18

MQ update

December 2000

In this issue

MQ Update

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
Post Office Box 350100
Westminster CO 80035-0100, USA
Telephone: +1 303 410 9344
Fax: +1 303 438 0290

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

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.

‘Queue Manager Not Active’ messages on S/390

Some new Service Level Agreements (SLAs) are being drawn up in our organization that stipulate certain requirements with regards to MQSeries Queue Manager and channel availability. Meeting our SLA also requires us to produce statistics to show whether the stipulated levels of availability were attained. The obvious thing to use to implement this seemed to be event messages. By enabling queue manager start/stop instrumentation events, we should get messages on the *SYSTEM.ADMIN.QMGR.EVENT* queue from which we can extract the times during which the queue manager was available. Channel event messages cannot be disabled, so we should also get messages about when channels are started and stopped, as long as the *SYSTEM.ADMIN.CHANNEL.EVENT* queue exists. Now we just need to write some event monitors to process these messages and extract the information we need to produce the statistics required by management.

However, according to the *MQSeries Programmable System Management* manual, the *Queue Manager Not Active* event is not produced by MQSeries for OS/390. So our problem was to find a way to determine when the queue manager stops.

The solution we chose was to make some simple changes to an existing program – the MQSeries SupportPac’s *MA12: MQSeries for MVS/ESA Batch Trigger Monitor*, which can be downloaded from:

```
http://www-4.ibm.com/software/ts/mqseries/txppacs/
```

The trigger monitor is a COBOL program, **CKTIBAT2**, which runs continuously whenever MQSeries is active, waiting for messages on an initiation queue. For each message that arrives, it submits a JCL stream to the JES internal reader. The changes we made were to include the *MQGMO-FAIL-IF-QUIESCING* option on the *MQGET* call and then to test for Reason Code *MQRC-Q-MGR-QUIESCING* if the *MQGET* fails. If the queue manager is quiescing, we put a *Queue Manager Not Active* event message on the *SYSTEM.ADMIN.QMGR.EVENT* queue.

Some limitations of this approach:

- The obvious shortcoming is that the event message will not be created in a forced shutdown. However, our procedures dictate that a normal shutdown should always be attempted first, in which case the *Queue Manager Not Active* event message will be created.
- The program changes described below reserve only four characters for the Queue Manager's name. That being said, it's worth remembering that OS/390 also imposes this restriction on Queue Manager names.

To implement the changes, **CKTIBAT2** was altered as detailed below. Where existing code has been included, the new lines have been highlighted for clarity.

The first thing to do is add structures to *WORKING-STORAGE* to accommodate the options of the message being *MQPUT* and the event message:

CODE SEGMENT #1

```
        01  W05-MQM-PUT-MESSAGE-OPTIONS.
            COPY CMQPMOV.
       *
       *    Structure for queue manager not active event message.
       *
        01  W05-MQM-EVENT-MESSAGE.
            COPY CMQCFHV.
            COPY CMQCFSTV.
       **    String - Queue manager name limited to 4 chars on MVS.
            15 MQCFST-STRING         PIC X(4) VALUE SPACES.
            COPY CMQCFINV.
```

Also add a structure to *WORKING-STORAGE* to print a message when the queue manager shuts down:

CODE SEGMENT #2

```
01  W04-MESSAGE-5.
            05  W04-HOURS-5          PIC X(2).
            05  FILLER               PIC X VALUE ':'.
            05  W04-MINUTES-5        PIC X(2).
```

```
           05  FILLER              PIC X VALUE ':'.
           05  W04-SECONDS-5       PIC X(2).
           05  FILLER              PIC X(3)  VALUE SPACES.
           05  FILLER              PIC X(42) VALUE
                    '********** Queue manager quiescing.'.
           05  FILLER              PIC X(79) VALUE SPACES.
```

Include the constants required for event messages (you'll find this section of code at line 216 in the original source file):

CODE SEGMENT #3

```
      *
      *    Copy file of constants (for filling in the control blocks)
      *    and return codes (for testing the result of a call)
      *
       01  W05-MQM-CONSTANTS.
           COPY CMQV.
           COPY CMQCFV.
```

Add *MQGMO-FAIL-IF-QUIESCING* before the *MQGET* call (at line 428):

CODE SEGMENT #4

```
           MOVE MQGMO-WAIT TO MQGMO-OPTIONS.
           ADD MQGMO-NO-SYNCPOINT TO MQGMO-OPTIONS.
           ADD MQGMO-FAIL-IF-QUIESCING TO MQGMO-OPTIONS.
```

Test for *MQRC-Q-MGR-QUIESCING* after the *MQGET* loop (at line 478):

CODE SEGMENT #5

```
           IF NOT W03-COMPCODE = MQCC-OK
           THEN
              IF W03-REASON = MQRC-Q-MGR-QUIESCING
              THEN
                 PERFORM PUT-EVENT-MESSAGE
              ELSE
                 MOVE 'GET'          TO W04-MSG4-TYPE
                 MOVE W03-COMPCODE   TO W04-MSG4-COMPCODE
                 MOVE W03-REASON     TO W04-MSG4-REASON
                 MOVE W04-MESSAGE-4  TO PRINT-DATA
              END-IF
           END-IF.
```

Lastly, add the section *PUT-EVENT-MESSAGE*:

CODE SEGMENT #6

```
 PUT-EVENT-MESSAGE SECTION.
*  ------------------------------------------------------------
*    This section is called when the queue manager is quiescing.
*    It puts a queue manager not active event message on the
*    SYSTEM.ADMIN.QMGR.EVENT queue.
*  ------------------------------------------------------------
*
      ACCEPT W00-TIME FROM TIME.
      MOVE W00-HOURS TO W04-HOURS-5.
      MOVE W00-MINUTES TO W04-MINUTES-5.
      MOVE W00-SECONDS TO W04-SECONDS-5.
      MOVE W04-MESSAGE-5 TO PRINT-DATA.
*
      MOVE 'SYSTEM.ADMIN.QMGR.EVENT' TO MQOD-OBJECTNAME.
*
      MOVE MQRO-NONE               TO MQMD-REPORT.
      MOVE MQMT-DATAGRAM           TO MQMD-MSGTYPE.
      MOVE MQEI-UNLIMITED          TO MQMD-EXPIRY.
      MOVE MQFB-NONE               TO MQMD-FEEDBACK.
      MOVE MQENC-NATIVE            TO MQMD-ENCODING.
      MOVE MQCCSI-Q-MGR            TO MQMD-CODEDCHARSETID.
      MOVE MQFMT-EVENT             TO MQMD-FORMAT.
      MOVE MQPRI-PRIORITY-AS-Q-DEF TO MQMD-PRIORITY.
      MOVE MQPER-PERSISTENT        TO MQMD-PERSISTENCE.
      MOVE MQMI-NONE               TO MQMD-MSGID.
      MOVE MQCI-NONE               TO MQMD-CORRELID.
      MOVE SPACES                  TO MQMD-REPLYTOQ.
      MOVE SPACES                  TO MQMD-REPLYTOQMGR.
*
      COMPUTE W03-DATA-LENGTH = MQCFH-STRUC-LENGTH +
                                MQCFST-STRUC-LENGTH-FIXED + 4 +
                                MQCFIN-STRUC-LENGTH.
*
      MOVE MQCFT-EVENT             TO MQCFH-TYPE.
      MOVE MQCMD-Q-MGR-EVENT       TO MQCFH-COMMAND.
      MOVE MQCC-WARNING            TO MQCFH-COMPCODE.
      MOVE MQRC-Q-MGR-NOT-ACTIVE   TO MQCFH-REASON.
      MOVE 2                       TO MQCFH-PARAMETERCOUNT.
      COMPUTE MQCFST-STRUCLENGTH = MQCFST-STRUC-LENGTH-FIXED + 4.
      MOVE MQCA-Q-MGR-NAME         TO MQCFST-PARAMETER.
      MOVE 4                       TO MQCFST-STRINGLENGTH.
      MOVE W02-MQM                 TO MQCFST-STRING.
      MOVE MQIACF-REASON-QUALIFIER TO MQCFIN-PARAMETER.
      MOVE MQRQ-Q-MGR-QUIESCING    TO MQCFIN-VALUE.
*
*     Put the message
```

```
      *
            CALL 'MQPUT1' USING W03-HCONN
                                MQOD
                                MQMD
                                MQPMO
                                W03-DATA-LENGTH
                                W05-MQM-EVENT-MESSAGE
                                W03-COMPCODE
                                W03-REASON.
      *
           IF NOT W03-COMPCODE = MQCC-OK
           THEN
              WRITE PRINT-REC
              MOVE 'PUT1'         TO W04-MSG4-TYPE
              MOVE W03-COMPCODE  TO W04-MSG4-COMPCODE
              MOVE W03-REASON    TO W04-MSG4-REASON
              MOVE W04-MESSAGE-4 TO PRINT-DATA
           END-IF.
      *
       PUT-EVENT-MESSAGE-END.
           EXIT.
```

