6

MQ update

December 1999

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
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.

‘Lowest common denominator’ file transfer

After successfully completing a proof-of-concept or pilot MQSeries project, there is often an understandable push to deploy MQSeries as the enterprise’s data transfer solution. Indeed, the implementation of a single corporate-wide messaging infrastructure is often a primary goal behind an evaluation of MQSeries, with the expected standardization, simplified maintenance, and centralized management being key justifications.

However, implementing MQSeries is not just a simple deployment but a combination of integrating new visual front-ends with legacy systems, supporting new OLTP applications, and ‘retrofitting’ existing batch file-oriented applications to the new standard. The numerous bridges that come with MQSeries facilitate integration with legacy systems, while MQSeries’ ability to provide both synchronous and asynchronous communication satisfy the requirements of OLTP and batch applications respectively.

The aim of this article is to investigate MQSeries’ support for the batch environment and to describe an application that implements the MQ API to provide generic file transfer capabilities.

MQSeries Version 5.0 provides two new built-in facilities for sending files that exceed the maximum size for a single message: *message segmentation* and *reference messages*. However, if an organization needs to integrate platforms that include Level 1 or Level 2 platforms (Digital VAX VMS or OpenVMS, SCO Open Server and UnixWare, Tandem NSK, VSE/ESA, and Windows), then a programmatic method is required. Clearly the minimum requirement of a programmatic solution is that it provides a way of identifying components that belong to the same file and establishing their logical order or sequence.

There are two obvious ways of providing this information: by prefixing it to each message and by using fields in the *MQMD* message descriptor that are available to the user. The advantage of the former is that it offers greater extensibility, as the application developer is not constrained by the size or type of *MQMD* fields, while the advantage of the latter is that (except for Level 1 products) MQSeries supports a selective *MQGET* by *MsgId* and/or *CorrellId* that simplifies the task of rebuilding the file at the receiving end if these fields are used to hold key data. This article describes a solution that uses the *MQMD*, though it also draws attention to alternative methods that may be required to overcome the limitations imposed by Level 1 products.

The *MsgId* and *CorrellId* fields are contiguous, 24-byte fields that were introduced to the *MQMD* in the first version of MQSeries. The fact that they are contiguous means that the developer has a forty-eight byte array in which to store file control data. On the other hand, the fact that they are defined as byte fields by MQSeries means that the application is responsible for any data conversion required. One way to embed the necessary file control data in the *MQMD* is simply to write the data in the *MsgId* field using, say, C's *sprintf* function (care is needed to ensure that no more than forty-eight bytes are written). The data could be in the form *filename recordnumber recordtotal*, with a space between each item. The *filename* can be simple or fully-qualified. The *recordnumber-recordtotal* pair (for instance, '1 2' or '2 2') provides the only data required to verify that all segments of a file are received and that the file can be reconstructed in the correct sequence. Alternatively, *filename* can be put in the *MsgId*, with the *record* pair being stored in the *CorellId*, or the *MsgId* could contain *filename recordnumber* while the *CorellId* contains *recordtotal* alone. The logic of the receiver program in executing the selective *GET*s by *MsgId*/*CorellId* is all that needs to be implemented.

The basic sender program flow can now be specified (note the use of the continuation character, '➤', to indicate that one line of code maps to several lines of print):

```
Open file FILENAME
Calculate TOTALRECORDS (size of file FILENAME divided by
                        ➤  MESSAGESIZE)
QM = MQCONN
Q = MQOPEN TARGETQNAME for output
While not EOF FILENAME
    THISRECORD += 1
    BYTESREAD = read MESSAGESIZE bytes from file into MESSAGEBUF
    Set MQMD.MsgId = "FILENAME THISRECORD TOTALRECORDS"
    MQPUT (QM Q MQMD PUTOPTIONS BYTESREAD MESSAGEBUF CC RC)
MQCLOSE Q
MQDISC QM
```

In a homogeneous environment, *MESSAGESIZE* can be either a fixed value that is the smallest maximum message size of all supported platforms or a known, specified size for each application. A more flexible approach is to use the *MQINQ* call to find the maximum message size applicable. While it’s possible to acquire this attribute from the queue manager, the value retrieved could be misleading as it’s the one that controls the maximum message size that can be applied to a new queue. This means that it’s possible that there are messages on queues that exceed this queue manager value as they were *PUT* when the value was higher. A better method is to inquire about the maximum message size of the target queue itself. This is done in a multi-step process if we want the flexibility of supporting aliases and local and remote queues in the same generic application. First, we need to establish the type of the *TARGETQNAME* with an *MQINQ* call. If it’s an alias, we then need to establish its base queue name using a second *MQINQ* call. We can’t inquire about the maximum message size of the actual queue object yet as the queue may be local or remote and a remote queue may not have a directly associated size. So we use *MQINQ* again to establish the type of the actual queue object. If it’s a remote queue, we use *MQINQ* to get the name of the transmit queue associated with it. Finally, we can use *MQINQ* to determine the effective maximum message length *MESSAGESIZE*.

If all applications that receive data via MQSeries and process records or transactions contained in the files received do so regardless of completeness or order, then there is no need to send control file data in the first place. In this case persistent messages are likely to be used and it’s also likely that the file is sent using either MQSeries’ syncpoint or unit-of-work control, where available (VMS VAX, SCO Unix, and UnixWare being the exceptions). This is done by specifying *MQPMO_SYNCPOINT* in *PUTOPTIONS*, checking the *CC* (Completion Code) of each *MQPUT* call, and calling either *MQBACK* to back out of the operation if any *CC* indicates failure or *MQCMIT* to commit the work when *EOF* is reached. The developer should remember that MQSeries itself can return an *MQRC_BACKED_OUT RC* (Reason Code) on either an *MQCMIT* or *MQPUT* call, indicating that the queue manager has backed out of the syncpoint operation so far. This typically happens if a resource error, such as insufficient log space, is encountered. By specifying *MQOO_FAIL_IF_QUIESCING* on any open call, the system is able to handle the queue manager's imminent shutdown while maintaining data integrity.

More optional are enhancements that optimize front-end file processing or format the data in ways that make it easier for legacy back-end applications to process. An example of the former is to read the file into a memory buffer larger than the *MESSAGESIZE*, where the *MESSAGEBUF* is simply windowed over it on subsequent *MQPUT* calls. An example of the latter, the sender application can ensure that logical records do not span messages and that records are blank-padded before sending, though this may result in a noticeable communications overhead in a distributed environment. Similarly, options that allow messages to be sent starting with one other than the first, or that stop transmission before the last, can be useful for error recovery and re-sends, as well as for testing. In addition, information that allows the receiving application to execute another application once the file is rebuilt can be embedded in the *MQMD*. Other user-accessible *MQMD* fields, such as *ApplIdentityData* and *ApplOriginData*, can also be used judiciously to support this additional functionality.

