14716 (UQ19196)
  - NFS Configuration Dialog - PQ16251 (UQ18722)
  - LE Socket Support - PQ16251 (UQ18646)
- LE/VSE 1.4 Runtime library (5696-067)
- MQSeries for VSE/ESA V2R1 PTFs
  - PQ21416 (UQ24423)
  - PQ21567 (UQ24461)
  - PQ21701 (UQ24561)
  - PQ22999 (UQ25561)
  - PQ23020 (UQ25803)
  - PQ25186 (UQ28360)
  - PQ25961 (UQ29335)
  - PQ26361 (UQ29678)
MQSeries for VSE/ESA V2R1 only
- Clients can only be connected with TCP/IP
- Queue Manager would have been previously defined
  ▶ On VSE/ESA only one Queue Manager can be defined
- Define local queues for application use
  ▶ Selection 2 from Configurations Main Menu (MQMT txn)
- Define MQI channel for client to use
  ▶ Selection 3 from Configurations Main Menu
  ▶ Specify channel type C, and protocol T
  ▶ Name must match name used on VM/ESA (case sensitive)
  ▶ Port defaults to 1414
- When MQSeries subsystem reinitialized, listener task will be ready to accept communication requests on port 1414 (or whatever port specified when Queue Manager defined)
VSE Master Terminal Main Menu

05/10/1999
00:04:25
MQMMP

IBM MQSeries for VSE/ESA Version 2.1.0
*** Master Terminal Main Menu ***
N1#13CIA
CICS
A001

SYSTEM IS ACTIVE

1. Configuration
2. Operations
3. Monitoring
4. Browse Queue Records

Please enter one of the options listed.
VSE Configuration Main Menu

IBM MQSeries for VSE/ESA Version 2.1.0

05/10/1999  IBM MQSeries for VSE/ESA Version 2.1.0  N1#13CIA
08:06:10  *** Configuration Main Menu ***  CICS
MQMNCFG  A001

SYSTEM IS ACTIVE

Maintenance Options:
1. Global System Definition
2. Queue Definitions
3. Channel Definitions

Display Options:
4. Global System Definition
5. Queue Definitions
6. Channel Definitions

Option: _

Function terminated.

5686-A06 (C) Copyright IBM Corp. 1998 All Rights Reserved.
ENTER = Process  PF2 = Main Menu  PF3 = Quit

IBM
VSE Queue Manager

Session A - WSCVM (TCP/IP) - [24 x 80]

05/10/1999 IBM MQSeries for VSE/ESA Version 2.1.0 M1#13CIA
00:05:05 Global System Definition CICS
MQMMSYS Queue Manager Information A001
Queue Manager ........ : MQVSE.QMANAGER
Description Line 1 .... : MQSeries for VSE/ESA V2
Description Line 2 .... :

Queue System Values
Maximum Number of Tasks . : 00000100 System Wait Interval : 0000005
Maximum Concurrent Queues . : 00000100 Max. Recovery Tasks : 0000
Allow TDQ Write on Errors : Y GSMT Allow Internal Dump : Y

Queue Maximum Values
Maximum Q Depth ........ : 01000000 Maximum Global Locks : 00000100
Maximum Message Size .... : 00002048 Maximum Local Locks : 00000100
Maximum Single Q Access . : 00000100 Checkpoint Threshold : 1000

Global QUEUE /File Names
Local Code Page .... : 00500 TCP/IP Listener Port : 01414
Configuration File . : MQFCNFG Licensed Clients . : 00020
LOG Queue Name . . . : SYSTEM.LOG
Dead Letter Name . . . : SYSTEM.DEAD.LETTER.QUEUE
Monitor Queue Name . : SYSTEM.MONITOR

Requested record displayed.
PF2 = Main Config PF3 = Quit PF4/ENTER = Read PF6 = Update
VSE Queue Definition

LOCAL QUEUE DEFINITION

Object Name: QUEUE.VM
Description line 1: 
Description line 2: 

Put Enabled: Y, Y=Yes, N=No
Get Enabled: Y, Y=Yes, N=No

Default Inbound status: A, A=Active, I=Inactive
Outbound: A, A=Active, I=Inactive

Dual Update Queue: 

