8 MQ update

February 2000

In this issue

Published by

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

North American office

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

Contributions

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

MQ Update on-line

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

Editor

Harry Lewis

Disclaimer

Readers are cautioned that, although the information in this journal is presented in good faith, neither Xephon nor the organizations or individuals that supplied information in this journal give any warranty or make any representations as to the accuracy of the material it contains. Neither Xephon nor the contributing organizations or individuals accept any liability of any kind howsoever arising out of the use of such material. Readers should satisfy themselves as to the correctness and relevance to their circumstances of all advice, information, code, JCL, scripts, and other contents of this journal before making any use of it.

Subscriptions and back-issues

A year's subscription to *MQ Update*, comprising twelve monthly issues, costs £255.00 in the UK; $380.00 in the USA and Canada; £261.00 in Europe; £267.00 in Australasia and Japan; and £265.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the July 1999 issue, are available separately to subscribers for £22.50 ($33.50) each including postage.

Printed in England.

Accessing OS/390 data from Microsoft Word

The exchange of messages between programs using MQSeries is asynchronous. This means that the sending program is de-coupled from the receiving one, and processing continues without the sender waiting for a response from the receiver. But quasi-synchronous data exchange can be simulated using a request/reply scenario. This may be used for accessing host-based data by calling host programs from Windows NT-based applications. The architecture is shown in Figure 1 overleaf.

THE ARCHITECTURE

The NT-based MQSeries client

MQSeries clients for all platforms may be downloaded from the SupportPac library (*www.software.ibm.com/ts/mqseries/txppacs*). The client software is available free of charge and is classified as 'category 3', which means that it is supplied under the terms and conditions of the International Program License Agreement (IPLA) and, hence, comes with a program defect service.

An application that is to run in the MQSeries client environment must be linked to the relevant client library. The link between the application and the MQSeries client code is established dynamically at runtime. The MQI channel, the communication type, and the address of the server must be known to the MQSeries client application. The simplest way of doing this is to use the client's *MQSERVER* environment variable:

```
SET MQSERVER=CLIENTCHANNEL/TCP/server-address(port)
```

The NT-based MQSeries client application

The complete range of MQI calls is available to client applications. The calls are synchronous and are actually executed on the server, so that the client must wait for their completion and reason code (and maybe a return message) before they may proceed. A communication error results in the failure of the call.

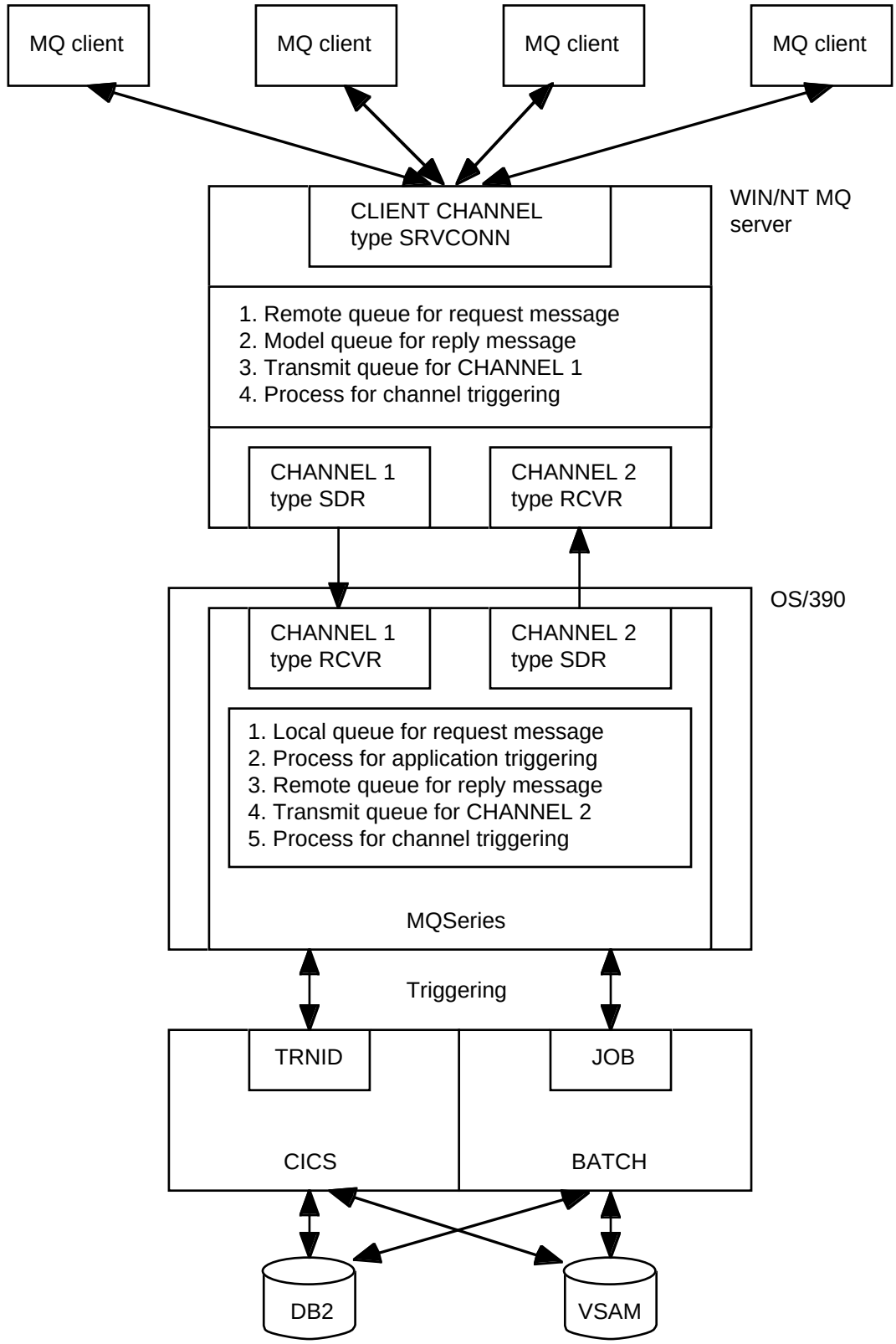

Figure 1: The basic architecture

As the process of updating queues is not local to the application, updates cannot be coordinated along with local resource updates. This means that a user-id/password pair must be sent from the client to the server for both authentication and access control (when required). User-written security exits may be used. In a request/reply scenario, the client application *MQOPEN*s a model queue and gets back a unique, non-persistent temporary queue name. A request message is *MQPUT* on the remote queue with the *REPYTOQUEUE* parameter. The temporary queue is *MQOPEN*ed for *MQGET* with the *WAIT* option. The client application then waits for the reply message.

The NT-based MQSeries server system

Once an MQSeries server installation is complete, it is necessary to define the following objects on the server if a request/reply set-up is to be used:

1 The MQI channel. For example:

```
DEFINE CHANNEL  (CLIENTCHANNEL) +
       CHLTYPE  (SVRCONN) +
       TRPTYPE  (TCP) +
       DESCR    ('MQSeries Client channel')
```

2 The sender channel to the OS/390 queue manager.

3 The receiver channel from the OS/390 queue manager.

4 The transmit queue for the sender channel, complete with *TRIGGER*, *PROCESS*, and *INITQ* specifications.

5 The process definition related to channel triggering.

6 The remote queue for the request messages. For example:

```
DEFINE QREMOTE ('REQUESTDATA') +
DESCR ('Remote queue for desktop data request') +
RNAME ('REQUESTDATA') +
RQMNAME (OS/390 qmgr name) +
XMITQ (transmission queue name)
```

7 The model queue for reply queues. For example:

```
DEFINE QMODEL ('REPLYDATA.MODEL.QUEUE') +
LIKE     ('SYSTEM.DEFAULT.MODEL.QUEUE') +
DESCR    ('Model queue for desktop data reply')
```

Message retry should be disabled on the receiver channel, as the temporary reply queue is deleted when a client application terminates abnormally, and so the receiving MCA is unable to put messages on the target queue. Instead, define and specify a dead letter queue for the queue manager to use for replies that cannot be delivered. Additionally, the channel initiator process (**runmqchi** or *START CHINIT*) and the listener process (**runmqlsr** or *START LISTENER*) must be enabled.

The OS/390 MQSeries queue manager

To create a request/reply set-up on the host queue manager, we need to implement Distributed Queue Management (DQM), which requires definitions of the following objects:

1 The receiver channel for the Windows NT-based queue manager.

2 The sender channel to the Windows NT-based queue manager.

3 The transmit queue for the sender channel, along with *TRIGGER*, *PROCESS*, and *INITQ* specifications.

4 The related process definition for channel triggering.

5 The local queue for the request messages with TRIGGER, PROCESS, and INITQ specifications.

6 The process definition related to application triggering.

7 The remote queue (queue manager alias) for reply messages. For example:

```
QUEUE NAME = WIN/NT qmgr name
REMOTE NAME = ''
REMOTE QUEUE MANAGER = WIN/NT qmgr name
TRANSMISSION QUEUE = sender channel xmit queue
```

A trigger monitor for the CICS environment is supplied with the OS/390 version of MQSeries. For triggering batch jobs, you must write your own batch trigger monitor. This task is discussed in *MQ Update* Issue 4 (October 1999), which also provides sample code.

The OS/390 MQSeries application

The application is triggered by the local request queue. The request

queue is *MQOPEN*ed and messages are retrieved using an *MQGET* call. The buffer area contains code for each required host function. Before the reply message for each request is *MQPUT1* on the remote queue, the *MQMD* and *MQOD* fields of the reply message should be set as follows:

```
MSGTYPE = MQMT_REPLY
PERSISTENCE = MQPER_NOT_PERSISTENT
FORMAT = MQFMT_STRING (for sender channel data conversion)
OBJECTNAME = replytoq name from MQMD of request message
OBJECTQMGRNAME = replytoqmgr name from MQMD of request message
```

Each reply message is syncpointed and, once all requests are performed, the request queue is *MQCLOSE*d and the application exits.