The sending application may utilize MQSeries' *COA* and *COD* report options, as well as the exception and expiration report options, where available. Alternatively, it could use the reply's *QM* and *Q* fields to enable the receiving application to return a built-in MQ or application-specific positive/negative acknowledgement (*PAN/NAN*), which reports on complete and successfully processed files rather than just component messages received.

So let's now turn to the basic operation of the receiver application. This is typically triggered by the arrival of the first message on the target queue it services, and it implements *MQGET* with *wait* in order to service messages or new files that may arrive continually at short intervals. The trigger may also have to be developed in-house if the platform (typically Level 1) does not provide one.

Here again the application should make use of *MQINQ* to determine the maximum size of messages that need processing and use this to allocate a sufficiently large receiving buffer. If the *RC* is

MQRC_TRUNCATED_MSG_FAILED, the receiver should specify the *GET* option *MQGMO_ACCEPT_TRUNCATED_MSG* and *GET* the message again after expanding the receiving buffer.

There are a number of ways in which the receiving application can rebuild files. The simplest is to ensure that the first message read has *THISRECORD* set to *1*. If it has, the receiver can issue a selective *GET* with the *MsgID* and/or *CorellId* to retrieve the next *THISRECORD* needed using *FILENAME* and *TOTALRECORDS*. This assumes that the queue is dedicated to files of this application's type, so it might be desirable, for example, for the application to check the beginning of the *FILENAME* to ensure that it matches a particular pattern. In either case, the application needs to open the queue exclusively so that another application doesn't *GET* the first record of a file and leave the second one stuck as the initial one on the queue.

In the vast majority of cases this simple approach works well enough. In order to handle more exceptions, however, the receiving application may wish to take advantage of MQ's browse capability to make an initial complete pass of the queue, keeping track of all *FILENAME*s, *TOTALRECORDS*, and *THISRECORD*s encountered. For the last of these items, the check may comprise no more than a simple count, though if you're cautious about ensuring that a total of six is the sum of one, two, and three and not two, two, and two, then you should use a bit field. Now the application is ready to rebuild the first complete file. It could, at this stage, return a count indicating the number of complete files remaining, so that its invoking job (a shell script or JCL) can process the rebuilt file and invoke the receiver again. Alternatively, the application itself could rebuild all the complete files it encounters on each pass and leave the ones that aren't yet complete for a subsequent pass, when the missing parts may have arrived or the files could be flagged as errors and disposed of. If browsing is used, *GET* with *wait* is used on the browse pass and immediate *GET*s are used on subsequent passes to rebuild the file.

On platforms that don't fully support the browse capability, there are a number of ways of achieving a similar effect. One approach is to browse the first message and, if it's got the *THISRECORD* required, *GET* the message under the cursor, using the application to handle error conditions arising from out-of-sequence messages and, possibly, a redirection and/or error queue to hold messages that may otherwise cause application deadlock. Another option is the use of a sparse file (such as a Unix 'dd') to handle out-of-sequence messages, leaving the application to handle error conditions based on incomplete files. On platforms where only the ability to browse the first message is supported, it is also necessary to handle the unlocking of the record.

Remember that, even if you make an *MQGET* call with *MQGMO_CONVERT* specified, the *MsgId* and *CorellId* will not be automatically converted by MQ as they're *MQBYTE24* fields. In common with the sender, the application should make use of the truncated message, syncpoint, and fail functionality that MQSeries provides if quiescing.

When implemented this way, a single sender/receiver application pair can support most, if not all, legacy applications that need file transfer capabilities. The result is a standard environment that is easy to administer, control, and extend to embrace new systems, including external systems belonging to business partners. Another advantage of this approach is the lack of an implicit requirement for specific platforms and MQSeries revision levels. The system may also be modified easily to support new business requirements, as they arise. In short, a 'least common denominator' approach to file transfer via MQSeries may well be the most flexible and robust one as well.

Paul O'Pella
Project Manager
'The Pep Boys – Manny, Moe and Jack' (USA)

Creating QM definition scripts from the BSDS

This month's instalment concludes this article on creating queue manager definitions (the first part appeared in last month's issue).