Automatic Reorganize (Y/M): N

Record being updated - Press UPDATE key again.
VSE Channel Definition

Session A - WSCVM (TCP/IP) - [24 x 80]

05/10/1999
00:10:23
IBM MQSeries for VSE/ESA Version 2.1.0
NI#13CIA

Channel Record
DISPLAY
CICS

Last Check Point
Last Update 19990506 A001
MSN 00000000 Time 18:24:08 Interv 000000 Create Date 19990506
Name : SYSTEM.DEF.SRVCONN
Protocol : T (L/T) Port : 0000 Type : C (S/R/C)
Partner :

Allocation Retries : Get Retries
Number of Retries : 00000000
Delay Time - fast : 00000000 Delay Time : 00000000
Delay Time - slow : 00000000

Max Messages per Batch : 000001 Max Transmission Size : 000000
Message Sequence Wrap : 000000 Max Message Size : 000000

Mess Seq Req(Y/N) : Y Conves Cap (Y/N) : Y Split Msg(Y/N) : N

Transmission Queue Name :
TP Name:
Checkpoint Values : Frequency : 0000 Time Span : 0000
Enable(Y/N) N Dead Letter Store(Y/N) N
Channel record displayed.
PF2 =Menu PF3 =Quit PF4 =Read PF5 =Add PF6=Update PF9 =List PF12 =Delete
Sample REXX Application

/**************************************************************************
/* Description: Rexx version of Sample C program that gets messages from a message queue (MQGET) */
/**************************************************************************/

queue_mgr = ''
queue_name = ''
more = 1

Parse ARG queue_name queue_mgr .
hostname = '32.227.58.121'
channel = 'SYSTEM.DEF.SRVCONN'

If queue_name = '' Then
  Do
    Say "Required parameter missing - queue name"
    Exit 99
  End

Input values to create MQI channel
Say "Sample AMQSGET0 start"

"GLOBALV SELECT CENV SET MQSERVER "channel"/TCP/"hostname"

Address COMMAND 'GLOBAL LOADLIB SCEERUN AMQLLIB'
Address COMMAND 'SET STORECLR ENDCMD'
Address COMMAND 'NUCXLOAD RXMQV'

/* Initialise the interface */

mqrc = RXMQV('INIT')
If WORD(mqrc,1) <> MQCC_OK Then
    Do
        Say "Unable to initialize RXMQV environment"
        Exit WORD(mqrc,1)
    End

/* Connect to the Queue Manager */

mqrc = RXMQV('CONN', queue_mgr)
If WORD(mqrc,1) <> MQCC_OK Then
   Do
      Say "Unable to connect to Queue Manager" queue_mgr
      Exit WORD(mqrc,1)
   End
/* Open Queue for Input */

oo = MQOO_INPUT_AS_Q_DEF+MQOO_FAIL_IF_QUIESCING

mqrc = RXMQV('OPEN', queue_name, oo, 'hqn', 'ood.')
If WORD(mqrc,1) <> MQCC_OK Then
   Do
      Say "Unable to open queue:" queue_name "RC=" mqrc
      mqrc = RXMQV('DISC')
      Exit 99
   End
/* Read messages from Queue */
Call Get_messages

/* Stop access to a Queue */

mqrc = RXMQV('CLOSE', hqn, MQCO_NONE )

/* Disconnect from the QM (Closing hqn in the process) */

mqrc = RXMQV('DISC')

/* Remove the Interface functions from the Rexx Workspace ... */

mqrc = RXMQV('TERM')

Say "Sample AMQSGET0 end"

Exit 0

/*--------------------------------------------------------------------------*/
/* Subroutine Get_messages */
/*--------------------------------------------------------------------------*/

Get_messages:

/* Get all msgs from the queue */

more = 1
Do While more
  g.0 = 500
  g.1 = ''
  igmo.opt = MQGMO_WAIT + MQGMO_CONVERT
  igmo.WAIT = 15000

mqrc = RXMQV('GET', hqn,'g.','igmd.','ogmd.','igmo.','ogmo.')
If WORD(mqrc,1) <> MQCC_OK Then
  If RXMQV.LASTAC = MQRC_NO_MSG_AVAILABLE Then
    Do
      Say "No more messages"
      more = 0
      Iterate
    End
  Else
    Do
      Say "Get failed with reason code:" RXMQV.LASTAC
      more = 0
      Iterate
    End