Sample code for an MQSeries client application

The sample code is written using Visual Basic for Applications (VBA). Hence, it can be called from any standard Microsoft application that includes VBA, such as Word, Excel, and Access.

The sample code is for an application that retrieves the address of a customer from a host database. The customer id is given to a host service module by the client application using MQSeries. The requested customer address is then returned by the service module to the client application via the reply queue.

Note that the module that is automatically loaded when a Word template is opened must be called ‘autonew’, and that it has to contain a procedure called ‘main’.

In order to be able to program MQSeries using either VB or VBA, it is necessary to download and install both the communication DLLs and the standard include member that allow either VB or VBA to use the MQI. These are available from category 2 SupportPac MA04 at the following URL:

```
http://www.software.ibm.com/ts/mqseries/txppacs/txpm1.html#win
```

Two DLLs (*mqicstd.dll* and *mqmcstd.dll*) have to be copied to the MQSeries client’s *bin* directory.

SAMPLE VBA CODE

```
'*********************************************************************'
'*  Include :                                                        *'
'*     Global definitions                                            *'
'*     Data structures                                               *'
'*     Constants                                                     *'
'*                                                                   *'
'*  Author  :       j.langnau                                        *'
'*********************************************************************'

Dim Rc               As Long
Dim Cc               As Long
Dim Rc1              As Long
Dim aHconn           As Long
Dim aHobj            As Long
Dim aHobjTemp        As Long
Dim aTempQueue       As String
Dim aDataLength      As Long
Dim aBufferLength    As Long
Dim aBuffer          As String
Dim aCustomer        As tCustomerVs
Dim aStart           As Integer
Dim sendDataStruc    As Srv_Parameters
Dim receiveDataStruc As RetParameters
Dim aMsgDesc         As MQMD
Dim aObjDesc         As MQOD
Dim aGetMsgOpts      As MQGMO
Dim aPutMsgOpts      As MQPMO

Global Const gRemoteQueue      = "DTC1.REQUESTDATA"
Global Const gReplyModelQueue  = "DTC1.REPLYDATA.MODEL.QUEUE"
Global Const gDynamicQueueName = "DTC1.REPLYDATA.*"
Global Const gWaitInterval     = 6000
Global Const gMaxBufferLength  = 512

Type tCustomerVs    'Data-area
    CUSTCOUNTID         As String * 2    ' Country code
    CUSTOMERID          As String * 4    ' Customer Id
    CUSTOMERFACTORINGID As String * 9    ' Factoring number
    CUSTOMERNAME        As String * 30   ' Name
    CUSTOMERNAME2       As String * 30   ' Name 2
    CUSTOMERSTREET      As String * 30   ' Street
    CUSTOMERZIPCODE     As String * 5    ' Zip code
    CUSTOMERCITY        As String * 30   ' City
    CUSTOMERFACS        As String * 21   ' Fax
    CUSTOMERCOUNTRYNAME As String * 20   ' Country
End Type

'*********************************************************************'
```

```
Type Srv_Parameters             ' Structure for Service module
    SrvModuleType   As String * 2
    SrvModul        As String * 8
    SrvId           As String * 2
    SrvVer          As String * 1
    SrvSubId        As String * 2
    SrvSubVer       As String * 1
    SrvLanguage     As String * 2
    SrvMaxLenReplyQ As String * 10
End Type
Const SrvHeaderLength = 24

'*********************************************************************'
'* Structure of Returnheader (Service module)
'*********************************************************************'
Type RetParameters
    Rtc     As String * 1
    RsType  As String * 8
    Rsc     As String * 10
    msg     As String * 80
End Type

Const RetHeaderLength = 99

'*********************************************************************'
' Set defaults for Service module
'*********************************************************************'

Sub Srv_Parameters_defaults(struc As Srv_Parameters, aSrvId As String)
    With struc
        .SrvModuleType   = "01"
        .SrvModul        = "CRSTAM  "    ' PLI service module
        .SrvId           = aSrvId        ' Id for service dispatcher
        .SrvVer          = "1"           ' Version
        .SrvSubId        = "  "
        .SrvSubVer       = " "
        .SrvLanguage     = "DE"          ' Language code for messages
        .SrvMaxLenReplyQ = "0000000000"     'init
        Mid(.SrvMaxLenReplyQ, Len(.SrvMaxLenReplyQ) - 2, 3) = _
                                                gMaxBufferLength
    End With
End Sub
'*********************************************************************'
'*
'*********************************************************************'
Sub put_Parms_into_buffer(struc As Srv_Parameters, Buffer As String)
    Buffer = ""
    With struc
        Buffer = .SrvModuleType
        Buffer = Buffer + .SrvModul
```

```
        Buffer = Buffer + .SrvId
        Buffer = Buffer + .SrvVer
        Buffer = Buffer + .SrvSubId
        Buffer = Buffer + .SrvSubVer
        Buffer = Buffer + .SrvLanguage
        Buffer = Buffer + .SrvMaxLenReplyQ
    End With
End Sub
'*********************************************************************'
'*
'*********************************************************************'
Sub get_Parms_from_Buffer(struc As RetParameters, Buffer As String)
'receive the Returnheader from the buffer
Dim pos As Integer
    pos = 1
    With struc
        .Rtc    = Mid(Buffer, pos, Len(.Rtc))
         pos    = pos + Len(.Rtc)
        .RsType = Mid(Buffer, pos, Len(.RsType))
         pos    = pos + Len(.RsType)
        .Rsc    = Mid(Buffer, pos, Len(.Rsc))
         pos    = pos + Len(.Rsc)
        .msg    = Mid(Buffer, pos, Len(.msg))
    End With
End Sub

'*********************************************************************'
'*
'*********************************************************************'
Sub put_buffer_in_tCustomervs(struc As tCustomerVs, aBuffer As String)

Dim pos As Integer
'jl    aBuffer = Qcv_ansi(aBuffer)
    pos = RetHeaderLength + 1
    With struc
         .CUSTOMERFACTORINGID = Mid(aBuffer, pos, _
                                    Len(.CUSTOMERFACTORINGID))
          pos = pos + Len(.CUSTOMERFACTORINGID)
         .CUSTOMERNAME = Mid(aBuffer, pos, Len(.CUSTOMERNAME))
          pos = pos + Len(.CUSTOMERNAME)
         .CUSTOMERNAME2 = Mid(aBuffer, pos, Len(.CUSTOMERNAME2))
          pos = pos + Len(.CUSTOMERNAME2)
         .CUSTOMERSTREET = Mid(aBuffer, pos, Len(.CUSTOMERSTREET))
          pos = pos + Len(.CUSTOMERSTREET)
         .CUSTOMERZIPCODE = Mid(aBuffer, pos, Len(.CUSTOMERZIPCODE))
          pos = pos + Len(.CUSTOMERZIPCODE)
         .CUSTOMERCITY = Mid(aBuffer, pos, Len(.CUSTOMERCITY))
          pos = pos + Len(.CUSTOMERCITY)
         .CUSTOMERFACS = Mid(aBuffer, pos, Len(.CUSTOMERFACS))
          pos = pos + Len(.CUSTOMERFACS)
```

```
         .CUSTOMERCOUNTRYNAME = Mid(aBuffer, pos, _
                                    Len(.CUSTOMERCOUNTRYNAME))
    End With
End Sub

'*********************************************************************'
'*
'*********************************************************************'
Sub put_tCustomerVs_in_buffer(struc As tCustomerVs, Buffer As String)
Dim pos As Integer
    With struc
         Buffer = Buffer + .CUSTCOUNTID
         Buffer = Buffer + .CUSTOMERID
    End With
End Sub

'*********************************************************************'
Sub DoWriteText(aText As String, aUnit As Integer)
    Selection.Text = aText
    Selection.EndOf 1
    Selection.Next
End Sub
Function DoOpenTempQueue(aHconn As Long, aHobj As Long) As String
Dim Cc, Rc As Long
Dim aObjDesc As MQOD
Dim aOption As Long

    MQOD_DEFAULTS aObjDesc

    aObjDesc.ObjectName = gReplyModelQueue
    aOption             = MQOO_INPUT_SHARED

    MQOPEN aHconn, aObjDesc, aOption, Hobj, Cc, Rc

    If Cc > 0 Then
        DoOpenTempQueue = ""
        MsgBox "Unable to open the queue " & aObjDesc.ObjectName _
                         & " ! RC(" & Rc & ")", vbCritical, "MQSeries"
    Else
        DoOpenTempQueue = aObjDesc.ObjectName
        aHobj           = Hobj
    End If

End Function

'*********************************************************************'
'*                                                                   *'
'*  Module name:    Main                                             *'
'*                                                                   *'
'*  Description:    This procedure is called when a new document     *'
```

```
'*                  is created.                                          *'
'*                                                                       *'
'*  Author     :    j.langnau                                            *'
'*                                                                       *'
'************************************************************************'

Sub Main()
Dim Rc As Boolean
Dim kd As tCustomerVs
Dim msg As String

        kd.CUSTOMERID = 12345   ' static input
        kd.CUSTCOUNTID = 0      ' static input

        DoSearchCustomer kd, msg

        If msg = "" Then
            With kd
                ' Printout the result to the Word Template
                DoWriteText .CUSTOMERNAME & Chr(13), 1
             End With
        Else
            DoWriteText msg & Chr(13), 1
        End If
End Sub

'************************************************************************'
'*                                                                      *'
'*  Module name:    mFunction                                           *'
'*                                                                      *'
'*  Description:    This module contains functions that can be          *'
'*                  called by the application.                          *'
'*                                                                      *'
'*  Author     :   j.langnau                                            *'
'*                                                                      *'
'************************************************************************'

Sub DoSearchCustomer(aCustomer As tCustomerVs, retMsg As String)
    Dim dataBuffer As String

    put_tCustomerVs_in_buffer aCustomer, dataBuffer

    PerformFunction dataBuffer, "RK"

    If receiveDataStruc.Rtc = "0" Then
        put_buffer_in_tCustomervs aCustomer, dataBuffer
        retMsg = ""
    Else
        retMsg = receiveDataStruc.msg
    End If
End Sub
```

```
'*********************************************************************'
'*                                                                   *'
'*  Name        : PerformFunction                                    *'
'*                                                                   *'
'*  Description : Service dispatcher for data access.                *'
'*                Our sample contains only one service.              *'
'*                                                                   *'
'*  Steps       : 1. Connect to the queue manager                    *'
'*                2. Open the temporary reply queue defined as       *'
'*                   modelqueue *'                                   *'
'*                3. Open the output queue that triggers the host    *'
'*                   program.                                        *'
'*                4. Write to the output queue                       *'
'*                5. Close the output queue                          *'
'*                6. Get/wait for reply                              *'
'*                7. Output the result to the new document           *'
'*                8. Close the temporary reply queue                 *'
'*                   (option=delete).                                *'
'*********************************************************************'
Sub PerformFunction(dataBuffer As String, SrvId As String)
'connect to the queue manager
    MQCONN QMgrName, aHconn, Cc, Rc
'create temporary reply queue
    aTempQueue = DoOpenTempQueue(aHconn, aHobjTemp)
'open output queue
    MQOD_DEFAULTS aObjDesc
    aObjDesc.ObjectName = gRemoteQueue
    aOption = MQOO_OUTPUT
    MQOPEN aHconn, aObjDesc, aOption, aHobj, Cc, Rc
'set SrvParameters
    Srv_Parameters_defaults sendDataStruc, SrvId
    put_Parms_into_buffer sendDataStruc, aBuffer
    aBuffer = aBuffer + dataBuffer
'MQPUT
    MQMD_DEFAULTS aMsgDesc
    aMsgDesc.MsgType    = MQMT_REQUEST
    MQPMO_DEFAULTS aPutMsgOpts
    aBufferLength       = Len(aBuffer)
    aMsgDesc.Format     = MQFMT_STRING
    'aMsgDesc.ReplyToQ  = aReplyQueue
    aMsgDesc.ReplyToQ   = aTempQueue
    aMsgDesc.ReplyToQMgr = aReplyQMgr
    MQPUT aHconn, aHobj, aMsgDesc, aPutMsgOpts, aBufferLength, _
          aBuffer, Cc, Rc
'close the output queue
    MQCLOSE aHconn, aHobj, aOption, Cc, Rc1
'MQGet Wait
    MQMD_DEFAULTS aMsgDesc
    MQGMO_DEFAULTS aGetMsgOpts
    aGetMsgOpts.Options      = MQGMO_WAIT
```

```
    aGetMsgOpts.WaitInterval = gWaitInterval
    aBuffer                  = Space(gMaxBufferLength)
    aMsgDesc.CodedCharSetId  = 437
    aBufferLength            = Len(aBuffer)
    MQGET aHconn, aHobjTemp, aMsgDesc, aGetMsgOpts, aBufferLength, _
          aBuffer, aDataLength, Cc, Rc1
        get_Parms_from_Buffer receiveDataStruc, aBuffer

    Select Case Rc1
       Case MQRC_NONE
          If receiveDataStruc.Rtc = "0" Then
             dataBuffer = aBuffer
          End If
       Case Else
            receiveDataStruc.msg = Rc1 & "-" & ErrorText
     End Select
'close the temp queue
     aOption = MQCO_DELETE
     MQCLOSE aHconn, aHobjTemp, aOption, Cc, Rc1
 End Sub
```