PPFC14M0 (CONTINUED)

```
.ENDBOOL AIF   ('&PF_NEST(&PF_NI)'(5,4) EQ 'DO').DOEND
         POPINS    &PF_ST(&PF_NI+1)
         BC    15-&PF_CCVAL,&PF_LIND(&PF_LI-1)
         AIF   (NOT &ORIND).POPLBL
&PF_LIND(&PF_LI)   EQU   *
.POPLBL  ANOP
&PF_LI   SETA  &PF_LI-1
         MEXIT
.DOEND   ANOP
&PF_CTR  SETA  &PF_ST(&PF_NI+1)
         AGO   .ENDLBL
.NXTLBL  AIF   ('&PF_IIND3(&PF_CTR)' NE '&PF_LIND(&PF_LI)').INCTR
&PF_IIND3(&PF_CTR) SETC  '&PF_LIND(&PF_LI-3)'
.INCTR   ANOP
&PF_CTR  SETA  &PF_CTR+1
.ENDLBL  AIF   (&PF_CTR LE &PF_II).NXTLBL
         POPINS    &PF_ST(&PF_NI+1)
         BC    &PF_CCVAL,&PF_LIND(&PF_LI-3)
         AIF   (NOT &ANDIND).POP2LBL                         @BA43405
&PF_LIND(&PF_LI-1) EQU   *
.POP2LBL ANOP
&PF_LI   SETA  &PF_LI-2
&PF_NEST(&PF_NI)   SETC  '   Y'.'&PF_NEST(&PF_NI)'(5,4)
         MEND
**********************************************************************
         MACRO
         IF    &P1,&P2,&P3,&P4,&P5,&P6,&P7,&P8,&P9,&P10,&P11,&P12,&P13,X
               &P14,&P15,&P16,&P17,&P18,&P19,&P20,&P21,&P22,&P23,&P24, X
               &P25,&P26,&P27,&P28,&P29,&P30,&P31,&P32,&P33,&P34,&P35, X
               &P36,&P37,&P38,&P39,&P40,&P41,&P42,&P43,&P44,&P45,&P46, X
               &P47,&P48,&P49,&P50,&CC=
         PUSHNEST IF
         PUSHLAB
         IFPROC    &CC,&P1,&P2,&P3,&P4,&P5,&P6,&P7,&P8,&P9,&P10,&P11,  X
               &P12,&P13,&P14,&P15,&P16,&P17,&P18,&P19,&P20,&P21,&P22, X
               &P23,&P24,&P25,&P26,&P27,&P28,&P29,&P30,&P31,&P32,&P33, X
               &P34,&P35,&P36,&P37,&P38,&P39,&P40,&P41,&P42,&P43,&P44, X
               &P45,&P46,&P47,&P48,&P49,&P50
         MEND
**********************************************************************
         MACRO
         ELSE  &OPTN                                       @BIAH3WI
         COPY  PPFGBLC0
         AIF   ('&OPTN' EQ 'NULL').EXIT                    @BIAH3WI
&PF_LIND(&PF_LI+1) SETC  '&PF_LIND(&PF_LI)'
&PF_LI   SETA &PF_LI-1
         PUSHLAB
         AIF   ('&OPTN' EQ 'NOBRANCH').BYPBR               @BIAH3WI
         BC    15,&PF_LIND(&PF_LI)
.BYPBR   ANOP                                                  @BIAH3WI
&PF_LIND(&PF_LI+1) EQU   *
.EXIT    ANOP                                                  @BIAH3WI
         MEND
***********************************************************************
         MACRO
         ENDIF
         COPY  PPFGBLCO
         POPNEST   IF
&PF_LIND(&PF_LI)   EQU   *
&PF_LI   SETA  &PF_LI-1
         MEND
***********************************************************************
         MACRO
         DOPROC &FROM,&TO,&BY,&UNTIL,&WHILE,&P1
         COPY  PPFGBLCO
         LCLA  &I
         LCLC  &LCLWK1
         PUSHLAB
         PUSHINS   (EQU,*,,,,&PF_LIND(&PF_LI))
&PF_ST(&PF_NI)     SETA  &PF_II+1
         PUSHLAB
         AIF   (T'&FROM EQ 'O').NOIND
         AIF   ('&FROM(3)' EQ '').INCR
         LA    &FROM(3),&PF_LIND(&PF_LI)
.INCR    ANOP
&I       SETA  &I+1
         AIF   ('&SYSLIST(&I,2)' EQ '').TEST
         AIF   ('&SYSLIST(&I,2)' EQ '0').GENSR
         AIF   ('&SYSLIST(&I,2)'(1,1) EQ '-').NEGVAL
         AIF   (T'&SYSLIST(&I,2) EQ 'N').POSVAL
         AIF   ('&SYSLIST(&I,2)'(1,1) EQ '(').GENLR
         L     &SYSLIST(&I,1),&SYSLIST(&I,2)
         AGO   .TEST
.GENLR   LR    &SYSLIST(&I,1),&SYSLIST(&I,2)
         AGO   .TEST
.POSVAL  AIF   (&SYSLIST(&I,2) GE 4096).TSTMAG
         LA    &SYSLIST(&I,1),&SYSLIST(&I,2)
         AGO   .TEST
.TSTMAG  AIF   (&SYSLIST(&I,2) GE 32768).FULLIT
         AGO   .HALFLIT
.NEGVAL  ANOP
&LCLWK1  SETC  '&SYSLIST(&I,2)'(2,7)
         AIF   (&LCLWK1 GE 32768).FULLIT
.HALFLIT LH    &SYSLIST(&I,1),=H'&SYSLIST(&I,2)'
         AGO   .TEST
.FULLIT  L     &SYSLIST(&I,1),=F'&SYSLIST(&I,2)'
         AGO   .TEST
.GENSR   SR    &SYSLIST(&I,1),&SYSLIST(&I,1)
```

```
.TEST    AIF   (&I LT 3).INCR
         AIF   (T'&UNTIL NE 'O').ERRMG2
.CKWHILE AIF   (T'&WHILE NE 'O').COMPGEN
&PF_LIND(&PF_LI)   EQU   *
.POSTIND AIF   (T'&P1 EQ 'O').GETIND
         AIF   (T'&BY NE 'O').PFB
         AIF   (T'&TO NE 'O').PFT
         AIF   ('&FROM(3)' NE '').BCTRZ
         PUSHINS   (BCT,&FROM(1),&PF_LIND(&PF_LI))
         AGO   .ERRMG
.BCTRZ   PUSHINS   (BCTR,&FROM(1),&FROM(3))
         AGO   .ERRMG
.PFT     PUSHINS   (&P1,&FROM(1),&TO(1),&PF_LIND(&PF_LI))
         MEXIT
         PUSHINS   (&P1,&FROM(1),&BY(1),&PF_LIND(&PF_LI))
         MEXIT
.GETIND  AIF   ('&FROM(3)' EQ '').BCTR1
         PUSHINS   (BCTR,&FROM(1),&FROM(3))
         MEXIT
.BCTR1   AIF   (T'&BY NE 'O').FB
         AIF   (T'&TO EQ 'O').FONLY
         PUSHINS   (BXLE,&FROM(1),&TO(1),&PF_LIND(&PF_LI))
         MEXIT
.FONLY   PUSHINS   (BCT,&FROM(1),&PF_LIND(&PF_LI))
         MEXIT
.FB      AIF   (T'&TO NE 'O').FTB
         AIF   ('&BY(2)' EQ '').GENBXLE
         AIF   ('&BY(2)'(1,1) NE '-').GENBXLE
         AGO   .GENBXH
.FTB     AIF   ('&TO(2)' EQ '' OR '&FROM(2)' EQ '').GENBXLE
         AIF   ('&FROM(2)'(1,1) EQ '-').TRYTNEG
         AIF   (T'&FROM(2) NE 'N').GENBXLE
         AIF   ('&TO(2)'(1,1) EQ '-').GENBXH
         AIF   (T'&TO(2) NE 'N').GENBXLE
         AIF   (&FROM(2) GT &TO(2)).GENBXH
.GENBXLE PUSHINS   (BXLE,&FROM(1),&BY(1),&PF_LIND(&PF_LI))
         MEXIT
.TRYTNEG AIF   ('&TO(2)'(1,1) NE '-').GENBXLE
         AIF   ('&FROM(2)'(2,7) GE '&TO(2)'(2,7)).GENBXLE
.GENBXH  PUSHINS   (BXH,&FROM(1),&BY(1),&PF_LIND(&PF_LI))
         MEXIT
.NOIND   AIF   (T'&WHILE EQ 'O').NOWHILE
         AIF   (T'&UNTIL NE 'O').COMPGEN
         BC    15,&PF_LIND(&PF_LI)
         PUSHLAB
&PF_LI   SETA  &PF_LI-1
&PF_LIND(&PF_LI+1) EQU *
         AIF   ('&WHILE(6)' EQ '').OKSUBL
         STKINS    &WHILE
         MEXIT
```

```
.OKSUBL  STKINS    (&WHILE(1),&WHILE(2),&WHILE(3),&WHILE(4),          X
               &WHILE(5),&PF_LIND(&PF_LI))
         AIF   ('&WHILE(2)' EQ '').LABEL
         PUSHINS   (BC,&PF_CCVAL,&PF_LIND(&PF_LI+1))