  Say "message <"||g.1||">
End

Return
MQSeries JAVA Client

- Client API packaged as JAVA class library
- Available from:
- VM is not listed as a platform for this package
  - Works fine with JAVA for VM/ESA
  - Workstation version of package delivered as zip file
    - Installation instructions in readme file, and html pages unzipped from downloaded file
  - Upload workstation version to VM
    - Upload class files in binary, source files with ascii to ebcdic translation
    - Duplicate directory structure created on workstation in VM/ESA BFS
    - FTP can be used for file transfer, or easier yet, use NFS with trans=no on the mount command
- Class library can be used from native java programs, or from netrexx programs
import com.ibm.mq

queue_mgr = ''
queue_name = ''
more = 1

Parse ARG queue_name queue_mngr .
hostname = 'myworkstation'
channel = 'SYSTEM.DEF.SRVCONN'

If queue_name = '' Then
    Do
        Say "Required parameter missing - queue name"
        Exit 99
    End

Say "Sample AMQSGET0 start"
Do

MQException.log = null
MQEnvironment.hostname = hostname
MQEnvironment.channel = channel

qmgr = MQQueueManager(queue_mgr)
openoptions = int MQC.MQOO_INPUT_AS_Q_DEF + MQC.MQOO_FAIL_IF_QUIESCING

the_queue = qmgr.accessQueue(queue_name, openoptions, queue_mgr, -'', '')

gmo = MQGetMessageOptions()
gmo.options = MQC.MQGMO_WAIT + MQC.MQGMO_CONVERT
gmo.waitInterval = 15000

buffer = MQMessage()
buffer.correlationId = MQC.MQMI_NONE
buffer.messageId = MQC.MQMI_NONE
Loop While more
   the_queue.get(buffer, gmo)
   the_message = Rexx null
   the_message = Rexx buffer.readLine()
   the_message = the_message.substr(3).strip
   If the_message = 'DONE' Then
      Leave
   Say "message <"||the_message||">"
   buffer.clearMessage()
   buffer.correlationId = MQC.MQMI_NONE
   buffer.messageId = MQC.MQMI_NONE

Catch inex = MQException
   If inex.completionCode = inex.MQCC_FAILED Then
      If inex.reasonCode = inex.MQRC_NO_MSGAVAILABLE Then
         Do
            Say "No more messages"
            more = 0
         End
      Else
         Do
            Say inex.exceptionSource "failed with reason code: " -
               inex.reasonCode
            Exit (int inex.reasonCode)
         End
   End
End
the_queue.close()
qmgr.disconnect()

Catch ex = MQException
    If ex.completionCode = ex.MQCC_FAILED Then
        Do
            Say ex.exceptionSource "failed with reason code: " -
            ex.reasonCode
            Exit (int ex.reasonCode)
        End
    End

Say "Sample AMQSGET0 end"
Exit
Summary
MQSeries provides time independent program to program communication facility between many disparate platforms
- Asynchronous program to program communication
- Guaranteed message delivery
- TCP/IP and SNA supported as transport protocols

MQSeries client provides a "thin client" supporting the entire MQSeries API with very little resource requirement
- Applications can be placed on platforms where full MQSeries implementation does not exist
- VM/ESA can now host MQSeries applications
  - Provides easy reliable access to data managed by CICS, and other transaction processing systems on non-VM platforms
  - Expands application and data availability on VM/ESA
  - Brings VM advantages to data managed by applications on other platforms
- VSE/ESA can now support clients
Appendix: Sample Applications
On VM system
- global txtlib sceelked amqtext amqtextc commtxt cmssaaa
- global loadlib sceerun
- cc amqsput0
- cc amqsget0
- cplink amqsput0
- load cpobj
- genmod amqsput0 (from ceestart
- cplink amqsget0
- load cpobj
- genmod amqsget0 (from ceestart
Steps To Prepare Sample Application