Eric Judd
Technical Consultant
Metropolitan Life (South Africa)

Microsoft MSMQ to MQSeries Bridge (part 2)

This month's instalment concludes this article on the MSMQ to MQSeries Bridge (the first instalment appeared in last month's issue of *MQ Update*).

To send a message from MSMQ to MQSeries, you define an MSMQ foreign computer that represents the MQSeries Queue Manager (assuming that the MQSeries destination queue already exists). The following process can be used for sending messages:

- Either create the foreign queue using the MSMQ Explorer or ensure that your MSMQ application issues an MSMQ *MQCreateQueue()* API call that creates a foreign queue on the foreign computer that represents the MQSeries destination queue.

- To open the foreign queue, the application issues an MSMQ *MQOpenQueue()* API call.
- To send a message to the foreign queue, the application issues an MSMQ *MQSendMessage()* API call. MSMQ transmits the message and stores it temporarily on an MSMQ Connector queue.
- MSMQ-MQSeries Bridge takes the message from the MSMQ Connector queue and converts the message properties to the MQSeries message structure. MSMQ-MQSeries Bridge transmits the message to the MQSeries destination queue.
- An MQSeries application issues an MQSeries *MQGET()* API call to receive the message from the MQSeries queue.

MSMQ processes the message from the initial *MQSendMessage()* call until it is placed on the Connector queue. MSMQ-MQSeries Bridge converts and transmits the message to MQSeries, which then handles the transmission from that point on.

While there are a few differences in the reverse process of sending a message from MQSeries to MSMQ, the process is nevertheless essentially the inverse of that of sending messages from MSMQ to MQSeries. You define MQSeries aliases, transmission queues, and channels for the MSMQ destination queue and/or the MSMQ Server (assuming the MSMQ destination queue already exists). A typical procedure for sending a message from MQSeries to MSMQ is:

- An MQSeries application issues an *MQOPEN()* API call to a remote queue that represents the MSMQ destination queue.
- The MQSeries application issues an *MQPUT()* API call to send a message to the remote queue. MQSeries transmits the message and stores it temporarily on an MQSeries transmission queue located at the MQSeries Queue Manager.
- MSMQ-MQSeries Bridge takes the message from the transmission queue and converts the message structure to MSMQ message properties. The MSMQ-MQSeries Bridge then transmits the message to the MSMQ destination queue.

- An MSMQ application then issues an MSMQ *MQReceiveMessage()* API call to receive the message from the MSMQ queue.

With MSMQ-MQSeries Bridge, you send messages either by normal service or high service.

Normal service makes use of MSMQ's and MQSeries' 'deliver once' feature, which ensures that each message is delivered once and only once to the receiving application. However, there is overhead associated with using this form of delivery. High service can improve performance, but a message may be delivered more than once in the event of a system failure during transmission.

The MSMQ-MQSeries Bridge sends messages that are part of a transaction by normal service and those that are not by high service when sending messages from MSMQ to MQSeries. When sending messages from MQSeries to MSMQ, you specify a particular alias for the remote queue manager address, and this determines whether messages are sent by normal or high service.

After installation, you need to configure MSMQ, MQSeries, and the MSMQ-MQSeries Bridge.

With MSMQ Explorer, you can:

- Set up foreign connected networks (CNs) that act as gateways to MQSeries.
- Set up foreign computers that act as MQSeries Queue Managers.
- Set up other foreign computers and computers that host MSMQ-MQSeries Bridge on foreign connected networks.

You can use either the MSMQ Explorer or the *MQCreateQueue()* API call to set up foreign queues that represent MQSeries queues to MSMQ.

Use the MSMQ-MQSeries Bridge Explorer to:

- Set up MQI (Message Queue Interface) channels to MQSeries.
- Define the MSMQ foreign connected networks to the MSMQ-MQSeries Bridge and set up their properties.

- Define the names of MQSeries transmission queues for normal and high service message pipes.
- Export definition files for transfer to MQSeries.

To configure the MSMQ-MQSeries Bridge for use in your MSMQ and MQSeries networks:

- Insert the foreign CNs that allow the MSMQ-MQSeries Bridge to communicate with MSMQ.
- Set up the MQSeries Queue Manager and MQI channel that allow MSMQ-MQSeries Bridge to communicate with MQSeries.
- Set up properties, such as timeouts and number of threads, that MSMQ-MQSeries Bridge uses for communication.
- Complete the MSMQ-MQSeries Bridge configuration, then export the configuration as MQSeries definitions and transfer them to MQSeries.

You need to define the properties of the following objects to send messages using the MSMQ-MQSeries Bridge:

- The MSMQ-MQSeries Bridges themselves
- Connected networks
- Message pipes.

While most of these properties take default values, some must be set up by the administrator.

In MQSeries, you need to:

- Import the definition files that you created in the MSMQ-MQSeries Bridge Explorer (alternatively, manually set up the transmission queues), queue manager aliases, model queues, and MQI channels for TCP/IP or SNA transport.
- Ensure that you use the same name for the same entity when defining it at different locations (for instance, an MSMQ-MQSeries Bridge computer called *SAMPLEMQ1* should have the same name in both the MSMQ-MQSeries Bridge Explorer and in MQSeries).

Within the MSMQ-MQSeries Bridge Explorer, right-click the MSMQ-MQSeries Bridge icon and choose the *Properties* option. The properties window displays several tabs. Before you add a connected network, set up the properties on the *MQI Channels* tab.

General tab

This displays the following read-only information:

- *Path name*

  The network name of the MSMQ-MQSeries Bridge computer.

- *Service name*

  The name of the MSMQ-MQSeries Bridge.

- *Version*

  The version of the MSMQ-MQSeries Bridge.

- *Status*

  Running, Paused, or *Stopped*.

Advanced tab

By default, MSMQ-MQSeries Bridge allocates one thread for each type of message pipe. If the MSMQ-MQSeries Bridge is connected to more than one MQSeries Queue Manager, you should allocate it a larger number of threads. This improves performance and availability (if a pipe to a Queue Manager fails, other pipes continue running). For each type of message pipe, specify:

- *Max threads*

  The maximum number of threads (the recommended number is one per MQSeries QM).

- *Refresh queue cache*

  The interval (in minutes) at which the message pipes check for cache timeout.

MQI channels tab

MSMQ-MQSeries Bridge accesses MQSeries via MQI channels defined in both MQSeries and MSMQ-MQSeries Bridge. Each channel connects an MSMQ-MQSeries Bridge to an MQSeries Queue Manager. Ordinarily, you should define one MQI channel for each foreign connected network.

First, you should define the properties of the channels in the MSMQ-MQSeries Bridge. You can then export the definitions to MQSeries. The channels currently defined are listed in the dialog box. Click *Add* to create a new MQI channel, *Properties* to edit the settings of an existing channel, or *Remove* to delete a channel.

1 On the *General* tab of the *Channel Properties* window, specify the following options:

   – *Channel name*

     An MQSeries name for the MQI channel. For convenience, you can assign the same name as you assigned to the connected network representing MQSeries in MSMQ, for example *MQS_CN*.