I would like to thank Joachim Langnau of LANGNAU Consulting (*http://www.langnau.com*) for his contribution to this article, especially for writing the MQSeries client program.

Raimund Kleebaur
MQSeries Programmer
Hugo Boss AG (Germany)

E-mail generation from MQSeries using Notes

REQUIREMENT

Many applications, including ones running on mainframes, need to generate e-mail messages as a result of business transactions. The question then arises how to interface the program to the corporate e-mail system. Using MQSeries as middleware has the benefit of de-coupling the application from the e-mail system, which may, in any case, be on a different platform. The asynchronous nature of e-mail is also a good match to MQSeries' approach. This article describes how

to build a bridge between MQSeries messages and Lotus Notes, where the primary use of the bridge is to handle outgoing e-mail.

ALTERNATIVES

There are many ways in which to join Notes and MQSeries, though most of them are designed from the point of view of the Notes application needing access to data on other platforms. In this sense, the Notes application initiates the request. By contrast, in our case the request is coming from the MQSeries side (for example, from a CICS transaction) as far as the e-mail application is concerned. What's needed is a means of transporting the message data to a Notes application and generating a Notes e-mail based on the message's contents.

Having researched the alternatives, I decided that the best approach was to write a Notes program (known as an 'agent') in Lotuscript and to connect to MQSeries using the Lotuscript Extension for MQSeries (MQLSX). This requires the least programming effort and represents a standard approach to Notes programming. As Lotuscript is interpreted, no compilers are required and most technical staff should be able to understand how the code works without special training.

Another advantage of this approach is that it incurs no purchase costs for the integration software, as MQLSX is free. Some other approaches require components, such as the Lotus Notes MQSeries Enterprise Integration product (MQEI), which are chargeable. The alternative approaches also require at least as much programming overall to achieve the same end result – while they tend to simplify the mainframe connection (something that is not complex with MQSeries anyway, especially using TCP/IP), they incur a greater programming overhead than Lotuscript for generating e-mail.

COMPONENTS

As usual in the era of the Internet, it's possible to find examples on linking MQSeries and Notes on the Web, the nearest one to my requirements being one from IBM called 'MQSeries link extra agent for Notes'. This provides a complete working agent to transform MQSeries messages into documents in a Notes database. It does not,

however, provide any specific support for e-mail generation, and so we decided not to use it but to write a smaller, purpose-built agent. However, I mention the availability of this IBM sample code as it may be useful to other readers.

One of the nice things about Notes is the high degree of integration between its various components, most of which are stored in a single Notes database. This includes the configuration data, the agent code, the saved messages (if required), and the various forms and views. This article includes an exported version of the agent; to recreate the Notes version, simply import the ASCII text file (which is available to *MQ Update* subscribers at Xephon's Web site) into a Notes database when editing an agent component (you may need a Notes developer to do this, though it's a fairly straightforward procedure).

You also need to obtain the Lotuscript extension for MQSeries and the MQSeries Trigger Monitor for Lotus Notes (they're available from the IBM MQSeries Web site free of charge).

In addition, you need either an MQSeries client (free) or an MQSeries server installed on the Notes server. For the purpose of my development, Windows NT was used to run Notes, though the code should be portable to any other MQSeries and Notes platform. Either a Notes client or Notes server can be used, though many sites will find it convenient to host this bridge on an existing Notes server.

MODE OF OPERATION

The agent is designed to be triggered on message arrival, though it could alternatively wait for messages, the choice being essentially about whether you want the agent to run continuously (this decision has implications for Notes agent threads). The agent has a configuration document that's part of the database, and the mode of operation is set as a parameter in this document and does not require coding to change. To use triggering, the trigger monitor program runs as a separate Windows NT task and waits for trigger messages to arrive. On the arrival of a message, the Notes agent starts immediately. Other ways to execute the agent include using a standard Notes agent schedule and manually running the agent from the database action menu.

The agent provided can be set to wait for MQSeries messages, and may also be configured to retry as many times as desired. Beware, though, that using MQGET with 'wait' causes the Notes thread to remain in use – it will not release control back to Notes. My recommendation is to trigger on arrival, perhaps wait for a second or so for subsequent messages, and then terminate. It's important to coordinate the mode of operation with the type of triggering defined in MQSeries. As this is a complex subject, and it's beyond the scope of this article to explore it in detail, I'd just suggest using both *TRIGGER(FIRST)* and scheduling the agent to run occasionally to clear up any problems that can happen with triggering when the agent fails.

FUNCTION

The Lotuscript agent carries out the following actions when it's invoked:

1 Declare various variables used by the agent program.

2 Read parameters from the configuration document in the current database.

3 Connect to the queue manager and open the input queue.

4 Get the first message from the queue and parse it into fields based on offsets.

5 Build the mail document using these fields.

6 Send the document either using the 'send' method or saving it in 'mail.box'.

7 Commit the message syncpoint, deleting it from the input queue.

8 Get the next message from the queue (if the queue is empty, then either retry, wait, or exit, depending on the configuration parameters).

9 Close the input queue and disconnect from the queue manager.

10 Terminate (return control back to Notes).

A test agent to generate an MQSeries message on a queue is provided at the end of the article.

AGENT PROGRAM LOGIC

The agent processes the entire input MQSeries queue until it's empty. The MQGET issued can wait for a message, and the time to wait is coded as a parameter. The agent also retries after the queue is empty, and the number of retries is also coded as a parameter.

While no retries are actually required, I recommend using one to catch any messages that may have arrived in the short time that the agent is running. The best setting for you depends on how your agent is run – if it's scheduled, then retries are probably sensible; if it's triggered, then they're not essential.

However, the agent should neither retry frequently nor have long wait periods, as either may unnecessarily tie up the server (unless agents are set to multithreaded, with the number of threads being greater than one). Another option is to use a retry value of '999' for indefinite looping in the agent.