- On system with full MQSeries installed
  - Define local queue manager (give default queue attribute if no other managers defined yet)
    - CRTMQM
  - Define local queue
    - DEFINE QLOCAL
  - Define channel
    - DEFINE CHANNEL(some.name) CHLTYPE(SVRCOMP) TRPTYPE(TCP)
  - Start queue manager
    - STRMQM
  - Start command processor
    - STRMQCSV
  - Start listener for tcp transport
    - RUNMQQLSR
Steps To Prepare Sample Application

- On VM system with MQSeries client support
  - Define global variable identifying target system
    - GLOBALV SELECT CENV SETLP MQSERVER SOME.NAME/TCP/9.82.1.246
  - Run amqspuit0 queue.name queue_manager_name
  - Enter message lines
  - Enter null line
  - Run amqsget0 queue.name queue_manager_name
  - Message lines entered earlier are pulled off queue, and deleted
Sample Application Execution

```
wscm2vm
Connection Edit Options Help
amqsput0 QUEUE.VM my.vm.queue
Sample AMQSPUT0 start
target queue is QUEUE.VM
This is message 1
This is message 2
This is message 3
This is message 4

Sample AMQSPUT0 end
Ready; T=0.12/0.16 17:23:40
amqsget0 QUEUE.VM my.vm.queue
Sample AMQSGET0 start
message <This is message 1>
message <This is message 2>
message <This is message 3>
message <This is message 4>
no more messages
Sample AMQSGET0 end
Ready; T=0.12/0.15 17:24:04
```
static char *sccsid = "@(#) samples/c/amqsget0.c, samples, p000 \
1.7 95/11/04 12:28:35";
/**************************************************************/
/*                                                                  */
/* Program name: AMQSGET0                                           */
/*                                                                  */
/* Description: Sample C program that gets messages from            */
/*               a message queue (example using MQGET)               */
/*                                                                  */
/* Statement: Licensed Materials - Property of IBM                  */
/*                                                                  */
/*                                                                  */
/*                                                                  */
/*                                                                  */
/*                                                                  */
/*                                                                  */
/* Function:                                                       */
/*                                                                  */
/* AMQSGET0 is a sample C program to get messages from a            */
/* message queue, and is an example of MQGET.                      */
/*                                                                  */
Sample Get Application

/* -- sample reads from message queue named in the parameter */
/* */
/* -- displays the contents of the message queue, */
/* */
/* assuming each message data to represent a line of */
/* */
/* text to be written */
/* */
/* messages are removed from the queue */
/* */
/* -- writes a message for each MQI reason other than */
/* */
/* MQRC_NONE; stops if there is a MQI completion code */
/* */
/* of MQCC_FAILED */
/* */
/* */
/* Program logic: */
/* */
/* Take name of input queue from the parameter */
/* */
/* MQOPEN queue for INPUT */
/* */
/* while no MQI failures, */
/* */
/* . MQGET next message, remove from queue */
/* */
/* . print the result */
/* */
/* . (no message available counts as failure, and loop ends) */
/* */
/* MQCLOSE the subject queue */
/* */
/* */
/***********************************************************/
Sample GET Application

/*
 * AMQSGET0 has 2 parameters -
 * - the name of the message queue (required)
 * - the queue manager name (optional)
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>

int main(int argc, char **argv)
{
    /* Declare MQI structures needed */
    MQOD     od = {MQOD_DEFAULT};    /* Object Descriptor */
    MQMD     md = {MQMD_DEFAULT};    /* Message Descriptor */
    MQGMO    gmo = {MQGMO_DEFAULT};  /* get message options */
    /* includes for MQI */
    include <cmqc.h>