         MEXIT
.LABEL   PUSHINS   (BC,&PF_CCVAL,&PF_LIND(&PF_LI+1),,,&PF_LIND(&PF_LI))
         MEXIT
.NOWHILE AIF   (T'&UNTIL EQ 'O').TRYINF
&PF_LIND(&PF_LI)   EQU   *
.UNT     STKINS    &UNTIL
         PUSHINS   (BC,15-&PF_CCVAL,&PF_LIND(&PF_LI))
         MEXIT
.TRYINF  AIF   ('&P1' NE 'INF').ERRMG1
&PF_LIND(&PF_LI)   EQU   *
         PUSHINS   (BC,15,&PF_LIND(&PF_LI))
         MEXIT
.COMPGEN AIF   ('&WHILE(6)' EQ '').OK
         STKINS    &WHILE
         AGO   .BCHINST
.OK      STKINS    (&WHILE(1),&WHILE(2),&WHILE(3),&WHILE(4),          X
               &WHILE(5),&PF_LIND(&PF_LI))
         AIF   (N'&WHILE GT 1).ENDCOMP
&PF_LIND(&PF_LI)   BC    15-&PF_CCVAL,&PF_LIND(&PF_LI-1)
         AGO   .FLAGEQU
.ENDCOMP ANOP
&PF_ST(&PF_NI+1)   SETA  &PF_II
         POPINS    &PF_ST(&PF_NI+1)
.BCHINST BC    15-&PF_CCVAL,&PF_LIND(&PF_LI-1)
.FLAGEQU ANOP
&PF_NEST(&PF_NI)   SETC  '   Y'.'&PF_NEST(&PF_NI)'(5,4)
         AIF   (T'&FROM NE 'O').POSTIND
         AGO   .UNT
.ERRMG   MNOTE 4,'POSITIONAL PARAMETER IGNORED. BCT/BCTR LOOP END USED'
         MEXIT
.ERRMG2  MNOTE 4,'UNTIL KEYWORD INVALID WITH INDEXING GROUP. IGNORED'
         AGO   .CKWHILE
.ERRMG1  MNOTE 4,'NO WHILE,UNTIL,OR INDEXING PARAMETERS ON DO MACRO.'
         MEND
***********************************************************************
         MACRO
         DO    &P1,&FROM=,&TO=,&BY=,&UNTIL=,&WHILE=
         PUSHNEST DO
         DOPROC    &FROM,&TO,&BY,&UNTIL,&WHILE,&P1
         MEND
***********************************************************************
         MACRO
         DOEXIT    &P1,&P2,&P3,&P4,&P5,&P6,&P7,&P8,&P9,&P10,&P11,&P12, X
               &P13,&P14,&P15,&P16,&P17,&P18,&P19,&P20,&P21,&P22,&P23, X
               &P24,&P25,&P26,&P27,&P28,&P29,&P30,&P31,&P32,&P33,&P34, X
               &P35,&P36,&P37,&P38,&P39,&P40,&P41,&P42,&P43,&P44,&P45, X
               &P46,&P47,&P48,&P49,&P50,&CC=
         COPY  PPFGBLCO
         PUSHLAB
&PF_NEST(&PF_NI)   SETC  '   Y'.'&PF_NEST(&PF_NI)'(5,4)
         IFPROC    &CC,&P1,&P2,&P3,&P4,&P5,&P6,&P7,&P8,&P9,&P10,&P11,  X
               &P12,&P13,&P14,&P15,&P16,&P17,&P18,&P19,&P20,&P21,&P22, X
               &P23,&P24,&P25,&P26,&P27,&P28,&P29,&P30,&P31,&P32,&P33, X
               &P34,&P35,&P36,&P37,&P38,&P39,&P40,&P41,&P42,&P43,&P44, X
               &P45,&P46,&P47,&P48,&P49,&P50
         MEND
***********************************************************************
         MACRO
         ENDDO
         GBLA   &PF_ST(51),&PF_NI,&PF_LI,&PF_II
         POPINS &PF_ST(&PF_NI)
&PF_II   SETA   &PF_II-1
         POPNEST   DO
&PF_LI   SETA  &PF_LI-2
         MEND
***********************************************************************
         MACRO
         STRTSRCH  &P1,&FROM=,&TO=,&BY=,&UNTIL=,&WHILE=
         PUSHLAB
         PUSHNEST  SRCH
         DOPROC    &FROM,&TO,&BY,&UNTIL,&WHILE,&P1
         PUSHLAB
         MEND
***********************************************************************
         MACRO
         EXITIF    &P1,&P2,&P3,&P4,&P5,&P6,&P7,&P8,&P9,&P10,&P11,&P12, X
               &P13,&P14,&P15,&P16,&P17,&P18,&P19,&P20,&P21,&P22,&P23, X
               &P24,&P25,&P26,&P27,&P28,&P29,&P30,&P31,&P32,&P33,&P34, X
               &P35,&P36,&P37,&P38,&P39,&P40,&P41,&P42,&P43,&P44,&P45, X
               &P46,&P47,&P48,&P49,&P50,&CC=
         IFPROC    &CC,&P1,&P2,&P3,&P4,&P5,&P6,&P7,&P8,&P9,&P10,&P11,  X
               &P12,&P13,&P14,&P15,&P16,&P17,&P18,&P19,&P20,&P21,&P22, X
               &P23,&P24,&P25,&P26,&P27,&P28,&P29,&P30,&P31,&P32,&P33, X
               &P34,&P35,&P36,&P37,&P38,&P39,&P40,&P41,&P42,&P43,&P44, X
               &P45,&P46,&P47,&P48,&P49,&P50
         MEND
***********************************************************************
         MACRO
         ORELSE
         COPY  PPFGBLCO
&PF_LIND(&PF_LI+1) SETC  '&PF_LIND(&PF_LI)'
&PF_LI   SETA  &PF_LI-1
         PUSHLAB
         BC    15,&PF_LIND(&PF_LI-3)
&PF_LIND(&PF_LI+1) EQU   *
&PF_NEST(&PF_NI)   SETC  '  P'.'&PF_NEST(&PF_NI)'(4,5)
```

```
         MEND
**********************************************************************
         MACRO
         ENDLOOP
         COPY  PPFGBLCO
         AIF   ('&PF_NEST(&PF_NI)'(3,1) EQ 'P').CALLEND
         BC    15,&PF_LIND(&PF_LI-3)
&PF_LIND(&PF_LI)   EQU   *
.CALLEND ANOP
&PF_NEST(&PF_NI)   SETC  '   '.'&PF_NEST(&PF_NI)'(4,5)
         POPINS    &PF_ST(&PF_NI)
&PF_II   SETA  &PF_II-1
&PF_LI   SETA  &PF_LI-3
         MEND
**********************************************************************
         MACRO
         ENDSRCH
         COPY  PPFGBLCO
         POPNEST   SRCH
&PF_LIND(&PF_LI)   EQU   *
&PF_LI   SETA  &PF_LI-1
         MEND
**********************************************************************
         MACRO
         CASENTRY  &P1,&VECTOR=,&POWER=0
         COPY  PPFGBLCO
         PUSHNEST  CASE
         PUSHLAB
         PUSHLAB
         AIF   (&PF_AI GE 50).OVER
&PF_AI   SETA  &PF_AI+1
&PF_AIND(&PF_AI)   SETA  0
&PF_RIND(&PF_AI)   SETC  '&P1'
&PF_MULT(&PF_AI)   SETA  1
&PF_CTR  SETA &POWER
.SHIFTLP AIF   (&PF_CTR LE 0).GENSHFT
&PF_MULT(&PF_AI)   SETA  &PF_MULT(&PF_AI)+&PF_MULT(&PF_AI)
&PF_CTR  SETA  &PF_CTR-1
         AGO   .SHIFTLP
.GENSHFT AIF   (&PF_MULT(&PF_AI) EQ 4).TESTVEC
         AIF   (&PF_MULT(&PF_AI) GT 4).RTSHIFT
         SLA   &P1,2-&POWER
         AGO   .TESTVEC
.RTSHIFT SRA   &P1,&POWER-2
.TESTVEC AIF   ('&VECTOR' EQ 'B' OR '&VECTOR' EQ 'BR').BRVEC
         PUSHLAB
         A     &P1,&PF_LIND(&PF_LI)
         L     &P1,0(&P1)
         BR    &P1
&PF_LIND(&PF_LI)   DC    A(&PF_LIND(&PF_LI-2))
```

```
&PF_LI   SETA  &PF_LI-1
         MEXIT
.BRVEC   BC    15,&PF_LIND(&PF_LI-1)(&P1)
&PF_NEST(&PF_NI)   SETC  '   B'.'&PF_NEST(&PF_NI)'(5,4)
         MEXIT
.OVER    MNOTE 8,'TOTAL CASES STK EXCEEDED. FURTHER EXPANSIONS INVALID'
         MEND