CONFIGURATION OF THE AGENT

A configuration document is created using a form in the database. This method avoids hard-coding values, such as the name of the queue manager. These values are read by the agent when it initializes. An example configuration is shown below.

- Name of configuration: *Default*
  Use 'Default' for live version.
- MQSeries queue manager: *TST1*
  Change as required.
- MQSeries incoming queue: *TEST.EMAIL.QUEUE*
  Change as required.
- MQSeries GET wait interval: *1000*
  Milliseconds
- GET retries after no messages available: *2*

'0' means exit immediately if the queue is empty, '999' means loop continuously, and any values between are allowed.

- Mail type: *Create*
  'Send' is the standard (for Notes client), 'Create' results in a new document in the server's *mail.box*.
- Mail 'from/reply-to', if blank: *e-mail Gateway*
  Match the mailing database entry.
- Destination to use if blank in message: *Dead e-mails*
  Notes address to which to default.

Most of the values are self-explanatory. An interesting one is the mail type – if e-mail is sent using the standard Notes 'send document' method, it acquires the 'from' address of either the Notes client or server that executes the agent (it may also inherit the agent signature). For this sort of bridge, the apparent origin of the e-mail (and therefore the 'reply-to' address) is application-dependent and should be supplied in the MQSeries message. In order to 'fake' this address, the e-mail should be created directly in the server's mail box file (the agent has code to handle this).

I have also provided default values for both the 'from' and 'dest' addresses in case they're left blank in the MQSeries message. This is mainly useful for testing. The MQSeries message layout is a simple string with various fields that are extracted and used to create the document. The layout can be changed as required.

DEALING WITH REPLIES

As the Notes 'from/reply-to' field is set in the outgoing e-mail, replies can be directed automatically either to a person who has a mail box on the same server or a group mailbox for manual processing. If automated reply handling is required, then a 'mail in' record can be configured on the Notes server for a reply address. Replies are then stored in a Notes database automatically as documents, and a Notes agent can either be developed or configured to process them on or after arrival.

You can use a 'mail in' technique that's similar to the reverse of the process used in this agent to convert incoming e-mail into MQSeries

messages. Sample Lotuscript code for this can be found at *http://www.lotus411.com* and *http://www.lotus.com* (and other locations).

INSTALLING THE MQSERIES CLIENT

Although you could use an MQSeries server as the client, it's cheaper and simpler to use an MQSeries client. These are free of charge and downloadable from the Web as support pack MAC4 (see *http://www.software.ibm.com/mqseries*). Use the latest client version (Version 5.1 at the time of writing) for Windows 32-bit platforms, which is installed on the Notes server using **setup.exe**.

INSTALLING THE MQLSX FILES

The MQLSX files are supplied free of charge by IBM as support pack MA7D (go to *http://www.software.ibm.com/mqseries*, where the file *ma7d.zip* needs to be downloaded). **setup.exe** is executed on the Notes server to install the support pack, which includes comprehensive documentation on the use of MQSeries LSX.

INSTALLING THE TRIGGER MONITOR FOR LOTUS NOTES

Although I don't normally recommend triggering, it is suitable for this type of application, and IBM provides a Trigger Monitor for Notes Agents in support pack MA7E. Two versions of the trigger monitor executable are provided – one for MQSeries clients and one for MQSeries servers.

Full details of the configuration of the trigger monitor are provided in the support pack. Using the trigger monitor requires the correct MQSeries definitions, including:

1 The MQSeries e-mail request queue (the application data queue). Triggering should be enabled on this queue.

2 An initiation queue linked to the above.

3 A process entry linked to the above. This contains the Notes agent name and database name, which must match the names on the Notes server and in the Notes database. The trigger monitor's documentation has more details on this.

It is possible (and even desirable) to make the bridge program act as a direct MQSeries client of an MVS queue manager. It can also be triggered directly from an MVS initiation queue. However, in order to set the right APPLTYPE (*NotesAgent* or *22*) on an MVS process object, an APAR (PQ21707) needs to be installed on MQSeries for MVS. If this fails, contact IBM Hursley for a modified client trigger monitor program that circumvents this.

The trigger type should be set to *FIRST* on the request queue. This creates a trigger message when the queue depth changes from zero to one. The agent then reads all messages on the queue, which is, therefore, ready for the next trigger. However, if the agent fails and the queue depth is left at a value greater than zero, the queue is then unable to generate any more triggers. To avoid this, the Notes agent should be scheduled to run at regular intervals (perhaps daily, depending on your requirements) using a standard Notes schedule.

EXAMPLE MESSAGE LAYOUT

- *Field:* E-mail destination ‘send-to’
  Offset: 001
  Length: 80
  Description: Notes’ ‘send-to’ field, left justified and blank padded.
- *Field:* E-mail destination ‘CC’
  Offset: 081
  Length: 80
  Description: Notes’ ‘CC’ field, left justified and blank padded.
- *Field:* E-mail destination ‘BC’
  Offset: 161
  Length: 80
  Description: Notes’ ‘BC’ field, left justified and blank padded.
- *Field:* E-mail subject
  Offset: 241
  Length: 80
  Description: Notes’ ‘subject title’, left justified and blank padded.
- *Field:* E-mail ‘reply-to/from’

Offset: 321
Length: 80
Description: Notes' 'reply-to field', left justified and blank padded (set as 'from' field).

- *Field:* Number of body lines
  Offset: 401
  Length: 5
  Description: Number of lines, as char '99999' (with leading zeros).

- *Field:* Body text data (lines)
  Offset: 406
  Length: N x 80
  Description: As required.

ENVIRONMENT VARIABLES FOR MQ CLIENT

- *Name:* MQCCSID
  Value: 850
  Comment: Character set number.

- *Name:* MQSERVER
  Value: CHANNEL1/TCP/TST1.YOURDOMAIN.COM
  Comment: Alternative to channel table.

- *Name:* MQ_USER_ID
  Value: USER1
  Comment: Suitable RACF user-id.

ADDITIONAL NOTES COMPONENTS

To complete the Notes database set-up, some additional GUI components are needed. These are created by using the 'Design' feature of the Notes client when editing the Notes database – you need to create a new standard Notes database to contain the components, which is fairly simple to do.

- Create a form to edit the configuration document and give the form the name 'Agent Configuration'. On this form you need to

create fields to allow the entry of the following text variables that are read by the agent (the variables could also be hard-coded):

```
Description                              Variable Name (field name)
Name of configuration document           Form_Config_Name
MQSeries queue manager name              Form_Queue_Manager
MQSeries incoming queue name             Form_Input_Queue
MQSeries GET wait interval               Form_Get_Wait
GET retries after no msg available       Form_Get_Retries
Mail send type ('Send' or 'Create')      Form_Send_Type
Mail from/reply-to if blank in msg       Form_Mail_From
Dest to use if blank in message          Form_Missing_Dest
```

- Create another form with the name 'Memo' and make this the default form. Add some e-mail fields to the form, such as *subject*, *send-to*, and *body*, for displaying any saved e-mail documents.
- Create a view to allow the configuration document to be displayed and modified (the view must have the name 'Agent Configuration'). Define columns in the view to include some of the above fields. This view is also used by the agent to locate the configuration document. Set the following value in the view selection box:

```
SELECT Form = "Agent Configuration"
```

- Creating an 'all documents' view is useful if you intend to save documents in the database. If so, include some mail field names, such as 'send-to' and 'subject', in the view. Leave the view selection box blank. To be able to see replies (if any), de-select the view design option 'Show response documents in hierarchy'.
- Remember that right-clicking a document shown in a view allows all fields in the document to be displayed. This is very useful during development. Also note that Lotuscript can be run in debug mode, which allows stepping through the code as it executes (enable this from *File*, *Tools* on the Notes client interface). I suggest testing the code on your workstation using Notes client and MQ client before moving it to a Notes server. I prefer to continue using the MQ client even on the Notes server.

(Note the use of the continuation character, '➤', in the code below to indicate that one line of code maps to more than one line of print.)