    int main(int argc, char **argv)
    {
        /* Declare MQI structures needed */
        MQOD     od = {MQOD_DEFAULT};    /* Object Descriptor */
        MQMD     md = {MQMD_DEFAULT};    /* Message Descriptor */
        MQGMO    gmo = {MQGMO_DEFAULT};  /* get message options */
        MQHCONN  Hcon;                   /* connection handle */
        MQOBJ    Hobj;                   /* object handle */
        MQLONG   O_options;             /* MQOPEN options */
        MQLONG   C_options;             /* MQCLOSE options */
        MQLONG   CompCode;              /* completion code */
        MQLONG   OpenCode;              /* MQOPEN completion code */
        MQLONG   Reason;                /* reason code */
        MQLONG   CReason;               /* reason code for MQCONN */
        MQBYTE   buffer[101];          /* message buffer */
        MQLONG   buflen;                /* buffer length */
        MQLONG   messlen;               /* message length received */
        char     QMName[50];            /* queue manager name */
```c
printf("Sample AMQSGET0 start\n");
if (argc < 2)
{
    printf("Required parameter missing - queue name\n");
    exit(99);
}

/****************************************************************************
/*                                                                */
/*   Create object descriptor for subject queue                   */
/*                                                                */
/****************************************************************************
strcpy(od.ObjectName, argv[1]);
QMName[0] = 0;   /* default */
if (argc > 2)
    strcpy(QMName, argv[2]);

/****************************************************************************
/*                                                                */
/*   Connect to queue manager                                     */
/*                                                                */
/****************************************************************************
MQCONN(QMName,                  /* queue manager                  */
    &Hcon,                   /* connection handle              */
    &CompCode,               /* completion code                */
    &CReason);               /* reason code                    */
```
Sample GET Application

/* report reason and stop if it failed */
if (CompCode == MQCC_FAILED) {
    printf("MQCONN ended with reason code %ld\n", CReason);
    exit( (int)CReason );
}

/*****************************************************************************/
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* Open the named message queue for input; exclusive or shared */
/* use of the queue is controlled by the queue definition here */
/* */
/*****************************************************************************/

O_options = MQOO_INPUT_AS_Q_DEF   /* open queue for input */
+ MQOO_FAIL_IF_QUIESCING;   /* but not if MQM stopping */
MQOPEN(Hcon, /* connection handle */
    &od, /* object descriptor for queue */
    O_options, /* open options */
    &Hobj, /* object handle */
    &OpenCode, /* completion code */
    &Reason); /* reason code */

Open returns a handle representing the object that has been opened

return codes and reason codes defined in Application Programming Reference book

Open queue using queue defined defaults

Object handle from MQConn call must be specified. Object descriptor is a structure containing the name of the object, and type of object we are trying to open.
/* report reason, if any; stop if failed */
if (Reason != MQRC_NONE)
{
    printf("MQOPEN ended with reason code %ld\n", Reason);
}

if (OpenCode == MQCC_FAILED)
{
    printf("unable to open queue for input\n");
}

/*******************************/
/*    Get messages from the message queue                        */
/*    Loop until there is a failure                                 */
/* */
/*******************************/
CompCode = OpenCode; /* use MQOPEN result for initial test */
while (CompCode != MQCC_FAILED)
{
    buflen = sizeof(buffer) - 1; /* buffer size available for GET */
    gmo.Options = MQGMO_WAIT /* wait for new messages */
        + MQGMO_CONVERT; /* convert if necessary */
    gmo.WaitInterval = 15000; /* 15 second limit for waiting */
In order to read the messages in sequence, `MsgId` and `CorrelID` must have the default value. `MQGET` sets them to the values in for message it returns, so re-initialise them before every call.

```c
memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
MQGET(Hcon, /* connection handle */ 
    Hobj, /* object handle */ 
    &md, /* message descriptor */ 
    &gmo, /* get message options */ 
    buflen, /* buffer length */ 
    buffer, /* message buffer */ 
    &messlen, /* message length */ 
    &CompCode, /* completion code */ 
    &Reason); /* reason code */
```

CorrelID is an application-specified field used to relate messages on a queue. Can select messages from queue based on CorrelID, MSGID, or both. When browsing, MSGID and CorrelID should be set to null, which means get next record from current pointer.

Retrieve a message from a LOCAL queue. Supply connection handle, and object handle. Message returned in buffer supplied.
/* report reason, if any */
if (Reason != MQRC_NONE)
{
    if (Reason == MQRC_NO_MSG_AVAILABLE)
    {
        /* special report for normal end */
        printf("no more messages\n");
    }
    else
    {
        /* general report for other reasons */
        printf("MQGET ended with reason code %ld\n", Reason);

        /* treat truncated message as a failure for this sample */
        if (Reason == MQRC_TRUNCATED_MSG_FAILED)
        {
            CompCode = MQCC_FAILED;
        }
    }
}