   – *MQSeries queue manager*

     The QM to which the channel connects.

   – *Transport type*

     TCP/IP or SNA.

2 On the *Address* tab of the *Channel Properties* window, specify the following parameters for TCP/IP:

   – *IP address and port*

     The IP address and port number of the MQSeries listener.

   For SNA specify:

   – *Side information record*

     The CPI-C symbolic name defined in SNA Server.

3 On the *Channel Properties* window's *Security* tab you specify:

– *MCA user*

  This is an existing or new MQSeries user name, for example *SAMPQUSER1*, under which the server side of the MQI channel runs.

  In MQSeries, you should set the MCA user's permissions (in other words, the queues that MSMQ-MQSeries Bridge can address). If you do not specify an MCA user, the server side of the channel runs under the default user name, which is given by the MQSeries *SYSTEM.DEF.SVRCONN* parameter.

Before you can add a foreign CN to an MSMQ-MQSeries Bridge, you should:

- Define the foreign CN in MSMQ and associate the MSMQ-MQSeries Bridge machine with the CN.
- Define an MQI Channel for the MSMQ-MQSeries Bridge.
- Open the MSMQ-MQSeries Bridge Explorer, if it is not already open.

To add a foreign CN, right-click the MSMQ-MQSeries Bridge to which the CN is connected. Choose *New*, *CN* from the pop-up menu, and select the CN name from the list. Along with the new CN, the following four message pipes are added to the tree in the left pane.

- The MSMQ-to-MQSeries normal service pipe:

```
MSMQ->MQS Normal
```

- The MSMQ-to-MQSeries high service pipe:

```
MSMQ->MQS High
```

- The MQSeries-to-MSMQ normal service pipe:

```
MQS->MSMQ Normal
```

- The MQSeries-to-MSMQ high service pipe:

```
MQS->MSMQ High
```

Right-click the new CN and set the properties of each message pipe. To delete a CN, right-click the CN and select *Delete* from the pop-up menu. To set the properties of a foreign connected network, right-click its icon in the MSMQ-MQSeries Bridge Explorer and choose the *Properties* option. On the *General* tab of the connected network properties window, specify the following options:

- *MQSeries QM name*

  Select from the list of MQSeries Queue Managers for which MQI channel definitions have been entered. Note that, if the same MSMQ-MQSeries Bridge is on two connected networks, they shouldn't both be connected to the same QM.

- *Reply-to QM name*

  This is the default MSMQ QM to which MQSeries should return report or acknowledgement messages. You would normally enter the name of the computer hosting the MSMQ-MQSeries Bridge here, for example *SAMPLEMQ1*.

  The MSMQ-MQSeries Bridge Explorer exports your entry to MQSeries as a queue manager alias. If the Windows NT machine name contains an invalid MQSeries character, such as a hyphen ('-'), replace it with another character, such as an underscore ('_') in the *Reply-to QM Name* field. If you want to receive acknowledgements by high rather than normal service, append a percentage sign ('%') to the name (for example *SAMPLEMQ1%*).

  As MQSeries uses aliases to identify the transmission queue to which it should send acknowledgements, you can redirect acknowledgements by specifying a different alias. For example, if you want to receive acknowledgements on another machine on which MSMQ-MQSeries Bridge is installed, enter the name of the machine and define the name as an alias in MQSeries.

- *Startup*

  Enabled or *Disabled* at MSMQ-MQSeries Bridge start-up (the procedure to change the status is detailed in the section below entitled *Starting, stopping, and pausing an object*).

To set the properties of a message pipe, right-click its icon in the MSMQ-MQSeries Bridge Explorer, and choose the *Properties* option. Here you'll find icons for message pipes.

The General tab of the message pipe properties window displays the following:

- *Status*

  Running, *Paused*, *Pending*, *Recovering*, *Stopped*, or *Error* (this property is read-only; again, see the section on *Starting, stopping, and pausing an object* for details of how you can change the message pipe's status).

- *Startup*

  Enabled or *Disabled* at MSMQ-MQSeries Bridge start-up.

The transmission queue name of an MQSeries-MSMQ normal or high service pipe is the unique MQSeries transmission queue name for the pipe. The default names are:

```
<CN name>.XMITQ
```

for a normal service pipe and:

```
<CN name>.XMITQ.HIGH
```

for a high service pipe. You may specify different names, if you wish. For example, if the CN name is *MQS_CN*, the default transmission queue names are:

```
MQS_CN.XMITQ
```

and:

```
MQS_CN.XMITQ.HIGH
```

Batch tab

To optimize performance, MSMQ-MQSeries Bridge batches messages. Increasing the batch size may improve performance, but it also increases the quantity of re-transmitted data after a communication failure.

To change the batch size, specify any combination of:

- *Maximum number of messages*

  The maximum number of messages in a batch.

- *Maximum accumulated size*

  The maximum size (in bytes) of a batch.

- *Maximum accumulated time*

  The maximum time (in milliseconds) during which messages are batched.

Transmission begins as soon as there are sufficient messages to be sent. When any of the above limits is reached, the message pipe checks that the batch was fully received on the destination side.

Cache tab

To reduce the overhead of opening and closing a queue for each message, MSMQ-MQSeries Bridge caches queue handles and reuses them when several messages are sent to the same queue. To customize this facility, specify the following option:

- *Cache expiry*

  The time (in minutes) after which MSMQ-MQSeries Bridge closes an unused queue handle.

The system checks the cache timeout only at the cache refresh time. Thus the maximum time that an unused queue handle could remain open is:

```
cache timeout + cache refresh time
```

Details on how to refresh the cache immediately (for example, to release a queue that is needed by another application) are provided in the section *Refreshing the cache* (at the end of this article).

Retry tab

If a message pipe fails, MSMQ-MQSeries Bridge tries to restart it automatically. For both the ‘short’ and ‘long’ retry cycles, specify:

- *Count*

  Maximum number of retries.

- *Delay*

  Interval between retries.

For example, you might specify a short cycle of three retries at 30-second intervals. If the connection is still not successful, the system continues in a long cycle of, say, ten retries at 300-second intervals.

Note that, if you connect by TCP/IP, you should set the *Delay* to at least twice the ‘keep alive’ time of the destination MQSeries system. This allows the MQSeries listener to release the resources of a broken connection before MSMQ-MQSeries Bridge tries to reconnect.

Many of the properties that you set in the MSMQ-MQSeries Bridge Explorer relate to MQSeries entities, such as:

- MQI channels
- Transmission queues
- QM names.

It is usually easier to enter names and options for these entities before you create the entities in MQSeries. This lets you export the data to a set of MQSeries definition files, which you can then use in MQSeries to create the entities automatically.

The MSMQ-MQSeries Bridge Explorer creates two types of definition file: one for MQSeries Server entities and one for MQSeries Client entities. Together, the files define all the MQSeries entities that you need for a basic MSMQ-MQSeries Bridge configuration.

The file *<QM_name>.txt* (for example, *mqs1.txt*) contains definitions for an MQSeries Server (one such file exists for each MQSeries Queue Manager), while the file *clientdf.txt* contains definitions for all MQSeries Client (only one such file is necessary).

To export the MQSeries definitions:

1 Configure the MSMQ-MQSeries Bridge in the MSMQ-MQSeries Bridge Explorer. Make sure you set all the properties for the

MQSeries connection, such as MQI channel names, TCP/IP or SNA details, MCA user name, QM names, and transmission queue names.

2 Right-click the MSMQ-MQSeries Bridge icon and choose *Export Server Definitions* from the pop-up menu.

3 In the dialogue box, specify a directory to store the *<QM_name>.txt* definition files.

4 Right-click the MSMQ-MQSeries Bridge and choose *Export Client Definitions* from the pop-up menu.

5 In the dialogue box, specify a directory to store the *clientdf.txt* definition file.

6 Repeat the above steps for each MSMQ-MQSeries Bridge in your network, storing each set of definition files in a separate directory.