THE MQSERIES TO E-MAIL AGENT LOTUSCRIPT CODE

```
'MQAGENT:

' Comments are denoted by an apostrophe

Option Declare
Uselsx "mqlsx"                  ' This call loads the MQLSX

' Warning - do not change the agent name without changing the
' MQ trigger data.

Public doc            As NotesDocument
Public MQFatalError   As Integer        ' Indicates a fatal error

' MQ Variables

Dim MQqms             As MQSession
Dim MQqmgr            As MQQueueManager
Dim MQq               As MQQueue
Dim MQgmo             As MQGetMessageOptions
Dim MQMsg             As MQMessage

Dim Queue_Manager     As NotesItem      ' Queue manager name
Dim Msgdata           As NotesItem      ' The data to put on the queue
Dim Input_Queue       As NotesItem      ' The name of the i/p queue
Dim Get_Wait_Time     As NotesItem      ' Wait time
Dim Get_Retries       As NotesItem      ' Retries if queue empty
Dim Send_Type         As NotesItem      ' Send or create
Dim Missing_Dest      As NotesItem      ' Destination if missing in msg
Dim Mail_From         As NotesItem      ' Mail origin if missing in msg

Dim BodyString        As String         ' String for e-mail body field
Dim OutString         As String         ' String for the message data
Dim ritem             As NotesRichTextItem

Sub Initialize

   Print "MQSeries Agent Started"

' This is the main routine for this agent
' Create some Notes session objects to use

   Dim session  As New NotesSession
   Dim db       As NotesDatabase
   Dim mqdefdoc As NotesDocument
   Dim view     As NotesView
   Dim Retries  As Long
   Dim Count    As Long

   Set db  = session.CurrentDatabase
```

```
   Set doc = New NotesDocument(db)

' Load script parameters from configuration view document (or
' hard-code)

   Set view          = db.GetView("Agent Configuration")
   Set mqdefdoc      = view.GetDocumentByKey("Default")
   Set Queue_Manager = mqdefdoc.GetFirstItem("Form_Queue_Manager")
   Set Input_Queue   = mqdefdoc.GetFirstItem("Form_Input_Queue")
   Set Get_Wait_Time = mqdefdoc.GetFirstItem("Form_Get_Wait")
   Set Get_Retries   = mqdefdoc.GetFirstItem("Form_Get_Retries")
   Set Send_Type     = mqdefdoc.GetFirstItem("Form_Send_Type")
   Set Missing_Dest  = mqdefdoc.GetFirstItem("Form_Missing_Dest")
   Set Mail_From     = mqdefdoc.GetFirstItem("Form_Mail_From")

   Print "Queue Name " + Queue_Manager.text + " " + Input_Queue.text

' Connect to MQ, open the input queue

   MQFatalError = False
   ConnectToMQ
   If MQFatalError = True Then Exit Sub

   OpenQForInput
   If MQFatalError = True Then Exit Sub

   Retries = Clng(Get_Retries.text) + 1

' Loop round until no message received after n retries

   Count = 0
   Do
      GetMsgFromQ
      If MQFatalError = True Then Exit Sub

' No message available, so decrement retry count (if count not 999)

      If OutString = "No Message" Then
         If Retries <> 999+1 Then
            Retries = Retries - 1
         End If

' If a message available, then send the e-mail and get next message

      Else
         If MQmsg.BackoutCount < 3 Then
            count = count + 1
            ParseTheMsg         ' Pick fields out of the message
            SendTheEmail        ' Send e-mail to mail system
            Print "Message processed data is " + OutString
```

```
            Set doc = New NotesDocument(db)    ' New doc for next msg
         Else
            Print "Message skipped, backout count high" + Outstring
         End If
         MQqmgr.Commit
         Retries = Clng(Get_Retries.text) + 1  ' Reset retry count
      End If

   Loop Until (Retries = 0)

   CloseQForInput
   DisconnectfromMQ

   Print "MQSeries agent ended, msgs read " &count

End Sub

Sub SendTheEmail

' Set any fixed values for the e-mail document

   doc.Form             = "Memo"    ' e-mail form name
   doc.DeliveryReport   = ""        ' e-mail delivery reporting flags

' Set up the rich text body field

   Call doc.RemoveItem("Body")
   Set ritem = doc.CreateRichTextItem("Body")
   Call ritem.AppendText(BodyString)

' Send the document to the mail system (no form attached)
' The document could be saved now in the database if desired
' using code like If doc.Save(True,True) Then …

    If Send_Type.text = "Create" Then
      SendEmailFrom         ' Direct mail document creation
   Else
      Call doc.Send(False)  ' Standard mail send (sets From = Agent id)
   End If

   Messagebox "Email Sent to " &doc.SendTo(0) + " " + doc.subject(0)

End Sub

Sub ParseTheMsg

' All message field offsets are coded in this subroutine; these can
' be changed to suit your message layout. Keep the body data as the
' last field.
'
```

```
' 001 - Send-to address - notes format (length 80 bytes)
' 081 - Copy-to address (80)
' 161 - Blind copy to address (80)
' 241 - Subject field (80)
' 321 - Reply-to address (80) - this also becomes the From address
' 401 - Body number of lines (NNNNN) (a 5 byte character number)
' 406 - Body lines x NNNNN each of 80 bytes - as required (last field)

   Dim BodyCount As Long
   Dim BodyOff   As Long
   Dim I         As Long
   Dim Flag      As Variant
   Dim Temp      As String

' Extract Notes e-mail fields from the message payload

   Temp = Trim(Mid(Outstring,001,80))
   If Temp <> "" Then
      doc.SendTo = Temp
   Else
      doc.SendTo = Missing_Dest.text   ' Default if blank
   End If

   doc.CopyTo      = Trim(Mid(Outstring,081,80))
   doc.BlindCopyTo = Trim(Mid(Outstring,161,80))

   Temp = Trim(Mid(Outstring,241,80))
   If Temp <> "" Then
      doc.Subject = Temp
   Else
      doc.Subject = "No Subject data"  ' Default if blank
   End If

   Temp = Trim(Mid(Outstring,321,80))
   If Temp <> "" Then
      doc.ReplyTo = Temp
   Else
      doc.ReplyTo = Mail_From.text     ' Default if blank
   End If

' Extract body of e-mail - nnnnn lines, as supplied

   Flag = False

   Temp = Trim(Mid(Outstring,401,5))   ' e-mail number of lines (NNNNN)
   If Isnumeric(Temp) = False Then
      Temp = "No Email Body length"
      Flag = True
   End If
```

```
   If Flag = True Then
      BodyString = Temp
      Exit Sub                          ' Note the exit here
   End If

   Temp = Trim(Mid(Outstring,401,5))    ' e-mail number of lines (NNNNN)
   BodyCount = Clng(Temp)
   BodyOff = 406                        ' Start of body lines
   Temp = ""

' Build temp, remove trailing spaces and add return char per line

   For I = 1 To BodyCount
      Temp = Temp & Rtrim(Mid(Outstring,BodyOff,80)) & Chr(13)
      BodyOff = BodyOff + 80
   Next

' Body data complete

   BodyString = Temp

End Sub

Sub SendEmailFrom

' This is an alternative way of sending the mail by creating a
' document directly in the mail box of the server. The from field can
' be set to any value, rather than always being the name of the
' id running the agent.

   Dim msess   As New NotesSession
   Dim mcurr   As NotesDatabase
   Dim mailbox As New NotesDatabase("","")
   Dim maildoc As NotesDocument
   Dim myitem  As Notesitem

' Get handle to mail.box on the current mail server of the user

   Set mcurr = msess.CurrentDatabase
   Call mailbox.Open(mcurr.server, "mail.box")

' Create a new document in the mail box of the sever - use ReplyTo
' field as From field

   Set maildoc  = mailbox.CreateDocument

   maildoc.Form             = doc.Form
   maildoc.From             = doc.ReplyTo
   maildoc.SendFrom         = doc.ReplyTo
   maildoc.SendTo           = doc.SendTo
```

```
   maildoc.Recipients       = doc.SendTo
   maildoc.Subject          = doc.Subject
   maildoc.DeliveryReport   = doc.DeliveryReport
   maildoc.DeliveryPriority = doc.DeliveryPriority

   Set myitem = doc.GetFirstItem( "Body" )
   Call myitem.CopyItemToDocument (maildoc, "" )

   maildoc.PostedDate       = Now()
   maildoc.ComposedDate     = Now()
   maildoc.DeliveredDate    = Now()
   maildoc.Principal        = doc.ReplyTo
   maildoc.EncryptOnSend    = False

   Call maildoc.Save( True, False )

End Sub

Public Sub ConnectToMQ

' This sub connects to MQSeries using the AccessQueueManager call
' Event handlers for the MQ session are registered

   If MQqms Is Nothing Then
      Set MQqms = New MQSession
' The event handlers for the MQ session need to be registered here
      On Event MQWARNING From MQqms Call WarningFromMQqms
      On Event MQERROR   From MQqms Call ErrorFromMQqms
   End If

' If not already connected to the queue manager, try to connect

   If MQqmgr Is Nothing Then
      ' Print "Accessing MQ Queue Manager"
      Set MQqmgr = MQqms.AccessQueueManager(Queue_Manager.Text)

' MQqmgr.ConnectionStatus is now TRUE if this call completed
' successfully. The error event handlers for the MQ queue manger
' need to be registered here.

      On Event MQWARNING From MQqmgr Call WarningFromMQqmgr
      On Event MQERROR   From MQqmgr Call ErrorFromMQqmgr
   End If

   If MQqmgr.ConnectionStatus = True Then   ' successfully connected
      Print "MQ connect successful"
   Else
      Delete MQqmgr                    ' Delete queue manager object
   End If
```

```
End Sub

Sub WarningFromMQqms (MQsess As MQSession)

' Called when an Mqwarning event is raised on the MQ Session

   Messagebox "Warning from MQ session, reason code "
   ➤  &MQSess.ReasonCode
   MQSess.ClearErrorCodes

End Sub

Sub ErrorFromMQqms (MQsess As MQSession)

' Called when an Mqerror is raised on the MQ session

   Messagebox "Error from MQ session, reason code " &MQsess.ReasonCode

   MQFatalError = True
   MQsess.ClearErrorCodes    ' Remember to clear the error codes

End Sub

Public Sub DisconnectFromMQ

   If MQqmgr Is Nothing Then
      Exit Sub
   End If

   MQqmgr.Disconnect      ' Disconnect from MQ; changes are committed
   Delete MQqmgr          ' Clean up by deleting the MQ objects
   Delete MQqms

End Sub

Public Sub OpenQForInput

   Dim OpenOptions As Long

' This subroutine is called to create the queue object
' Set up open options for input and create a new queue object

   OpenOptions = MQOO_INPUT_SHARED + MQOO_FAIL_IF_QUIESCING
   Set MQq = MQQmgr.AccessQueue(Input_Queue.Text,OpenOptions,"","","")

' Now register the event handlers for this new queue object

   On Event MQWARNING From MQq Call WarningFromMQq
   On Event MQERROR   From MQq Call ErrorFromMQq
```

```
End Sub

Public Sub GetMsgFromQ

' This subroutine gets the first message on the queue and reads
' the message data.

' Create a new message object and options

   Set MQMsg = New MQMessage
   Set MQgmo = New MQGetMessageOptions

' Set GetMessageOptions and GET msg (auto buffer management)

   MQgmo.Options = MQGMO_WAIT + MQGMO_FAIL_IF_QUIESCING +
   ➤  MQGMO_CONVERT + MQGMO_SYNCPOINT
   MQgmo.WaitInterval = Clng(Get_Wait_Time.text)

'    Print "MQ Get with wait time " + Get_Wait_Time.text
   MQq.Get MQMsg, MQgmo

' If everything went well, read the message in full, as a string

   If MQFatalError = False Then
      OutString = MQMsg.ReadString(MQMsg.MessageLength)
      If Len(OutString) = 0 Then
         OutString = "No Message"
      End If
   End If

End Sub

Public Sub CloseQForInput

' Make sure that every object is deleted

   If MQq Is Nothing Then
      Exit Sub
   End If

   Delete MQq
   Delete MQMsg
   Delete MQgmo

End Sub

Sub WarningFromMQqmgr (MQqmgr As MQQueueManager)

' Called when an Mqwarning event is raised on an MQ queue manager
' object
```

```
   Messagebox "Warning from MQ queue manager, reason code "
   ➤  &MQqmgr.ReasonCode

' Clear the error codes

   MQqmgr.ClearErrorCodes

End Sub

Sub ErrorFromMQqmgr (MQqmgr As MQQueueManager)

' Called when an Mqerror event is raised on an MQ queue manager object

   Messagebox "Error from MQ queue manager, reason code "
   ➤  &MQqmgr.ReasonCode

   MQqmgr.ClearErrorCodes  ' Clear the error codes
   MQFatalError = True     ' Mark this as a fatal error

End Sub

Sub WarningFromMQq(MQq As MQQueue)

' Called when an Mqwarning event is raised on an MQ queue object

   Messagebox "Warning from MQ queue, reason code" &MQq.ReasonCode

' Clear the error codes

   MQq.ClearErrorCodes

End Sub

Sub ErrorFromMQq (MQq As MQQueue)

' Called when an Mqerror event is raised on an MQ queue object

   If MQq.ReasonCode = MQRC_NO_MSG_AVAILABLE Then
      Print "Input queue empty - no msg available"
      Outstring = "No message"
   Else
      Messagebox "Error from MQ queue, reason code" &MQq.ReasonCode
      MQFatalError = True
   End If

' Clear the error codes

   MQq.ClearErrorCodes

End Sub
```