**********************************************************************
         MACRO
         CASE
         COPY  PPFGBLCO
         LCLA  &NBR,&CASENO
         PUSHLAB
         AIF   (N'&SYSLIST EQ 1).LDSUBL
&NBR     SETA  N'&SYSLIST
         AGO   .LDAIND
.LDSUBL  ANOP
&NBR     SETA  N'&SYSLIST(1)
.LDAIND  AIF   (&NBR LE 0).NOPRMS
&PF_AIND(&PF_AI)   SETA  &PF_AIND(&PF_AI)+&NBR
.TSTSUBL AIF   (T'&SYSLIST(1,2) EQ 'O' AND &NBR NE 1).NOTSUBL
&CASENO  SETA  &SYSLIST(1,&NBR)
         AGO  .TSTMULT
.NOTSUBL ANOP
&CASENO  SETA  &SYSLIST(&NBR)
.TSTMULT AIF   (&CASENO-(&CASENO/&PF_MULT(&PF_AI))*&PF_MULT(&PF_AI) NE +
               0).NOTMULT
         AIF   (&CASENO EQ 0).NOTMULT
         AIF   (&PF_CI GE 200).OVER
&PF_CI   SETA  &PF_CI+1
&PF_CIND1(&PF_CI)  SETA  &CASENO
&PF_CIND2(&PF_CI)  SETC  '&PF_LIND(&PF_LI)'
.RETRNPT ANOP
&NBR     SETA  &NBR-1
         AIF   (&NBR NE 0).TSTSUBL
.FRSTIME AIF   ('&PF_NEST(&PF_NI)'(3,1) NE ' ').BCGEN1
&PF_NEST(&PF_NI)   SETC  '  Y'.'&PF_NEST(&PF_NI)'(4,5)
         AGO   .EQUGN1
.BCGEN1  AIF   ('&PF_NEST(&PF_NI)'(4,1) EQ 'B').BCINST
         L     &PF_RIND(&PF_AI),&PF_LIND(&PF_LI-2)
         BR    &PF_RIND(&PF_AI)
         AGO   .EQUGN1
.BCINST  B     &PF_LIND(&PF_LI-1)
.EQUGN1  ANOP
&PF_LIND(&PF_LI)   EQU   *
&PF_LI   SETA  &PF_LI-1
         MEXIT
.NOTMULT MNOTE 8,'CASE &CASENO DELETED. NOT MULTIPLE OF &PF_MULT(&PF_AI+
               ).'
&PF_AIND(&PF_AI)   SETA  &PF_AIND(&PF_AI)-1
```

```
         AGO   .RETRNPT
.NOPRMS  MNOTE 'NO PARAMETERS FOUND WITH CASE MACRO'
         AGO   .FRSTIME
.OVER    MNOTE 8,'CASE NUMBER STK EXCEEDED. FURTHER EXPANSIONS INVALID'
         MEND
**********************************************************************
         MACRO
         ENDCASE
         COPY  PPFGBLCO
         ACTR  99999
         LCLA  &K,&I
         AIF   ('&PF_NEST(&PF_NI)'(4,1) EQ 'B').BVECT1
         L     &PF_RIND(&PF_AI),&PF_LIND(&PF_LI-1)
         BR    &PF_RIND(&PF_AI)
&PF_LIND(&PF_LI-1) DC    A(&PF_LIND(&PF_LI))
         AGO   .BLDVECT
.BVECT1  ANOP
&PF_LIND(&PF_LI-1) B     &PF_LIND(&PF_LI)
.BLDVECT AIF   (&PF_AIND(&PF_AI) LE 0).TESTCI
&K       SETA  &PF_MULT(&PF_AI)
.LOOPIN  ANOP
&I       SETA  1
.LOOP1   AIF   (&K EQ &PF_CIND1(&PF_CI-&I+1)).ELEND
         AIF   (&I EQ &PF_AIND(&PF_AI)).GENTRY
&I       SETA  &I+1
         AGO   .LOOP1
.GENTRY  AIF   ('&PF_NEST(&PF_NI)'(4,1) EQ 'B').BVECT2
         DC    A(&PF_LIND(&PF_LI))
         AGO   .INCRK
.ELEND   AIF   ('&PF_NEST(&PF_NI)'(4,1) EQ 'B').BVECT3
         DC    A(&PF_CIND2(&PF_CI-&I+1))
         AGO   .DECSTK
.BVECT3  B     &PF_CIND2(&PF_CI-&I+1)
.DECSTK  ANOP
&PF_AIND(&PF_AI)   SETA  &PF_AIND(&PF_AI)-1
&PF_CI   SETA  &PF_CI-1
         AIF   (&PF_AIND(&PF_AI) EQ 0).TESTCI
.LOOP2   AIF   (&I EQ 1).INCRK
&I       SETA &I-1
&PF_CIND1(&PF_CI-&I+1)   SETA  &PF_CIND1(&PF_CI-&I+2)
&PF_CIND2(&PF_CI-&I+1)   SETC  '&PF_CIND2(&PF_CI-&I+2)'
         AGO   .LOOP2
.BVECT2  B     &PF_LIND(&PF_LI)
.INCRK   ANOP
&K       SETA  &K+&PF_MULT(&PF_AI)
         AGO   .LOOPIN
.TESTCI  AIF   (&PF_CI LT 0).ASTKERR
&PF_LIND(&PF_LI)   EQU   *
&PF_LI   SETA  &PF_LI-2
&PF_AI   SETA  &PF_AI-1
```

```
         POPNEST   CASE
         AIF   (&PF_AI LT 0).ASTKERR
         MEXIT
.ASTKERR MNOTE 8,'NEGATIVE CASE MACRO STACK PTR. EXPANSION INVALID.'
         MEND
**********************************************************************
         MACRO
         SELECT    &EVERY
         COPY  PPFGBLC0
         GBLC  &PF_ESEL(50)
         GBLB  &PF_EVRY(50)
         PUSHNEST  SEL
&PF_EVRY(&PF_NI)   SETB  ('&EVERY' EQ 'EVERY')
&PF_ESEL(&PF_NI)   SETC  ''
         MEND
**********************************************************************
         MACRO
         WHEN  &P1,&P2,&P3,&P4,&P5,&P6,&P7,&P8,&P9,&P10,&P11,&P12,&P13,X
               &P14,&P15,&P16,&P17,&P18,&P19,&P20,&P21,&P22,&P23,&P24, X
               &P25,&P26,&P27,&P28,&P29,&P30,&P31,&P32,&P33,&P34,&P35, X
               &P36,&P37,&P38,&P39,&P40,&P41,&P42,&P43,&P44,&P45,&P46, X
               &P47,&P48,&P49,&P50,&CC=
         COPY  PPFGBLC0
         GBLC  &PF_ESEL(50)
         GBLB  &PF_EVRY(50)
         GBLB  &PF_NONSELD(50)
         AIF   ('&PF_ESEL(&PF_NI)' NE '').TSTEVRY
         AIF   ('&P1' EQ 'NONE').NONE1ST
         PUSHLAB
&PF_ESEL(&PF_NI)   SETC  '&PF_LIND(&PF_LI)'
         AGO   .BYPASS
.TSTEVRY AIF   (&PF_EVRY(&PF_NI)).TSTNON
         B     &PF_ESEL(&PF_NI)
         AGO   .CONT
.TSTNON  AIF   ('&P1' EQ 'NONE').BADEVRY
.CONT    ANOP
&PF_LIND(&PF_LI)   EQU   *
&PF_LI   SETA  &PF_LI-1
         AIF   (&PF_NONSELD(&PF_NI)).BADWHEN
.BYPASS  ANOP
&PF_NONSELD(&PF_NI)      SETB  ('&P1' EQ 'NONE')
         AIF   ('&P1' NE 'NONE').DOIF
         AIF   (&PF_EVRY(&PF_NI)).BADEVRY
         PUSHNEST  IF
         PUSHLAB
         AGO   .EXIT
.DOIF    PUSHNEST  IF
         PUSHLAB
         IFPROC    &CC,&P1,&P2,&P3,&P4,&P5,&P6,&P7,&P8,&P9,&P10,&P11,  X
               &P12,&P13,&P14,&P15,&P16,&P17,&P18,&P19,&P20,&P21,&P22, X
               &P23,&P24,&P25,&P26,&P27,&P28,&P29,&P30,&P31,&P32,&P33, X
               &P34,&P35,&P36,&P37,&P38,&P39,&P40,&P41,&P42,&P43,&P44, X
               &P45,&P46,&P47,&P48,&P49,&P50
.EXIT    POPNEST   IF
         MEXIT
.NONE1ST MNOTE 8,'''NONE'' INVALID IN THE FIRST WHEN OF A SELECT STRUCT+
               URE'
         MEXIT
.BADEVRY MNOTE 8,'''NONE'' OPTION INVALID WITH ''SELECT EVERY'''
         MEXIT
.BADWHEN MNOTE 8,'NO WHEN STATEMENT ALLOWED AFTER ''WHEN NONE'''
         MEND
***********************************************************************
         MACRO
         ENDSEL
         COPY  PPFGBLC0
         GBLC  &PF_ESEL(50)
         GBLB  &PF_EVRY(50)
         GBLB  &PF_NONSELD(50)
         AIF   (&PF_EVRY(&PF_NI)).ISEVRY
&PF_ESEL(&PF_NI)   EQU   *
         AIF   (&PF_NONSELD(&PF_NI)).DONE
.ISEVRY  ANOP
&PF_LIND(&PF_LI)   EQU   *
.DONE    POPNEST   SEL
&PF_LI   SETA  &PF_LI-2
         MEND
```