7 Follow the instructions earlier in this article on configuring MQSeries and importing MQSeries definitions from the MSMQ-MQSeries Bridge Explorer to complete the MQSeries configuration.

To back up a configuration, use **regedt32** to copy the following registry entries to a back-up file:

- Registry key to back up for stand-alone MSMQ-MQSeries configuration:

```
HKLM\Software\Microsoft\MQBridge\Server
```

- Registry key to back up for clustered MSMQ-MQSeries configuration:

```
HKLM\Cluster\Software\Microsoft\MQBridge\Server
```

MSMQ-MQSERIES BRIDGE EXPLORER DISPLAY

Open the MSMQ-MQSeries Bridge Explorer from your MSMQ-MQSeries Bridge folder. The MSMQ-MQSeries Bridge Explorer window is divided into two panes. In the left pane, an expandable tree displays the MSMQ-MQSeries Bridge installations in your network, foreign CNs, and message pipes. The right pane shows detailed information about an object that is selected in the tree.

Column name:

- *MQS H threads*

  The number of high service MQSeries-to-MSMQ threads.

- *MQS N threads*

  The number of normal service MQSeries-to-MSMQ threads.

- *MSMQ H threads*

  The number of high service MSMQ-to-MQSeries threads.

- *MSMQ N threads*

  The number of normal service MSMQ-to-MQSeries threads.

- *Lifetime*

  The time since the MSMQ-MQSeries Bridge started.

- *Path name*

  The machine name.

- *Status*

  The MSMQ-MQSeries Bridge's status (*Running*, *Paused*, or *Stopped*).

- *DLQ depth*

  The number of messages on the MSMQ-MQSeries Bridge's untransacted dead letter queue.

- *XDLQ depth*

  The number of messages on MSMQ-MQSeries Bridge's transacted dead letter queue.

Foreign CN display

- *QM name*

  The name of the MQSeries Queue Manager to which the CN is connected.

- *Startup*

  This setting determines whether the CN is enabled or disabled at MSMQ-MQSeries Bridge start-up.

- *Status*

  The CN status (*Running*, *Paused*, or *Stopped*).

Message pipe display

- *Acc size*

  The accumulated size (in bytes) of all messages transmitted since the message pipe started.

- *Lifetime*

  The time elapsed since the message pipe started.

- *Messages sent*

  The total number of messages that were sent since the message pipe started.

- *Msgs/sec*

  The number of messages per second (the current throughput of the pipe).

- *Q depth*

  The queue depth (the number of messages in the queue waiting to be transmitted).

- *Q name*

  The name of the MQSeries transmission queue or MSMQ connector queue associated with the message pipe.

- *Retries*

  The number of times the MSMQ-MQSeries Bridge has tried to activate the message pipe.

- *Startup*

  Whether the message pipe is enabled or disabled at MSMQ-MQSeries Bridge start-up.

- *Status*

  The message pipe status (*Running, Paused, Pending, Recovering, Stopped,* or *Error*).

CUSTOMIZING THE COLUMN DISPLAY

To add or remove a column in the right pane of the MSMQ-MQSeries Bridge Explorer:

- Select *Columns* from the *View* menu.
- Select the tab that corresponds with the type of object whose display you wish to customize ('MSMQ-MQSeries Bridge', 'CN', or 'Message Pipe').
- To add a column to the display, select its name in the *Available Columns* list and click *Add.*
- To remove a column, select its name in the *Show* list and click *Remove.*
- To change the sequence of the column display, select *Columns* in the *Show* list and click the up or down arrow.

The context menu of the MSMQ-MQSeries Bridge Explorer

MSMQ-MQSeries Bridge Explorer offers the standard Windows Explorer menu and toolbar. In addition, you can display a special pop-up menu with one or more of the following options by right-clicking an object:

- *New*

- *Start*
- *Stop*
- *Refresh cache*
- *Pause*
- *Resume*
- *Delete*
- *Export server definitions*
- *Export client definitions*
- *Properties.*

The subset of options on the menu depends on the object and its current status.

Status bar

The status bar at the bottom of the MSMQ-MQSeries Bridge Explorer window displays status and error messages. For example, the status bar displays an access denied message if an unauthorized user tries to start or stop the MSMQ-MQSeries Bridge service.

CONTROLLING MSMQ-MQSERIES BRIDGE

To control the operation of an object, right-click the object in the MSMQ-MQSeries Bridge Explorer and choose the appropriate command. Some of the commands are also available on the main Explorer menu.

STARTING, STOPPING, AND PAUSING AN OBJECT

Foreign CNs and message pipes start automatically at MSMQ-MQSeries Bridge start-up, if you selected the *Startup Enabled* option in the object's properties. To change the status of an object, right-click the object in the MSMQ-MQSeries Bridge Explorer and select one of the following options:

- *Stop*

  Stops the object and resets its counters (lifetime, messages sent, and accumulated size).

- *Start*

  Starts a stopped object.

- *Pause*

  Pauses the object, retaining its counters.

- *Resume*

  Resumes the operation of an object after a *Pause*.

The commands affect the selected object and all objects below it in the hierarchy. For example, starting a foreign CN starts all its enabled message pipes.

REFRESHING THE CACHE

Occasionally, you may want to refresh the MSMQ-MQSeries Bridge cache memory. A reason for doing this is to close queues needed by other applications.

To do this, right-click an MSMQ-MQSeries Bridge's icon in the MSMQ-MQSeries Bridge Explorer and choose the *Refresh Cache* option.

Stephen Ibaraki
Technical Manager and Head of Research
Capilano College (Canada)

MQSeries Queue Manager Clusters

The popularity of MQSeries springs from the unarguable benefits of connecting disparate platforms – something that's essential in today's environment of mergers, acquisitions, and joint ventures. But, while everyone might agree that connecting platforms is necessary, no one considers it easy.

MQSeries Version 5.1 on distributed platforms and Version 2.1 on OS/390 ('z/OS') allow you to network groups of related MQSeries queue managers, known as *clusters*. Using clusters on large, complex queue manager networks significantly reduces MQSeries' administrative burden. In addition, the clusters allow workload to be handled across queue managers, resulting in easier workload balancing and dynamic failover support. Clusters truly support the principle of strength in numbers.

This article describes the structure of MQSeries clusters and defines clustering components and their functions. It also examines the benefits that clustering offers and explores the steps needed to set up clusters. (Also see *MQSeries clusters: a hands-on view* in issues 15 and 16 of *MQ Update*, which offer practical advice on clustering.)

STRUCTURE OF CLUSTERING

Queue managers can be aggregated in a cluster if they share a common characteristic, such as location, department, or business function. What makes a cluster useful and powerful is its ability to share queues among its members. Any queue controlled by one queue manager is accessible to any other queue manager in the cluster, as long as it's defined as a cluster queue. Queue managers added to the cluster can access all cluster queues, regardless of which cluster queue belongs to which queue manager. This is networking at its finest.

GATEWAYS

Combining clusters allows you to integrate large, complex MQSeries networks. A queue manager that belongs to two or more clusters is known as a *gateway*, and it allows services that are provided by cluster queues in one cluster to be available to applications in other clusters. By creating gateways to interconnect multiple clusters, it is possible to bridge MQSeries networks.

NEW OBJECTS AND TERMS

Several new MQSeries objects and terms are associated with clustering:

- A *cluster queue manager* belongs to at least one cluster. It may make some of the queues it owns known to other members of the cluster, keeping others ‘hidden’. A cluster queue manager may not own any queues at all, in which case it simply feeds messages to other queue managers in the cluster.
- A *cluster namelist* allows administrators to group clusters and refer to them as a single entity when defining cluster queue managers and queues. Namelists let a single queue or queue manager belong to multiple clusters.
- A *cluster queue* is owned by a queue manager that is a member of a cluster. It’s made known to other queue managers in the cluster by a new attribute that defines the cluster to which it belongs. Note that a cluster queue may belong to one or more clusters (namelists are used to make a queue known to more than one cluster). An application can only send data to cluster queues that aren’t local – you can’t ‘get’ from a cluster queue that isn’t local. No such restriction applies to cluster queues that are local to an application – you can both output and input messages from them.
- A *repository* collects information about queue managers that belong to a cluster. This information includes queue manager names and locations and the objects that the queue managers host. The repository is implemented via messages on a queue. There are two kinds of repository: *full* and *partial*. A full repository contains information on all queue managers in a cluster. A partial repository contains only enough information to exchange messages. Information in a partial repository is updated on a need-to-know basis, and a full repository is notified about all

information that is passed to a partial repository. This allows synchronization between partial and full repositories.