EXAMPLE AGENT TO WRITE A TEST MESSAGE TO MQSERIES

```
'WRITE TEST MESSAGE:

Option Declare
Uselsx "mqlsx"

' This agent is self-contained and does not read the configuration
' document, so it can be moved to other dbs. Changes to queue names
' etc are made here in init sub.

' Write a test message to MQ:

' Object variables

Public doc            As NotesDocument
Public MQFatalError   As Integer        ' Indicates a fatal error

' MQ variables

Dim MQqms             As MQSession
Dim MQqmgr            As MQQueueManager
Dim MQq               As MQQueue
Dim MQpmo             As MQPutMessageOptions
Dim MQMsg             As MQMessage

Dim QMgr_Name         As NotesItem      ' Queue manager name from form
Dim Msgdata           As NotesItem      ' The data to put on the queue
Dim Queue_name        As NotesItem      ' The name of the queue
Dim ConnectedQMgrName As Variant        ' Name of the queue manager

Dim OutString         As String         ' String containing MQMessage

Sub Initialize

   Dim session As New NotesSession
   Dim db  As NotesDatabase
   Set db  = session.CurrentDatabase
   Set doc = New NotesDocument(db)
   Set QMgr_Name  = New NotesItem(doc,"QMgr_Name", "TST1")
   Set Queue_name = New NotesItem(doc,"Queue_name",
   ➤  "TEST.EMAIL.QUEUENAME")

' Format a test message

   Dim BodyCount As Long
   Dim BodyOff As Long
   Dim I As Long

   Outstring = Space(1000)
```

```
' Set up some fields

   Mid(Outstring,001,80) = " "     ' Send to (blank to pick up default)

   Mid(Outstring,081,80) = " "     ' Copy to
   Mid(Outstring,161,80) = " "     ' BCC

   Mid(Outstring,241,80) = "test message"   ' Subject
   Mid(Outstring,321,80) = " "              ' Reply to (Blank gets
                                            ' default)

   Mid(Outstring,401,5) = "00010"

   BodyCount = 10
   BodyOff = 406

   For I = 1 To BodyCount
      Mid(Outstring,BodyOff,80) = "Test message line number" &I
      BodyOff = BodyOff + 80
   Next

' Connect to MQ, open the queue, PUT message, and close

   MQFatalError = False
   If MQFatalError = False Then ConnectToMQ
   If MQFatalError = False Then OpenQForOutput
   If MQFatalError = False Then PutMsgOnQ
   If MQFatalError = False Then MQqmgr.commit
   If MQFatalError = False Then CloseQForOutput
   If MQFatalError = False Then DisconnectFromMQ

   Messagebox "Message Data is " + OutString

End Sub

Public Sub OpenQForOutput

' Create the queue object and open on the real MQ queue

   Dim OpenOptions As Long

' Set the open options
   OpenOptions = MQOO_OUTPUT + MQOO_FAIL_IF_QUIESCING
   Set MQq = MQQmgr.AccessQueue(Queue_name.Text,OpenOptions,"","","")
   On Event MQWARNING From MQq Call WarningFromMQq
   On Event MQERROR From MQq Call ErrorFromMQq

End Sub

Public Sub PutMsgOnQ
```

```
' Place on the queue as a string.

   Set Msgdata  = New NotesItem(doc,"Msgdata", Outstring)

' Create a new message object and the PUT message options

   Set MQMsg = New MQMessage
   Set MQpmo = New MQPutMessageOptions      ' Set the PutMessageOptions
   MQpmo.Options = MQPMO_FAIL_IF_QUIESCING + MQPMO_NO_SYNCPOINT

' Write as a string into our new message

   MQMsg.writestring (Msgdata.Text)

' Before putting message on the queue, set more options

   MQMsg.MessageType = MQMT_DATAGRAM
   MQMsg.Format = MQFMT_STRING

' Now we can put the message on the output queue

   If MQMsg.messagelength <> 0 Then
      MQq.Put MQMsg, MQpmo
   End If
   Print "MQ Message put complete"

End Sub

' Copy the code for the subroutines below from the agent example
' ************************************************************

Sub WarningFromMQqmgr (MQqmgr As MQQueueManager)

Sub ErrorFromMQqmgr (MQqmgr As MQQueueManager)

Sub WarningFromMQq(MQq As MQQueue)

Sub ErrorFromMQq (MQq As MQQueue)

Public Sub ConnectToMQ

Sub WarningFromMQqms (MQsess As MQSession)

Sub ErrorFromMQqms (MQsess As MQSession)

Public Sub DisconnectFromMQ
```

Administering MQSeries for MVS/ESA from Unix

INTRODUCTION

In this article I discuss how to administer MQSeries for MVS/ESA remotely from a Unix system using MQSeries for HP-UX Version 5.0. In this context, administration is restricted to MQ objects, such as queues and processes. The article describes key concepts used in remote administration and then shows you how to implement them using the facilities provided by MQSeries.

The set-up discussed here makes life easier for administrators who are more familiar with Unix than MVS and would like to control MQSeries for MVS/ESA from their Unix environment. Another benefit is to establish a single point of control. I find that the availability of remote administration makes every aspect of defining and changing queues and other MQSeries objects easier, as objects can be created and managed, and their version controlled, using non-mainframe repositories. I also find this method is more intuitive than using a CSQUTIL-like interface.

In this article I also discuss some common problems that face those using remote administration. Some of these problems can be addressed by customizing the administration application, and we discuss a number of possible solutions.

KEY CONCEPTS IN REMOTE ADMINISTRATION

In this section I summarize some of the concepts necessary for remote administration: command formats, command messages, command queues, and command servers.

Command formats

We can administer MQSeries objects using either *mqsc* or PCF (Programmable Command Format) commands. With MQSeries for HP-UX V5.0, it's possible interactively to create, change, and delete MQ objects using *mqsc* commands run from the **runmqsc** utility that's provided with MQSeries.

Command messages and command queues

For remote administration, an MQSeries queue manager is used to send commands to another, possibly remote, queue manager. Commands are sent as messages, referred to as 'command messages'. In general, command messages contain either an *mqsc* or PCF command (though command messages for MQSeries for MVS/ESA version 1.2 may contain only *mqsc* commands). While the format of the message generally determines the destination queue, messages containing PCF commands for MQSeries for Unix always use *SYSTEM.ADMIN.COMMAND.QUEUE* as the destination queue and those containing *mqsc* commands always use *SYSTEM.COMMAND.INPUT*.

Command server

A command server processes command messages. A command server is included in the MQSeries product. MQSeries Version 5 products use **strmqcsv** to start the command server, while MQSeries for MVS/ESA Version 1.2 uses the queue manager command below to start the command processor.

```
+cpf START CMDSERV
```

Administration application

Any authorized application can put command messages on the command queue. The application can be either local or remote, though remote applications require intersystem communication support. The administration application is usually client/server-based, with the administration application acting as the client and the command server acting as the server. Command messages are just requests to the command server – the administration application specifies the reply-to-queue in the request, the command server processes the requests and then it puts the reply on the reply-to-queue specified. The administration application then collects replies to *mqsc* requests. In the case of MQSeries for HP-UX Version 5, *SYSTEM.MQSC.REPLY.QUEUE* is defined as the model queue that can be used to receive replies.