PPFGBLC0 – COPYBOOK WITH GLOBAL MACROS USED BY PPFC14M0

```
         GBLA  &PF_CCVAL                COND CODE VARIABLE
         GBLA  &PF_CTR                  MACRO PARAMETER COUNTER
         GBLA  &PF_SEQ                  LABEL NUMBER GENERATOR
         GBLA  &PF_AI                   INDEX FOR TOTAL NO. CASES STK
         GBLA  &PF_CI                   INDEX FOR CASE AND LBL NO. STKS
         GBLA  &PF_II                   PTR TO INST STKS
         GBLA  &PF_LI                   INDEX FOR LABEL NUMBER STK
         GBLA  &PF_NI                   PTR TO NEST STK
         GBLA  &PF_AIND(50)             TOTAL CASES STK
         GBLA  &PF_CIND1(200)           CASE NUMBER STK
         GBLA  &PF_MULT(50)             CASE NUMBER MULTIPLIER
         GBLA  &PF_ST(51)               INST STK INCREASE AT EACH LEVEL
         GBLC  &PF_CIND2(200)           LABEL NUMBER STK FOR CASES
         GBLC  &PF_IIND1(100)           INSTRUCTION STK 1
         GBLC  &PF_IIND2(100)           INSTRUCTION STK 2
.*       GBLC  &PF_I22(100)             INSTRUCTION STK 2, 2ND PART
.*       GBLC  &PF_I23(100)             INSTRUCTION STK 2, 3RD PART
.*       GBLC  &PF_I24(100)             INSTRUCTION STK 2, 4TH PART
```

```
         GBLC  &PF_IIND3(100)          INSTRUCTION STK 3
.*       GBLC  &PF_I32(100)            INSTRUCTION STK 3, 2ND PART
.*       GBLC  &PF_I33(100)            INSTRUCTION STK 3, 3RD PART
.*       GBLC  &PF_I34(100)            INSTRUCTION STK 3, 4TH PART
         GBLC  &PF_IIND4(100)          INSTRUCTION STK 4
.*       GBLC  &PF_I42(100)            INSTRUCTION STK 4, 2ND PART
.*       GBLC  &PF_I43(100)            INSTRUCTION STK 4, 3RD PART
         GBLC  &PF_IIND5(100)          INSTRUCTION NAME STACK
         GBLC  &PF_LIND(101)           LABEL NUMBER STK
         GBLC  &PF_NEST(50)            NESTING STK
         GBLC  &PF_RIND(50)            REG STK FOR CASENTRY MACRO
```