It is recommended that at least two queue managers in a cluster host full repositories. Other queue managers that are members of the cluster would then host partial repositories. Repository information is stored in a queue named:

```
SYSTEM.CLUSTER.REPOSITORY.QUEUE
```

- A *cluster receiver channel* is an end-point of a channel from which a cluster queue manager may receive messages, which may be destined for either an application or the repository. The queue manager announces that it is available to receive messages using the cluster receiver channel.
- A cluster queue manager can send cluster information to a full repository from a *cluster sender channel*. This channel also notifies repository queue managers of any changes to either a queue manager's status or objects that the queue manager hosts. It also transmits messages similar to those in standard sender channels. One cluster sender channel is initially required to allow the queue manager to join a cluster. Other cluster sender channels are defined automatically when the need arises.

CLUSTER TRANSMISSION QUEUE

Every cluster queue manager requires a cluster transmission queue (*SYSTEM.CLUSTER.TRANSMIT.QUEUE*). This queue moves all messages in a cluster from one queue manager to another. Unlike sender channels, which allow you to specify the transmit queue they service, cluster sender channels have no such attribute, using the cluster transmission queue instead. This queue's name cannot be changed and is created as one of the default objects.

BENEFITS OF CLUSTERS

With the working definitions above, we now have a context in which we can discuss the benefits that clusters provide, which include reduced administration and workload balancing.

Reduced administrative burden

Building an MQSeries network using clusters eliminates many of the object definitions that are required when configuring MQSeries distributed queuing using the ‘standard’ methodology. With fewer definitions, reconfiguration is also simplified. Another benefit is that fewer objects means a reduced likelihood of error when defining objects.

Let’s compare configuration of three queue managers using traditional distributed queuing versus clustering. A queue manager hosts a local queue that provides a service, such as responding to inventory status inquiries. It must communicate directly with other queue managers.

1 Objects for distributed queuing:

   In the simple configuration shown in Figure 1, definitions are required for 39 objects. If you need to add another queue manager, this requires 19 additional definitions on the queue itself and 18 additional definitions on existing queue managers. This would bring the total up to 76 objects. Clearly, this becomes a large burden as the number of queue managers increases. You can reduce the number of objects by not using queue manager aliases and process definitions for channel triggering, though this does not significantly reduce the administrative burden.

	Definitions needed	
MQSeries object	**Per queue manager**	**For three queue managers**
Sender channel to other’s queue manager	2	6
Receiver channel from other’s queue manager	2	6
Transmission queue	2	6
Local queue	1	3
Remote queue	2	6
Queue manager alias	2	6
Process definition for starting channel	2	6

Figure 1: Objects required for distributed queuing

2 Objects required with clusters:

In contrast to the distributed queuing example, a similar configuration with clustering (Figure 2) requires just nine objects. What is even more remarkable is the reduction in the number of objects that are required to add a fourth queue manager to the network. With clusters, the number of new definitions is constant: each new queue manager requires only three definitions, and existing queue managers in the cluster require no additional objects.

	Definitions needed	
MQSeries object	**Per queue manager**	**For three queue managers**
Cluster sender channel	1	3
Cluster receiver channel	1	3
Local queue (cluster queue)	1	3

Figure 2: Objects required using clustering

Clustering lightens the administration load in other ways as well:

- Moving a cluster queue from one queue manager to another requires no changes to remote queue definitions.
- Adding and removing queue managers from the cluster requires changes only to the queue manager being removed.
- Defining fewer objects lessens the risk of some types of problem, such as channel mismatches and transmission queue complications.

Clustering also eliminates the possibility of remote queues pointing to the wrong queue at a remote queue manager.

Managing the workload

Clustering allows you to manage your workload better and increase service availability. It takes full advantage of networking by allowing applications to obtain a service from one of a host of queue managers in the cluster. Multiple queue managers in the cluster each host a queue that provides a service. To applications requesting this service, it makes no difference where the service provider resides. A network (or cluster) of independent queue managers providing the service increases availability – if one queue manager is unavailable, applications can obtain the service from any other queue manager that's configured appropriately. This architecture also offers increased throughput and even workload distribution across queue managers.

Messages destined for a particular queue can be handled by any queue manager in the cluster that supports an instance of the queue. Remote queue definitions and applications no longer explicitly name the queue manager to which the message should be routed. The destination queue manager is determined by a *workload management algorithm*.

The algorithm that ships with MQSeries is very simple – it works as follows:

1 If the message destination queue is local to the queue manager to which the application is connected, the message is placed there.

2 If there is no instance of the queue that's local to the application, the algorithm determines which destinations are available. Destinations with higher priority, as set by the MQSeries administrator, are chosen first using the cluster receiver channel's *NETPRTY* attribute.

3 If multiple channels have the same priority (*NETPRTY*), the algorithm chooses destinations sequentially in a circular manner (*A*, *B*, *C*, *A*, *B*, *C*, etc). A destination is a queue manager in the cluster that hosts an instance of the local queue to which the message is destined.

This algorithm doesn't assess the ability of the destination queue manager to service requests in a timely manner – it just routes messages evenly among the available destinations.

However, you can develop a customized workload management algorithm that takes an input, such as CPU utilization and queue depth, and uses it to make a more informed decision on where to route a message. MQSeries provides an exit point that you can call on the

MQSeries *OPEN* or *PUT* calls. Program control and the information in the parameters are passed to the exit. The information includes both a list of available destinations and the one chosen by the queue manager. The exit can either accept this choice or override it based on other factors, such as the CPU utilization of the host or the service response time.

Note that MQSeries always attempts to put your message on a queue that is local to your application (or, more accurately, a queue that is local to the queue manager to which you are connected). Therefore, if you want to take advantage of workload management, you need to deploy an architecture in which the queue manager you connect to does not host an instance of the queue on which you want to put messages. If the queue on which you put your messages does have a local instance, and you don't necessarily want your messages to go to this queue, then you must use a custom workload exit.

Workload balancing may also require that you partition the data being accessed. Data access services that use clustering for workload balancing must make data accessible to requests that may end up on any instance of a queue. The workload management exit must be able to choose the destination queue based on the content of the request message. This ensures that requests go to queues that are capable of servicing them. It can be quite challenging to handle multiple request types using workload balancing, as only one workload management exit can be defined per queue manager.

SETTING UP A CLUSTER

Now that we have a basic knowledge of the structure of a cluster and its usefulness, we can examine the mechanics of deploying one. In most cases, a cluster will consist of at least two queue managers, which means there are at least two full repositories, providing a degree of fail-over. Consider a hypothetical MQSeries network belonging to Acme Co, with one queue manager in Vancouver, BC (Canada), and the other in Los Angeles, California. The Vancouver queue manager hosts a queue that provides data about the inventory at a Vancouver warehouse.

Below are the steps needed to set up this cluster. Bear in mind that you would issue the commands using whatever standard methods are associated with your platform.

1 Determine which queue managers will host full repositories (in our example, both queue managers do so). Our queue managers are called *LOS_ANGELES* and *VANCOUVER*, and between them they support our warehouse systems, so we'll call our cluster *WAREHOUSE*.

2 Configure channel listeners and initiators as you would if you were setting up traditional distributed queuing on both of the queue managers.