Setting up remote administration

The MQSeries set-up required for remote administration is shown in Figure 1. Here, *HPMQ* is the queue manager on an HP-UX system and *MVMQ* is the queue manager on an MVS/ESA host. Queue managers are known to each other using alias queue manager definitions. The alias is simply a remote queue definition that identifies the transmission queue to be used during intersystem communication. We match the name of the alias queue manager with the name of an actual partner queue manager. The remote queue *MVMQ*, defined on queue manager *HPMQ*, is an alias for the *MVMQ* queue manager. Similarly the remote queue *HPMQ*, defined on the MVS/ESA queue manager, is an alias for the HP system's queue manager. The transmission queue requires a definition, as do the sender and receiver channels. These definitions are shown in the following sections, and the names of machines, queue managers, port numbers, and so on need to be customized to your site's requirements.

MQSeries definitions for HP-UX

The following script can be used on the MQSeries queue manager *HPMQ* to provide the definitions required for the system shown in Figure 1.

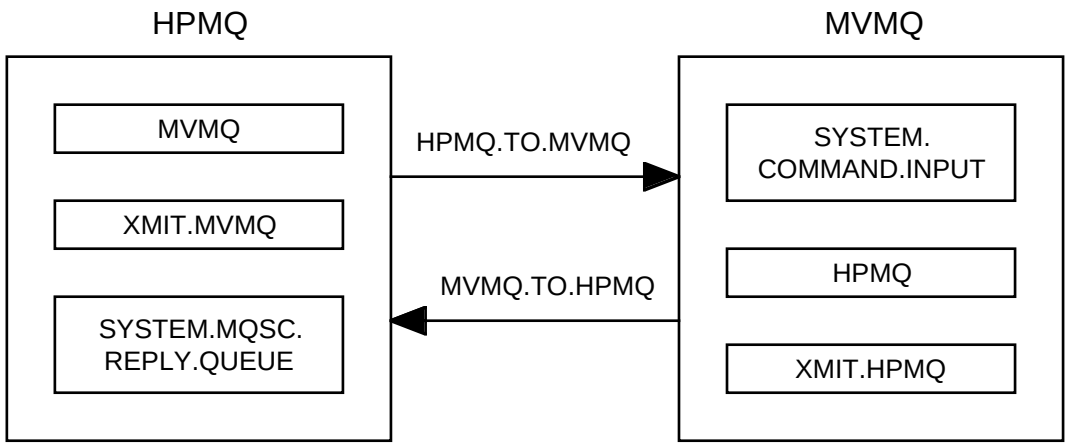

Figure 1: Set-up for remote administration

HPMQ SCRIPT

```
DEFINE QLOCAL ('XMITQ.MVMQ') +
       LIKE(SYSTEM.DEFAULT.LOCAL.QUEUE) +
       REPLACE SHARE USAGE(XMITQ) +
       TRIGGER +
       TRIGTYPE(FIRST) +
       TRIGDATA ('HPMQ.TO.MVMQ') +
       PROCESS ( P1 )
       INITQ ('SYSTEM.CHANNEL.INITQ') +
       DESCR('XMIT Queue for connection to MVS MQ ')

DEFINE QREMOTE ('MVMQ') +
       RQMNAME ( 'MVMQ') +
       REPLACE +
       XMITQ('XMITQ.MVMQ') +
       LIKE(SYSTEM.DEFAULT.REMOTE.QUEUE) +
       DESCR('Alias Qmanager for MVMQ ')

DEFINE CHANNEL('HPMQ.TO.MVMQ') CHLTYPE(SDR) +
       TRPTYPE(TCP) CONNAME('host(1414)') +
       LIKE (SYSTEM.DEF.SENDER ) +
       XMITQ('XMITQ.MVMQ') +
       CONVERT(YES) +
       REPLACE DESCR('Sender  channel to MVMQ ')

DEFINE CHANNEL('MVMQ.TO.HPMQ') CHLTYPE(RCVR) +
       LIKE (SYSTEM.DEF.RECEIVER ) +
       REPLACE DESCR(' channel to receive from MVMQ')

DEFINE PROCESS (P1) USERDATA ('HPMQ.TO.MVMQ')
```

Make sure that the files */etc/inetd.conf* and */etc/services* include the following entries (the continuation character, '➤', denotes a formatting line break that should not appear in the actual entry):

- */etc/inetd.conf*:

```
MQSeries stream tcp nowait mqm /opt/mqm/bin/amqcrsta
➤  amqcrsta -m HPMQ
```

- */etc/services*:

```
MQSeries    1414/tcp
```

After you add these entries, kill and then start the **inetd** daemon.

The most common error in this type of set-up is to forget to include *CONVERT(YES)* in the sender channel definition.

MQSeries definition for MVS/ESA

On the MVS/ESA queue manager (*MVMQ*) customize *CSQ4DISX* or use ISPF panels to implement the following definitions.

MVMQ DEFINITIONS

```
DEFINE QLOCAL( 'XMITQ.HPMQ' ) +

* Common queue attributes
         DESCR( 'Transmission queue for HPMQ' ) +
         PUT( ENABLED ) +
         DEFPRTY( 5 ) +
         DEFPSIST( YES ) +

* Local queue attributes
         GET( ENABLED ) +
         SHARE +
         DEFSOPT( EXCL ) +
         MSGDLVSQ( FIFO ) +
         RETINTVL( 999999999 ) +
         MAXDEPTH( 10000 ) +
         MAXMSGL( 4194304 ) +
         NOHARDENBO +
         BOTHRESH( 0 ) +
         BOQNAME( ' ' ) +
         STGCLASS( 'REMOTE' ) +
         USAGE( XMITQ ) +
         INDXTYPE( NONE ) +

* Event control attributes
         QDPMAXEV( ENABLED ) +
         QDPHIEV( DISABLED ) +
         QDEPTHHI( 80 ) +
         QDPLOEV( DISABLED ) +
         QDEPTHLO( 40 ) +
         QSVCIEV( NONE ) +
         QSVCINT( 999999999 ) +

* Trigger attributes
         TRIGGER +
         TRIGTYPE( FIRST ) +
         TRIGMPRI( 0 ) +
         TRIGDPTH( 1 ) +
         TRIGDATA( ' ' ) +
         PROCESS( 'HPMQ.SEND.PROCESS' ) +
         INITQ( 'SYSTEM.CHANNEL.INITQ' )
*
******
```

```
DEFINE PROCESS( 'HPMQ.SEND.PROCESS' ) +

* Process attributes
         DESCR( 'Process for sending messages to HPMQ' ) +
         APPLTYPE( MVS ) +
         APPLICID( 'CSQX START' ) +
         USERDATA( 'MVMQ.TO.HPMQ' ) +
         ENVRDATA( ' ' )
*
******
DEFINE CHANNEL( 'MVMQ.TO.HPMQ') +
         CHLTYPE( SDR ) +

* Sender channel attributes
         DESCR( 'Channel for sending messages to HPMQ' ) +
         TRPTYPE( TCP ) +
         XMITQ( 'XMITQ.HPMQ' ) +
         MCAUSER( ' ' ) +
         BATCHSZ( 50 ) +
         DISCINT( 6000 )      BATCHINT( 0 ) +
         SHORTRTY( 10 )       SHORTTMR( 60 ) +
         LONGRTY( 999999999 ) LONGTMR( 1200 ) +
         SCYEXIT( ' ' )       SCYDATA( ' ' ) +
         MSGEXIT( ' ' )       MSGDATA( ' ' ) +
         SENDEXIT( ' ' )      SENDDATA( ' ' ) +
         RCVEXIT( ' ' )       RCVDATA( ' ' ) +
         SEQWRAP( 999999999 ) +
         CONVERT( YES ) +
         NPMSPEED( FAST )     HBINT( 300 ) +
         MAXMSGL( 4194304 ) +

* Connection name attribute
         CONNAME('hp(1414)')

******
DEFINE QREMOTE( 'HPMQ' ) +

* Common queue attributes
*        DESCR( 'Queue for accessing TARGET.QUEUE on HPMQ' ) +
         DESCR( 'Alias Queue Manager for HPMQ' ) +
         PUT( ENABLED ) +
         DEFPSIST( YES ) +
         DEFPRTY( 9 ) +
         RQMNAME( HPMQ ) +
         XMITQ( 'XMITQ.HPMQ' )

DEFINE CHANNEL( 'HPMQ.TO.MVMQ') +
         CHLTYPE( RCVR ) +

* Receiver channel attributes
```

```
DESCR( 'Channel for receiving messages from HPMQ' ) +
TRPTYPE( TCP ) +
BATCHSZ( 50 ) +
SCYEXIT( ' ' )       SCYDATA( ' ' ) +
MSGEXIT( ' ' )       MSGDATA( ' ' ) +
SENDEXIT( ' ' )      SENDDATA( ' ' ) +
RCVEXIT( ' ' )       RCVDATA( ' ' ) +
MCAUSER( ' ' ) +
PUTAUT( DEF ) +
SEQWRAP( 999999999 ) +
NPMSPEED( FAST )     HBINT( 300 ) +
MAXMSGL( 4194304 )
```

Make sure that the command server starts whenever queue manager *MVMQ* is started.

OPERATION OF REMOTE ADMINISTRATION

1 Start the channels:

On the HP system, use the command **runmqsc HPMQ** to start channels using:

```
start channel ( HPMQ.TO.MVMQ )
start channel ( MVMQ.TO.HPMQ )
```

On the MVS/ESA host, use ISPF panels first to start the channel initiator and then the channels. The set-up is suitable for triggered channels. If the channel initiator is started with the queue manager, the channels are triggered when messages arrive on the transmission queue.

2 Verify that the sender and receiver channels' status is *RUNNING*. On the HP system, use the **runmqsc HPMQ** command to verify the status using:

```
display chstatus (*)
```

On the MVS/ESA host, you can use ISPF panels to determine the channels' status.

3 Use the **runmqsc** command to send messages to the remote queue manager. This command has two flags that are required for the remote administration of MQSeries for MVS/ESA: the **–w** flag puts the **runmqsc** command in indirect mode, which means

that *mqsc* commands passed to **runmqsc** are run on the remote queue manager. The replies/reports are returned to the local queue manager. The **–w** flag requires a 'wait time' in seconds. The **–x** flag specifies that the target queue manager is running under MVS/ESA.