REGEQU – A MACRO TO DEFINE REGISTER EQUATES

```
         MACRO
         REGEQU
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
.* ACCESS REGISTERS
AR0      EQU   0
AR1      EQU   1
AR2      EQU   2
AR3      EQU   3
AR4      EQU   4
AR5      EQU   5
AR6      EQU   6
AR7      EQU   7
AR8      EQU   8
AR9      EQU   9
AR10     EQU   10
AR11     EQU   11
AR12     EQU   12
AR13     EQU   13
AR14     EQU   14
AR15     EQU   15
```

```
.* FLOATING POINT REGISTERS
FPR0     EQU   0
FPR2     EQU   2
FPR4     EQU   4
FPR6     EQU   6
         MEND
```

Pieter Wiid
Senior Systems Engineer
Outsource (South Africa)

More PCF programming in Java

INTRODUCTION

In my previous article for *MQ Update* (*PCF programming in Java*, see Issue 3), I started laying out a basic framework for systems management of MQSeries using PCF commands from the Java environment. This article continues the exploration of PCF programming in Java by presenting wrappers for several common systems management operations.

You often find with MQSeries that the number of options that have to be specified when carrying out everyday operations can be confusing. These wrappers, therefore, aim to simplify these operations by applying common default values to the process. While this doesn't cover all systems management scenarios, it greatly simplifies the development of systems management utilities. Besides, the classes supplied can easily be extended to cover missing areas.

THE APPROACH TAKEN

As before, channels are the focus of systems management, since they are more complex and error-prone than queues or queue managers from a systems management perspective. That being said, this article also includes one class to simplify the interaction with queues.

In my experience, once a problem is detected, several steps are required to rectify it (and the subsequent problems that it may have caused). Detection of channel problems was covered previously, and the automation of problem recovery is now examined from the standpoint of the following common scenarios:

1 A channel has stopped as a result of its normal time out. This is not a problem, and the 'Inactive' channel state that results from this should not be confused with a 'Stopped' or 'Retry' state. This channel should be 'Pingable' and a restart should be possible. Of course, if the system involved does not employ transmission queue triggering to start channels, then a manual restart is required.

2 A channel has stopped as a result of a transient network problem. Theoretically, the channel should be usable once the network problem is resolved. However, this is often not the case, as a number of actions are taken by MQSeries to prevent message loss. These actions may result in message sequence number mismatches, transmission queue trigger/get disabling, and in-doubt messages. Without intervention by systems management software or manual intervention, no further message transmission will occur.

3 A channel has stopped as a result of a persistent network problem. This will eventually lead to the sort of problems described in the second scenario above. Network issues must, of course, be resolved first.

4 A channel has stopped as a result of problems with the partner queue manager. Problems in this area include the queue manager being stopped (either normally or as a consequence of operating system failures), MQ software errors, log files (or disks or page sets) becoming full, and general system stress. Problems in this area are usually too diverse for systems management software to tackle in a consistent way.

GENERAL ACTIONS

In general, problems in a correctly configured and properly managed system tend to fall into the second and third categories above. It is, therefore, possible to define a simple set of rules that may be implemented as a program using a language such as Java and the MQSeries PCF interface. These rules may not be the most efficient way of solving the problem, but if they can tackle even a modest proportion of situations without human intervention, they should prove immensely valuable.

A set of MQSeries management rules for an on-line system may take the following form:

1 If any channel is in a state other than ‘Running’, and there are messages on its transmission queue, then it is considered to be a problem.

2 If any problem persists after an attempt is made at correcting it (or after a series of attempts or a pre-defined period of time), then an alert is raised. Alerts, in this context, are intended for human beings and may take the form of SNMP alerts (for display on, say, a NetView console), operating system alerts, e-mail messages, etc.

3 If the systems management software has lost connectivity to one of the two queue managers in a channel connection, then assume that the network is failing. In this scenario, the only course of action that’s often available is to wait for recovery and retry, as it may be impossible even to transmit alerts to users. Standard network management software should come into play here.

4 If you have connectivity with both queue managers, try to kick-start the channel in the following way:

   Channel ‘kick-start’ operations:

   1 Stop both the sender and receiver pair (or server and requester).

   2 If the transmission queue has become ‘get-disabled’, re-enable it.

   3 If triggering is disabled, re-enable it.

   4 If the message sequence numbers of channels are out of sync, reset them.

5 Start the sender (or requester) channel.

This should fix a good proportion of problems. If the channel remains in a non-functional state after this procedure, human intervention is probably required. The steps outlined above all take a small amount of time to perform. If your system is an on-line, non-persistent system, it may be desirable to skip the checks outlined in steps 2, 3, and (possibly) 4. While this may result in a substantial increase in activity to re-enable and reset channels, these operations should not damage the system as a whole.

In order to perform the above operations, I've written the Java classes that are presented next.

SOURCE CODE FOR THE CLASSES WRITTEN.

When writing utility classes, it is usually necessary to provide many alternative ways to use the code. This is true in the following classes. For readers interested only in the MQSeries, nearly all MQ code is implemented in the last method(s) of each of the classes.

These classes are utility code, and are not executable in isolation. They require the 'IBM PCF for Java' support pack, IBM MQSeries client for Java, and a simple program to call them. The calling program may look something like the example below.