3 Alter the *VANCOUVER* queue manager to define it as a cluster queue manager that hosts a full repository:

```
ALTER QMGR REPOS(WAREHOUSE)
```

4 Define a cluster receiver channel (*CLUSRCVR*) on the *VANCOUVER* queue manager. This is required and can be accomplished with the following commands:

```
DEFINE CHANNEL(TO.VANCOUVER)
CHLTYPE(CLUSRCVR)
CONNAME('VANCOUVER.ACME.COM(1414)')
CLUSTER(WAREHOUSE)
```

5 Define a cluster sender channel (*CLUSSDR*) on the *VANCOUVER* queue manager. The cluster sender channel must point to a queue manager in the cluster that hosts a full repository. This step can be accomplished with the following commands:

```
DEFINE CHANNEL(TO.LOS_ANGELES)
CHLTYPE(CLUSSDR)
CONNAME('LOS_ANGELES.ACME.COM(1414)')
CLUSTER(WAREHOUSE)
```

6 Define a cluster receiver channel (*CLUSRCVR*) on the *LOS_ANGELES* queue manager:

```
DEFINE CHANNEL(TO.LOS_ANGELES)
CHLTYPE(CLUSRCVR)
CONNAME('LOS_ANGELES.ACME.COM(1414)')
CLUSTER(WAREHOUSE)
```

7 Define a cluster sender channel (*CLUSSDR*) on the *LOS_ANGELES* queue manager:

```
DEFINE CHANNEL(TO.VANCOUVER)
CHLTYPE(CLUSSDR)
CONNAME('VANCOUVER.ACME.COM(1414)')
CLUSTER(WAREHOUSE)
```

8 Define a cluster queue on one of the queue managers. In our example, the *LOS_ANGELES* application requires inventory status reports from the *VANCOUVER* warehouse. A service has been created at the *VANCOUVER* queue manager to provide this information. A queue is created on the *VANCOUVER* queue manager that services this requirement:

```
DEFINE QLOCAL(INVENTORY_STATUS)
CLUSTER(WAREHOUSE)
```

Figure 3 illustrates our small but functional cluster.

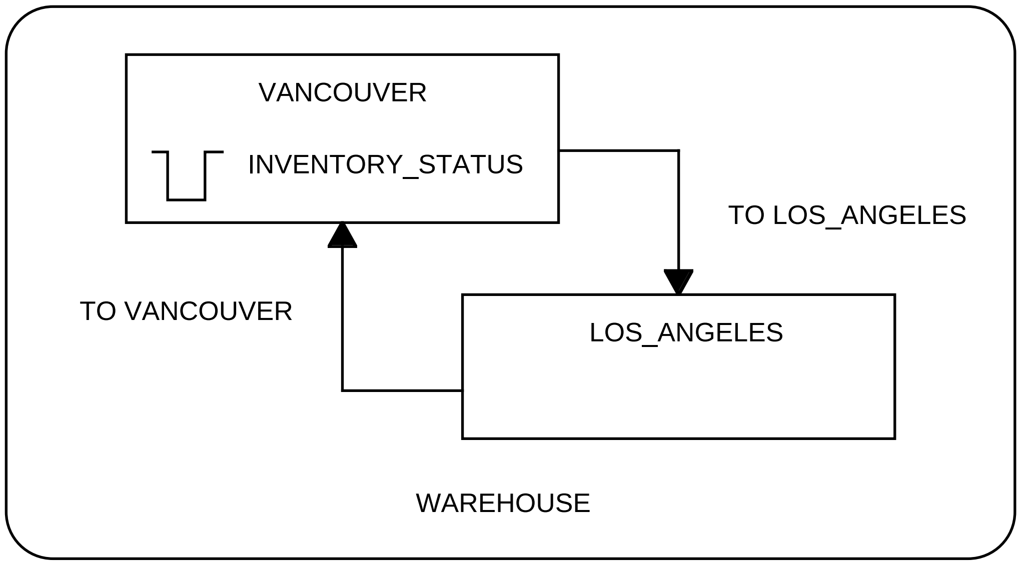

Figure 3: Basic cluster involving two locations

In our example, an application connecting to queue manager *LOS_ANGELES* can put messages on the *INVENTORY_STATUS* queue because information about *INVENTORY_STATUS* has been sent from queue manager *VANCOUVER* to queue manager

LOS_ANGELES. This information is stored in the repository at queue manager *LOS_ANGELES* in a queue named:

```
    SYSTEM.CLUSTER.REPOSITORY.QUEUE
```

An application servicing the *INVENTORY_STATUS* queue builds a reply using the 'reply to queue manager' and 'reply to queue' fields in the request's message descriptor to establish the reply's destination. The application will open this queue successfully even though there are no queue manager alias definitions. It knows about *LOS_ANGELES* in this case because the cluster queue managers share this information when they connect.

GROWING OUR NETWORK

In our Acme example, let's suppose a new queue manager is deployed in Dallas to support an on-line order processing system. This system requires access to both the *LOS_ANGELES* and *VANCOUVER* queue managers to collect inventory information.

Using traditional distributed queuing, you would need to define objects for facilitating a connection to *LOS_ANGELES* and, perhaps, *VANCOUVER* (you could always hop through *LOS_ANGELES* to *VANCOUVER* to reduce the number of definitions needed). You would also need remote queue definitions for access to the *INVENTORY_STATUS* queue and, ideally, a queue manager alias on *VANCOUVER* to return responses to *DALLAS*. Bringing *DALLAS* on-line would require some careful planning.

Using clusters, this effort is reduced substantially. Only two channel definitions are required: a cluster sender channel definition from *DALLAS* to any queue manager that hosts a repository (*VANCOUVER* or *LOS_ANGELES*, in our example) and a cluster receiver channel definition.

Once a connection is made to either *VANCOUVER* or *LOS_ANGELES*, the cluster queue managers pass information between them that allows *DALLAS* to access all cluster queues. No remote queue definitions are needed on *DALLAS* and no channel definitions are required on either *VANCOUVER* or *LOS_ANGELES*.

What if we make the connection from *DALLAS* to *LOS_ANGELES*? When we define the request to access the *INVENTORY_STATUS* queue on *VANCOUVER* from *DALLAS*, we don't do so via *LOS_ANGELES*, even though this is the queue manager with which we're registered in the cluster. Instead, the request causes the creation of objects to connect *DALLAS* directly to *VANCOUVER*. These are *dynamic objects* that you can display, but not alter or delete. The modified cluster is shown in Figure 4.

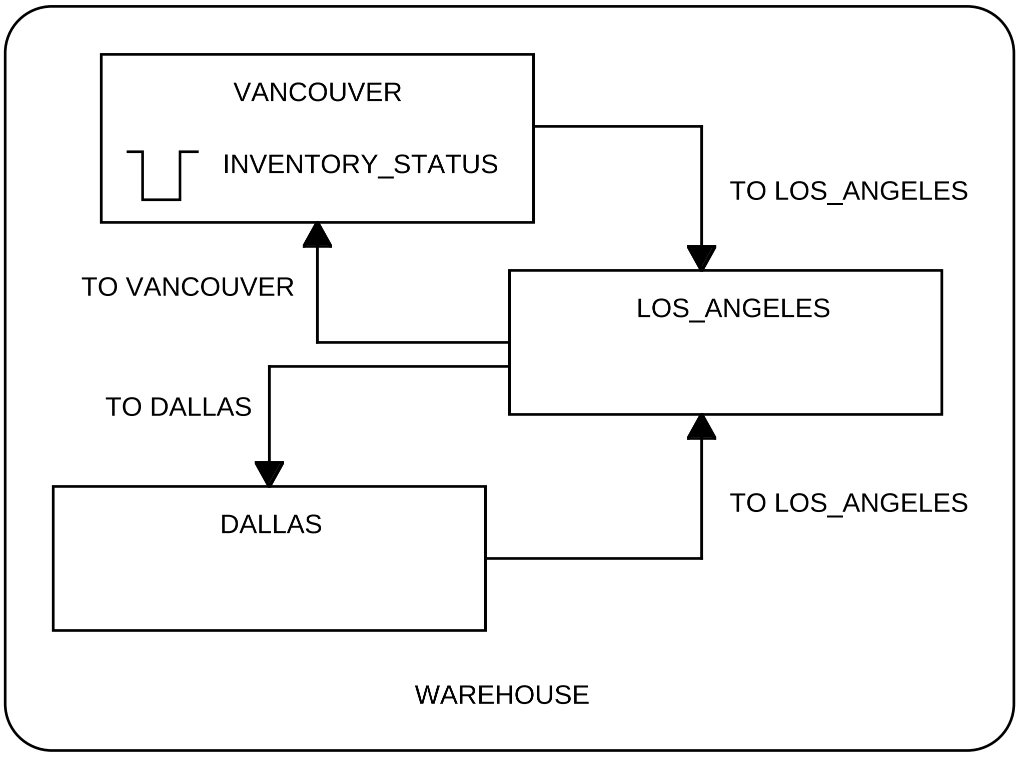

Figure 4: Enhanced cluster involving three locations

MIXING ENVIRONMENTS

So far we've looked at an environment comprising only cluster queue managers. However, real-life environments are typically more complex than this, which poses the question: can a queue manager in a cluster communicate with queue managers outside the cluster? The answer is 'yes'.