/**********************************************************/
/* Display each message received */
/**********************************************************/
if (CompCode != MQCC_FAILED)
{
    buffer[messlen] = '\0';       /* add terminator */
    printf("message <\%s>\n", buffer);
}
}
End access to an object.
On completion, the object handle is set to a value indicating it is unusable.
break connection between queue manager and application program.
connection handle set to value that is not a usable handle.
This program comes with the MQSeries Client support for VM/ESA
/* Function:                                                  */
/*                                                          */
/* AMQPPUT0 is a sample C program to put messages on a message*/
/* queue, and is an example of the use of MQPUT.            */
/*                                                          */
/* -- messages are sent to the queue named by the parameter */
/*                                                          */
/* -- gets lines from StdIn, and adds each to target queue,*/
/*    taking each line of text as the content of a datagram */
/*    message; the sample stops when a null line (or EOF) */
/*    is read.                                              */
/* New-line characters are removed.                          */
/* If a line is longer than 99 characters it is broken up   */
/* into 99-character pieces. Each piece becomes the content */
/* of a datagram message.                                   */
/* If the length of a line is a multiple of 99 plus 1       */
/* e.g. 199, the last piece will only contain a new-line    */
/* character so will terminate the input.                   */
/*                                                          */
/* -- writes a message for each MQI reason other than       */
/* MQRC_NONE; stops if there is a MQI completion code of     */
/* MQCC_FAILED                                             */
/*                                                          */
Program logic:

MQOPEN target queue for OUTPUT
while end of input file not reached,
  read next line of text
  MQPUT datagram message with text line as data
MQCLOSE target queue

AMQSPUT0 has 2 parameters
  - the name of the target queue (required)
  - queue manager name (optional)

#include <stdio.h>
#include <stdlib.h>
#include <ctest.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    /* Declare file and character for sample input */
    FILE *fp;

    /* Declare MQI structures needed */
    MQOD  od = {MQOD_DEFAULT}; /* Object Descriptor */
    MQMD  md = {MQMD_DEFAULT}; /* Message Descriptor */
    MQPMO pmo = {MQPMO_DEFAULT}; /* put message options */
    /** note, sample uses defaults where it can **/
    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 */
    char buffer[100]; /* message buffer */
    char QMName[50]; /* queue manager name */
printf("Sample AMQSPUT0 start\n");

if (argc < 2)
{
    printf("Required parameter missing - queue name\n");
    exit(99);
}

/******************************************************************************/
/*                                                                           */
/*   Connect to queue manager                                              */
/*                                                                           */
/******************************************************************************/

QMName[0] = 0;   /* default */

if (argc > 2)
    strcpy(QMName, argv[2]);

MQCONN(QMName,         /* queue manager         */
&Hcon,                  /* connection handle      */
&CompCode,               /* completion code        */
&CR Reason);             /* reason code             */

/* report reason and stop if it failed                                     */
if (CompCode == MQCC_FAILED)
{
    printf("MQCONN ended with reason code %ld\n", CReason);
    exit( (int)CReason );
}

}
```c
strncpy(od.ObjectName, argv[1], (size_t)MQ_Q_NAME_LENGTH);
printf("target queue is %s\n", od.ObjectName);

O_options = MQOO_OUTPUT + MQOO_FAIL_IF_QUIESCING;
MQOPEN(Hcon, &od, O_options, &Hobj, &OpenCode, &Reason);

if (Reason != MQRC_NONE)
{
    printf("MQOPEN ended with reason code %ld\n", Reason);
}
```

Open queue such that subsequent MQPUT calls can be made. Queue is opened to put messages on it.
if (OpenCode == MQCC_FAILED)
{
    printf("unable to open queue for output\n");
}