A sample session for remote administration is captured as follows:

```
hp$runmqsc -w 30 -x MVMQ
84H2002, 5765-B74 (C) Copyright IBM Corp. 1994, 1997
ALL RIGHTS RESERVED.
Starting MQSeries Commands.

display qmgr
     1 : display qmgr
CSQN205I   COUNT=3, RETURN=00000000, REASON=00000000
CSQM409I   QMNAME(MVMQ)
CSQ9022I ]MVM CSQMDRTS ' DISPLAY QMGR' NORMAL COMPLETION
display qmgr all
     2 : display qmgr all
CSQN205I   COUNT=3, RETURN=00000000, REASON=00000000
CSQM409I   QMNAME(MVMQ) DESCR(MVMQ, IBM MQSeries for MVS/ESA - V1.2)
PLATFORM(MVS) CPILEVEL(100) CMDLEVEL(120) CCSID(500) MAXPRTY(9)
MAXMSGL(4194304) SYNCPT(AVAILABLE) COMMANDQ(SYSTEM.COMMAND.INPUT)
DEADQ(MVMQ.DEAD.QUEUE) TRIGINT(999999999) MAXHANDS(256)
AUTHOREV(DISABLED) INHIBTEV(DISABLED) LOCALEV(DISABLED)
REMOTEEV(DISABLED) STRSTPEV(ENABLED) PERFMEV(DISABLED)
DEFXMITQ(MVMQ.DEFXMIT.QUEUE)
CSQ9022I ]MVM CSQMDRTS ' DISPLAY QMGR' NORMAL COMPLETION
end
     3 : end
2 MQSC commands read.
2 command responses received.
```

The **runmqsc** session terminates when you issue the command **end** (this is true of any **runmqsc** session). Every reply contains at least three messages: the first contains information about the number of messages being sent, the return code, and reason code for the request. The last message carries information about the status of the command being executed, and the remaining messages contain data specific to the results of the request. The **runmqsc** command does not format the output data, making it somewhat hard to interpret.

If any problems occur, check the following log files:

```
/var/mqm/errors/AMQERR01.LOG
/var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG
```

```
/var/mqm/qmgrs/HPMQ/errors/AMQERR01.LOG
```

LIMITATIONS

The use of **runmqsc** in indirect mode has one severe limitation – the local queue manager used by this command must be the default queue manager. This means that, in Figure 1, *HPMQ* must be the default queue manager, as it's the local queue manager used by **runmqsc**.

Additionally, remote administration cannot boot itself. In other words, you cannot start or stop the MVS/ESA queue managers remotely. You can stop the channels, but you cannot start the channels if the channels are stopped.

One of the most irritating problems with MQSeries for MVS/ESA V1.2 is that the channel initiator has to be stopped and re-started when TCP/IP is shut down and restarted. Remote administration cannot be used to fix this problem.

CUSTOM ADMINISTRATION APPLICATION

Given the limitation that **runmqsc** works only with the default queue manager, you may be forced to write your own administration application. Writing administration applications is explained here with reference to some sections in Chapter 14 of MQSeries' *System Management Guide*.

As described earlier, the remote administration application is essentially a client/server application. Requests are put on the command queue (*SYSTEM.COMMAND.QUEUE*) of the remote queue manager, with the reply-to-queue specified as a dynamic queue using the *SYSTEM.MQSC.REPLY.QUEUE* model. The message buffer should contain *mqsc* commands.

The message buffer size should be set to 32 KB, as neither the command nor reply should exceed 32 KB. The additional requirement is that we should be able to specify the local queue manager.

A simple remote administration application is listed below. In this program, the output is formatted so that the parameter and values being displayed are easy to read.

MYCONTROL.C

```
/*************************************************************************/
/*                                                                       */
/* mycontrol.c : A simple MQ remote administration application.          */
/*                                                                       */
/* The application accepts two parameters, the second of which           */
/* is optional:                                                          */
/*   - The name of remote QManager (MVS/ESA)                             */
/*   - The name of local queue manager that is able to communicate       */
/*     with the remote queue manager.                                    */
/*                                                                       */
/* If second parameter is not specified, the default queue manager       */
/* is used.                                                              */
/*                                                                       */
/* The application accepts mqsc commands from stdin. Each command        */
/* should be on a line by itself, and you can issue any number of        */
/* commands.                                                             */
/*                                                                       */
/* To terminate command input, enter an empty line. If stdin is          */
/* redirected from a file, EOF terminates input.                         */
/*                                                                       */
/*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
/* includes for MQI */
#include <cmqc.h>

#define MAX_BUFFER_LEN 32000

FILE  *fp ;
MQCHAR buffer  [MAX_BUFFER_LEN ] ;
MQLONG buflen ;

MQOD     od = {MQOD_DEFAULT};    /* Object descriptor             */
MQOD    odr = {MQOD_DEFAULT};    /* Object descriptor for reply   */
MQMD     md = {MQMD_DEFAULT};    /* Message descriptor            */
MQGMO   gmo = {MQGMO_DEFAULT};   /* GET message options           */
MQPMO   pmo = {MQPMO_DEFAULT};   /* PUT message options           */
MQHCONN  Hcon;                   /* Connection handle             */
MQHOBJ   Hobj;                   /* Object handle for server      */
MQHOBJ   Hreply;                 /* Object handle for reply       */
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   replylen;               /* Reply length                  */
MQCHAR48 replyQ;                 /* Reply queue name              */

char     QMName[49];             /* Queue manager name            */
char     RemoteQMName[49];       /* Remote queue manager name     */
char     parameter [256];
char     value [1024];

int main(int argc, char **argv)
{
   printf("mycontrol  start\n");
   if (argc < 2)
   {
     printf("Required parameter missing - queue name\n");
     exit(99);
   }
   strcpy (RemoteQMName, argv[1]);

   /*****************************************************************/
   /*                                                               */
   /*                     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               */
          &CReason);              /* Reason code                   */

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

   strncpy(od.ObjectName, "SYSTEM.COMMAND.INPUT" , MQ_Q_NAME_LENGTH);
   strncpy(od.ObjectQMgrName, RemoteQMName , MQ_Q_NAME_LENGTH);

#ifdef DEBUG
   printf("remote queue is %s\n", od.ObjectName);
   printf("remote queue manager is %s\n", od.ObjectQMgrName);
#endif

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

```
/*          Open the request message queue for output          */
/*                                                             */
/*************************************************************/
O_options = MQOO_OUTPUT         /* Open queue for output       */
     + 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                 */
```

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

Ashish Joshi
Consultant (USA)

MQSeries and NT screen resolution

Of all the things you'd expect not to affect middleware, screen resolution would probably be one of them. However, I experienced a problem installing MQSeries Server on an NT system with a display of less than 800 by 600 pixels. Every time I tried to install, I got the following message: "Server component cannot be installed without 800x600 or better screen resolution".

A lengthy search on the Web revealed that I'm not alone – while the 800x600 requirement is documented in the *readme.txt* on the CD, there is a workaround. To stop the installation program checking screen resolution, launch the installation using the following command:

```
setup -noprereqnonvga
```

System Administrator (UK)

MQ news

BEA has announced a range of new eLink products aimed at integrating enterprise processes and applications both within businesses and across the Web. eLink products are supported by a range of packaged adapters that provide interfaces between eLink and ERP, CRM, mainframe-based applications, and other technologies. These include a new adapter for MQSeries, which integrates third-party applications with MQSeries, adding to an updated adapter for mainframe TCP, which integrates OS/390 CICS and IMS applications running over TCP/IP.

Adapters for MQSeries are currently available on HP-UX. Availability on AIX, Solaris, and Windows NT is expected imminently. Prices are available on request from the vendor.

For further information contact:
BEA Systems Inc, 385 Moffet Park Drive, Sunnyvale, CA 94089
Tel: +1 408 743 4000
Fax: +1 408 734 9234
Web: http://www.beasys.com

BEA Systems Ltd, Windsor Court, Kingsmead Business Park, Frederick Place, London Road, High Wycombe, Bucks HP11 1JV, UK
Tel: +44 1494 559500
Fax: +44 1494 452202

* * *

New Era of Networks has announced NEON e-Business Adapter Development Kit, or e-ADK, which is a set of tools and libraries for developing adapters to interface applications with NEON e-Business Integration Servers. The kit includes a complete architectural framework for building a NEON adapter. Users can configure the kit to work with any supported transport, such as IBM MQSeries, Microsoft MSMQ, or flat files, and developers don't need to know details of low-level programming for each transport supported by the e-ADK.

It's out now for NT 4.0 and Solaris 2.6 and 7. Availability on AIX and HP/UX is planned for Q1 2000. Prices were not announced.

For further information contact:
NEON, 7400 East Orchard Road, Englewood, CO 80111, USA
Tel: +1 303 694 3933
Fax: +1 303 694 3885
Web: http://www.neonsoft.com

New Era of Networks Ltd, Aldermary House, 15 Queen Street, London EC4N 1TX, UK
Tel: + 44 171 329 4669
Fax:+ 44 171 329 4567

* * *

IBM has announced that Version 1.6 of its Business Integration Suite for Windows NT is to include MQSeries V5.1. Other components include: eNetwork Communications Server 6.02 (including Host On-Demand Entry V3.0), DB2 Universal Database Workgroup Edition 6.1 (including NetData V2.0.5 and DB2 Extenders), Lotus Domino 5.0, WebSphere Application Server Standard Edition V2.03 (including IBM HTTP Server V1.3), and SecureWay Directory V3.1. The Integration suite is priced at US$6,500.