In our Acme example, you can use traditional distributed queuing techniques to route messages to queue managers outside the cluster. You can also define a single queue manager in a cluster that acts as a gateway to other queue managers outside the cluster. The traditional definitions on this queue manager, such as queue manager alias (or transmission queue), would include the cluster attribute, causing the definition to be propagated to other queue managers in the cluster – clustering provides benefits, even in a mixed environment.

CONCLUSION

Clustering is a significant step forward in easing the administrative burden of MQSeries and improving workload management. By using clusters, administrators are freed from the burden of defining numerous, tightly coupled objects just to connect queue managers. That process is replaced with the simple task of defining two channels. By networking queue managers, workload management occurs automatically, improving the availability and reliability of MQSeries-based applications. MQSeries workload management still requires tweaking, but this is a good starting point.

Marc Verhiel (marc_verhiel@candle.com)
Candle Corporation (USA)

APPC support using the 'SideInfo' dataset

This article demonstrates how to customize APPC for distributed queuing without CICS by editing the file *CSQ4SIDE*.

CSQ4SIDE, which is supplied in the *MQM.SCSQPROC* dataset, can be customized to meet your site's specific requirements. It's here that you can define a 'SideInfo' structure for APPC support. A 'side information definition' is required for each connection to a remote queue manager, and one side information definition is required for listening on connections from remote queue managers. This provides sample definitions for APPC side information used for distributed queueing without CICS.

AN EXAMPLE OF A CUSTOMIZED CSQ4SIDE JOB

```
//JOBCARD
//*CSQ4SIDE JOB
//*******************************************************************
//*
//* @START_COPYRIGHT@
//*   Statement:     Licensed Materials - Property of IBM
//*
//*                  5695-137
//*                  (C) Copyright IBM Corporation. 1993, 1996
//*
//*   Status:        Version 1 Release 1
//* @END_COPYRIGHT@
//*
//*******************************************************************
//*               IBM MQSeries for MVS/ESA
//* This provides sample definitions for the APPC side information
//* dataset that's used for distributed queueing without CICS.
//*
//*******************************************************************
//* CUSTOMIZE THIS JOB HERE FOR YOUR INSTALLATION
//* YOU MUST MAKE GLOBAL CHANGES ON THESE PARAMETERS USING YOUR
//* EDITOR
//*******************************************************************
//*
//*    Replace   ++SIDE++
//*                      with the name of the APPC side information
//*                      dataset to be used.
//*
//*    Replace   ++MVSconn++
//*              ++CICSconn++
//*              TST4D100
//*                      with the connection names to be used.
//*
//*    Replace   ++listen++
//*                      with the listener LU name to be used.
//*
//*    Replace   ++MVSluname++
//*              ++CICSluname++
//*              ++OS2luname++
//*                      with the names of the LUs used by the
//*                      remote queue managers.
//*
//*    Replace   ++LOCALluname++
//*                      with the name of the LU used by the
//*                       local queue manager.
//*
//******************************************************************
//*
//DEFINE   EXEC PGM=ATBSDFMU
//SYSPRINT DD   SYSOUT=*
//*SYSSDLIB DD   DSN=SYS1.APPC.APPCSI,DISP=SHR
//SYSSDLIB DD   DSN=SYSB.APPCSI,DISP=SHR
//SYSSDOUT DD   SYSOUT=*
//SYSIN    DD   DATA,DLM=ZZ
 /*
 /******************************************************************
 /*
 /* A side information definition is required for each connection
 /* to a remote queue manager, and one side information definition is
 /* required for listening on connections from remote queue managers.
 /*
 /* DESTNAME   For connections to a remote queue manager,
 /*            this is the connection name (CONNAME) specified by
 /*            the channel definition on the local queue manager.
 /*            For connections from a remote queue manager,
 /*            this is the logical unit name (LUNAME) specified by
 /*            the START LISTENER command for the local queue manager.
 /*
 /* MODENAME   This is the logon mode used for the SNA session
 /*            connecting the LU for the local queue manager
 /*            with the LU for the remote queue manager.
 /*
 /* TPNAME, PARTNER_LU
 /*            These values depend on the remote queue manager.
 /*
 /* There is a sample definition for connections to each type of
 /* remote queue manager. Repeat the definitions for each connection,
 /* specifying the DESTNAMEs that you want; customize the other
 /* values as appropriate.
 /*
 /* There is a sample definition for connections from remote queue
 /* managers. Specify the DESTNAME that you want; customize the
 /* other values as appropriate. (Multiple definitions can be set up
 /* if desired, but only ONE can be used for listening at a time.)
 /*
 /******************************************************************
 /*
 /* Sample side information for connection TO a remote queue manager on
 /* MVS/ESA without CICS, OS/400, or AIX:
 /*
 /*   TPNAME   Any value, but must be the same as in the corresponding
 /*            side information on the remote queue manager.
 /*   PARTNER_LU
 /*            The name of the LU used by the remote queue manager.
```

```
/*
/*  SIDELETE
/*       DESTNAME(++MVSconn++)
/*  SIADD
/*       DESTNAME(++MVSconn++)
/*       TPNAME(APINGD)
/*       MODENAME(#INTER)
/*       PARTNER_LU(++MVSluname++)
/*
/*******************************************************************
/*
/* Sample side information for connection TO a remote queue manager on
/* MVS/ESA using CICS:
/*
/*   TPNAME   CKRC for a connection for a sender or server channel,
/*            CKSV for a connection for a requester channel.
/*   PARTNER_LU
/*            The name of the LU used by the CICS system.
/*
/*  SIDELETE
/*       DESTNAME(++CICSconn++)
/*  SIADD
/*       DESTNAME(++CICSconn++)
/*       TPNAME(CKRC)
/*       MODENAME(#INTER)
/*       PARTNER_LU(++CICSluname++)
/*
/*******************************************************************
/*
/* Sample side information for connection TO a remote queue manager on
/* OS/2:
/*
/*   TPNAME   As specified on the OS/2 Run Listener command, or
/*            defaulted from the OS/2 queue manager configuration
/*            file.
/*   PARTNER_LU
/*            The name of the LU used by the OS/2 queue manager.
/*
    SIDELETE
         DESTNAME(ST17817C)
    SIDELETE
         DESTNAME(ST67817C)
    SIDELETE
         DESTNAME(SF67123C)
    SIADD
         DESTNAME(SF67123C)
         TPNAME(RECV)
         MODENAME(LU62C)
         PARTNER_LU(SF67123C)
    SIDELETE
```

```
     DESTNAME(SF61048C)
SIADD
     DESTNAME(SF61048C)
     TPNAME(RECV)
     MODENAME(LU62C)
     PARTNER_LU(SF61048C)
SIDELETE
     DESTNAME(SF69848C)
SIADD
     DESTNAME(SF69848C)
     TPNAME(RECV)
     MODENAME(LU62C)
     PARTNER_LU(SF69848C)
SIDELETE
     DESTNAME(SF61097C)
SIADD
     DESTNAME(SF61097C)
     TPNAME(RECV)
     MODENAME(LU62C)
     PARTNER_LU(SF61097C)
SIDELETE
     DESTNAME(SF67739C)
SIADD
     DESTNAME(SF67739C)
     TPNAME(RECV)
     MODENAME(LU62C)
     PARTNER_LU(SF67739C)
SIDELETE
     DESTNAME(SF67781C)
SIADD
     DESTNAME(SF67781C)
     TPNAME(RECV)
     MODENAME(LU62C)
     PARTNER_LU(SF67781C)
SIDELETE
     DESTNAME(SF61106C)
SIADD
     DESTNAME(SF61106C)
     TPNAME(RECV)
     MODENAME(LU62C)
     PARTNER_LU(SF61106C)
SIDELETE
     DESTNAME(SF67747C)
SIADD
     DESTNAME(SF67747C)
     TPNAME(RECV)
     MODENAME(LU62C)
     PARTNER_LU(SF67747C)
SIDELETE
     DESTNAME(SF61118C)
```

```
SIADD
     DESTNAME(SF61118C)
     TPNAME(RECV)
     MODENAME(LU62C)
     PARTNER_LU(SF61118C)
SIDELETE
     DESTNAME(SF61042C)
SIADD
     DESTNAME(SF61042C)
     TPNAME(RECV)
     MODENAME(LU62C)
     PARTNER_LU(SF61042C)
SIDELETE
     DESTNAME(SF61046C)
SIADD
     DESTNAME(SF61046C)
     TPNAME(RECV)
     MODENAME(LU62C)
     PARTNER_LU(SF61046C)
SIDELETE
     DESTNAME(SF67746C)
SIADD
     DESTNAME(SF67746C)
     TPNAME(RECV)
     MODENAME(LU62C)
     PARTNER_LU(SF67746C)
SIDELETE
     DESTNAME(SF67770C)
SIADD
     DESTNAME(SF67770C)
     TPNAME(RECV)
     MODENAME(LU62C)
     PARTNER_LU(SF67770C)
SIDELETE
     DESTNAME(SF69341C)
SIADD
     DESTNAME(SF69341C)
     TPNAME(RECV)
     MODENAME(LU62C)
     PARTNER_LU(SF69341C)
SIDELETE
     DESTNAME(SF67342C)
SIADD
     DESTNAME(SF67342C)
     TPNAME(RECV)
     MODENAME(LU62C)
     PARTNER_LU(SF67342C)
SIDELETE
     DESTNAME(SF67111C)
SIADD
```

```
          DESTNAME(SF67111C)
          TPNAME(RECV)
          MODENAME(LU62C)
          PARTNER_LU(SF67111C)
     SIDELETE
          DESTNAME(SF67387C)
     SIADD
          DESTNAME(SF67387C)
          TPNAME(RECV)
          MODENAME(LU62C)
          PARTNER_LU(SF67387C)
     SIDELETE
          DESTNAME(SF61103C)
     SIADD
          DESTNAME(SF61103C)
          TPNAME(RECV)
          MODENAME(LU62C)
          PARTNER_LU(SF61103C)
     SIDELETE
          DESTNAME(SF67771C)
     SIADD
          DESTNAME(SF67771C)
          TPNAME(RECV)
          MODENAME(LU62C)
          PARTNER_LU(SF67771C)
     SIDELETE
          DESTNAME(SF65256C)
     SIADD
          DESTNAME(SF65256C)
          TPNAME(RECV)
          MODENAME(LU62C)
          PARTNER_LU(SF65256C)
     SIDELETE
          DESTNAME(SF67753C)
     SIADD
          DESTNAME(SF67753C)
          TPNAME(RECV)
          MODENAME(LU62C)
          PARTNER_LU(SF67753C)
```