/*******************************************************************/
/*                                                                */
/* Read lines from the file and put them to the message queue     */
/* Loop until null line or end of file, or there is a failure      */
/*                                                                */
/*******************************************************************/
CompCode = OpenCode;  /* use MQOPEN result for initial test */
fp = stdin;

while (CompCode != MQCC_FAILED)
{
    if (fgets(buffer, sizeof(buffer), fp) != NULL)
    {
        buflen = strlen(buffer);  /* length without null */
        if (buffer[buflen-1] == '\n')  /* last char is a new-line */
        {
            buffer[buflen-1] = '\0';  /* replace new-line with null */
            --buflen;  /* reduce buffer length */
        }
    }
    else buflen = 0;  /* treat EOF same as null line */
/*******************************************************************/
/* Put each buffer to the message queue */
/*******************************************************************/
if (buflen > 0)
{
    memcpy(md.Format, MQFMT_STRING, (size_t)MQ_FORMAT_LENGTH);
    MQPUT(Hcon, Hobj, &md, &pmo, buflen, buffer, &CompCode, &Reason);
    /* report reason, if any */
    if (Reason != MQRC_NONE)
    {
        printf("MQPUT ended with reason code %ld\n", Reason);
    }
    else /* satisfy end condition when empty line is read */
    {
        CompCode = MQCC_FAILED;
    }
Format name indicates nature of message data being sent to receiver. MQFMT_STRING is a built in format name specifying a message consisting entirely of characters.
/*************************************************************************
/*                                                                 /
/*     Close the target 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)
    {
        printf("MQCLOSE ended with reason code %ld\n", Reason);
    }
}
/**************************~
/*                           */
/*   Disconnect from MQM if not already connected                  */
/*                           */
**************************~
if (CReason != MQRC_ALREADY_CONNECTED)
{
    MQDISC(&Hcon, /* connection handle */
           &CompCode, /* completion code */
           &Reason); /* reason code */

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

/**************************~
/*                           */
/* END OF AMQSPUT0          */
/*                           */
/**************************~
printf("Sample AMQSPUT0 end\n");
return(0);
}
queue_mgr = ''
queue_name = ''
more = 1

Parse ARG queue_name queue_mgr .
hostname = '32.227.58.121'
channel = 'SYSTEM.DEF.SRVCONN'

If queue_name = '' Then
  Do
    Say "Required parameter missing - queue name"
    Exit 99
  End
Say "Sample AMQSPUT0 start"

"GLOBALV SELECT CENV SET MQSERVER "channel"/TCP/"hostname"

Address COMMAND 'GLOBAL LOADLIB SCEERUN AMQLLIB'
Address COMMAND 'SET STORECLR ENDCMD'
Address COMMAND 'NUCXLOAD RXMQV'

/* Initialise the interface */

mqrc = RXMQV('INIT')
If WORD(mqrc,1) <> MQCC_OK Then
  Do
    Say "Unable to initialize RXMQV environment"
    Exit mqrc
  End

/* Connect to the Queue Manager */

mqrc = RXMQV('CONN', queue_mgr )
If WORD(mqrc,1) <> MQCC_OK Then
  Do
    Say "Unable to connect to Queue Manager" queue_mngr
    Exit mqrc
  End

/* Open Queue for Output */

oo = MQOO_OUTPUT+MQOO_FAIL_IF_QUIESCING

mqrc = RXMQV('OPEN', queue_name, oo, 'hqn', 'ood.' )
If WORD(mqrc,1) <> MQCC_OK Then
  Do
    Say "Unable to open queue:" queue_name "RC=" mqrc
    mqrc = RXMQV('DISC')
    Exit 99
  End

/* Put messages on to Queue */
Call Put_messages

/* Stop access to a Queue */

mqrc = RXMQV('CLOSE', hqn, MQCO_NONE )

/* Disconnect from the QM (Closing hqn in the process) */

mqrc = RXMQV('DISC')

/* Remove the Interface functions from the Rexx Workspace ... */

mqrc = RXMQV('TERM')

Say "Sample AMQSPUT0 end"

Exit 0

/*====================================================================*/
/* Subroutine Put_messages                                         */
/*====================================================================*/
Put_messages:

/* Get all msgs from the queue */
more = 1

Say "Enter message for queue, null line to end"
Parse Pull new_msg

Do While (new_msg <> '')
    d.0 = LENGTH(new_msg)
    d.1 = new_msg
    imd.FORM = MQFMT_STRING

    mqrc = RXMQV('PUT', hqn,'d.','imd.','omd.','ipmo.','opmo.').
    If WORD(mqrc,1) <> MQCC_OK Then
        Do
            Say "Put failed: " mqrc
            Leave
        End
    End
    Parse Pull new_msg
End

Return