The remainder of this article appears in next month's issue of *MQ Update*.

Saida Davies
IBM (UK)

Articles wanted

The following topics have been proposed for articles in MQ Update. If you have any suggestions/comments on any of the topics, or (better still!) if you can write the article requested, please contact the Editor (see inside front cover for details).

1 Scaling the MQSeries administrative framework

MQSeries products are available on a variety of platforms, and the administration of queue managers on these platforms is usually tackled in a piecemeal way. There are some references in the MQSeries documentation to 'hub-and-spoke' architectures, and some nice ideas about MQSeries administration products with Web consoles and so on; however, what we need is an article on real-life MQSeries administration, using a single point-of-control for managing a complex MQSeries network. The article would deal with some or all of the following issues:

- Managing a large number of queue managers, channels, etc.
- Handling remote locations (or secure access).
- Managing a large number of object definitions using dynamic definitions.
- Routine problem fixes.
- 'Attention seeking' (how to send pages to the administrator, use flashing displays, etc).

2 Microsoft's .Net strategy and MQSeries

We're looking for an article giving an insight into what's possible with .Net, and what it and the MSMQ-MQSeries Bridge offer MQSeries users, revealing any surprises associated with integration, message attributes, dealing with a large number of messages, data conversion, etc.

3 Application development in MQSeries

MQSeries is used for many application types, such as request/response, FTP, scheduling, and event-based. Can anyone produce an article discussing design guidelines for such applications, tacking such subjects as how to engineer an application to deliver a performance target?

4 MQSeries applications for PDAs

MQSeries is now available on PDAs, and there is an exciting opportunity for developing, deploying, managing, and integrating applications as never before. We'd be very interested in some discussion on developing for MQSeries on PDAs, starting with a primer on PDA development, as most MQSeries developers will probably be unfamiliar with this.

5 MQSeries in disaster recovery

The way MQSeries is typically used in applications can result in updates to distributed data and/or data replication involving more than one database, such as DB2 UDB on NT and Oracle on the mainframe. Each of these databases has its own recovery options and schedules. However, from a business perspective, what's important is to tie the recovery of all the databases together. We'd like an article discussing the problems that using MQSeries brings to disaster recovery and providing tips on how these problems may be tackled.

6 MQSeries programming models

MQSeries can be programmed using a number of interfaces: AMI, JMS, MQI, and (possibly) SOAP. Can anyone compare and contrast the similarities and differences between these APIs from a programming perspective?

Contributions to *MQ Update* may be sent to the Editor at: *MQ Update, Xephon, 27-35 London Road, Newbury, Berkshire RG14 1JL, UK*. You may also e-mail articles to *info@xephon.com*. For more information about contributing, please download a copy of *Notes for contributors* from Xephon's Web site at *www.xephon.com/nfc.pdf*.

MQ news

MQSoftware is to enable its Q Pasa! management tool for MQSeries to work with IBM's WebSphere software, providing WebSphere and MQSeries users with a management system within their business applications. Q Pasa! provides management of MQSeries, MQSeries Integrator (Version 1 and 2), MQSeries Workflow, and MQSeries Everyplace, and provides management control of the MQSeries middleware environment including configuration, performance, problem, operations, and analysis management.

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

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

* * *

New Era of Networks has launched Open Business Interchange (also known as 'Open Biz'), which provides flexible and easy-to-use connections for B2C and B2B transactions. Acting as a connectivity hub between subscribers and their partners, it provides data transport, transformation, routing, and message tracking and authorization over virtually all standard Internet protocols and for most message formats. Open Biz allows trading partners to connect quickly using their chosen technology, eliminating the need for multiple gateways and sets of connectivity software.

Subscribers can choose formats, protocols, and encryption technologies independently. Support is provided for e-mail, FTP, MQSeries, HTTPS, XML, EDI, ebXML, and SOAP.

Available now as a hosted service or as a self-hosted service for beta customers, software licenses will be available in 2001. Pricing for the hosted service is subscription-based and depends on transaction volumes.

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 MQSeries for VSE/ESA Version 2.1.1, which comes with improved security, message data conversion, automatic VSAM reorganization, support for Java clients, and an improved batch interface.