SAMPLE_UTILITY.JAVA

```
import com.dmitri.pcf.*
import com.ibm.mq.pcf.*;
import com.ibm.mq.*;

public class TestClass {

/**
*   For testing only.
*/
public static void main(String[] args) {

try {
    PCFAgent iAgent = new PCFAgent("localhost", 1414,
                                  "SYSTEM.DEF.SVRCONN");

    EnquireQueuePCF.enableQueueGet("SYSTEM.DEFAULT.LOCAL.QUEUE",
                                iAgent);
    iAgent.disconnect();

  } catch (Exception ex) {
    ex.printStackTrace();
  }
}
}
```

STARTCHANNELPCF

This class uses the *MQCMD_START_CHANNEL* PCF command to start the channel supplied. A valid agent connection to the sender/ requester channel must be present. Responses to this command are supplied either in PCF return code format or in plain text that an operator can understand.

STARTCHANNELPCF.JAVA

```
package com.dmitri.pcf;
import com.ibm.mq.pcf.*;
import com.ibm.mq.*;

/**
*   @author Dmitri
*
*  This class is used to start channels. The class uses various
*  responses, including static and dynamic ones, and ones that
*  are human-oriented and machine-oriented.
*/
public class StartChannelPCF implements CMQCFC {

  private PCFAgent iAgent;

/**
*  Constructor, this requires a valid queue manager agent.
*/
  public StartChannelPCF(PCFAgent agent) {

    iAgent = agent;
  }

/**
*  This method starts the channel specified. The result is returned
*  in a format that is suitable for displaying to users.
*/
  public String startChannelString(String channelName) {
    return startChannelString(channelName, iAgent);
  }

/**
*  This method starts the channel specified. The result is returned
*  as an integer value for further processing.
*/
  public int startChannel(String channelName) {

    return startChannel(channelName, iAgent);
  }

/**
*  This method starts the channel specified. The result is returned
*  in a format suitable for displaying to users. Static version.
*/
  public static String startChannelString(String channelName,
                                          PCFAgent agent) {

    int reason = startChannel(channelName, agent);
    if (reason == 0) {
      return("Channel " + channelName + " Started");
    }
    return("Channel " + channelName +
           " failed to start, reason code : " + reason);
  }

/**
*  This method starts the channel specified. The result is returned
*  as an integer for further processing. Static version, actual
*  call implemented here.
*/
  public static int startChannel(String channelName, PCFAgent agent) {

    PCFParameter [] parameters = new PCFParameter [] {
            new MQCFST (MQCACH_CHANNEL_NAME, channelName),
            };

    PCFHashtable response = null;

    try {
      MQMessage [] pcfResponses = agent.send (MQCMD_START_CHANNEL,
                                              parameters);
      response = new PCFHashtable(pcfResponses[0]);
                                      // assume only the one response.

      if (response.isValid()) {
        return 0;
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
    return response.getReasonCode();
  }
}
```

STOPCHANNELPCF

This class uses the MQCMD_STOP_CHANNEL command to force a channel to stop. ‘Quiesce stops’ are available, but have not been implemented, as the quiescing of channels often fails under systems management error conditions.

STOPCHANNELPCF.JAVA

```
package com.dmitri.pcf;
import com.ibm.mq.pcf.*;
import com.ibm.mq.*;

/**
*   @author Dmitri
*
*  This class is used to stop channels. Various methods are used
*  to achieve this.
*/
public class StopChannelPCF implements CMQCFC {

  private PCFAgent iAgent;

/**
*  Constructor. This requires a valid queue manager agent (from
*  the com.ibm.mq.pcf package).
*/
  public StopChannelPCF(PCFAgent agent) {

    iAgent = agent;
  }

/**
*  This method stops the channel specified. The result is returned
*  in plain text that may be displayed for an operator to read.
*/
  public String stopChannelString(String channelName) {
    return stopChannelString(channelName, iAgent);
  }
```

```
/**
*  This method stops the channel specified. The result is returned
*  in plain text that may be displayed for an operator to read.
*  Static version
*/
  public static String stopChannelString(String channelName,
                                         PCFAgent agent) {

    int reason = stopChannel(channelName, agent);

    if (reason == 0) {
      return("Channel " + channelName + " Stopped Successfully");
    }
    return("Channel " + channelName +
           " failed to stop, reason code : " + reason);
  }

/**
*  This method stops the channel specified. The result is returned
*  in code format suitable for further processing.
*/
  public int stopChannel(String channelName) {

    return stopChannel(channelName, iAgent);
  }

/**
*  This method stops the channel specified. The result is returned
*  in code format suitable for further processing. An alternative
*  static version actually makes the call.
*/
  public static int stopChannel(String channelName, PCFAgent agent) {

    PCFParameter [] parameters = new PCFParameter [] {
            new MQCFST (MQCACH_CHANNEL_NAME, channelName),
            new MQCFIN (MQIACF_QUIESCE, MQQO_NO ),
                                    // always force channel closed.
            };

    PCFHashtable response = null;

    try {
      MQMessage [] pcfResponses = agent.send (MQCMD_STOP_CHANNEL,
                                              parameters);
      response = new PCFHashtable(pcfResponses[0]);
                                  // assume only the one response.

      if (response.isValid()) {
        return 0;
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
    return response.getReasonCode();
  }
}
```

RESETCHANNELPCF

This class uses the MQCMD_RESET_CHANNEL PCF command to reset a channel's message sequence number to zero. This is an example of how wrappers are able to simplify MQSeries operations by using defaults for optional values. If it was required that message sequence numbers were reset to alternative numbers, then the PCF command array would require the following additional parameter:

```
     new MQCFIN (MQIACH_MSG_SEQUENCE_NUMBER, sequenceNumber)
```

RESETCHANNELPCF.JAVA

```
package com.dmitri.pcf;
import com.ibm.mq.pcf.*;
import com.ibm.mq.*;

/**
*   @author Dmitri
*
*  This class is used to reset channels. Again, various methods
*  are provided.
*/
public class ResetChannelPCF implements CMQCFC {

  private PCFAgent iAgent;

/**
*  Constructor, this requires a valid queue manager agent.
*/
  public ResetChannelPCF(PCFAgent agent) {

    iAgent = agent;
  }

/**
*  This method resets the channel's message sequence number to 0,
*  which means that the next message will be message 1. The result
*  is returned in plain text that may be read by an operator.
*/
```

```
  public String resetChannelString(String channelName) {

    return resetChannelString(channelName, iAgent);
  }

/**
*  This method resets the channel's message sequence number to 0,
*  which means that the next message will be message 1. The result
*  is returned as an integer for further processing.
*/
  public int resetChannel(String channelName) {

    return resetChannel(channelName, iAgent);
  }

/**
*  This method resets the channel's message sequence number to 0,
*  which means that the next message will be message 1. The result
*  is returned in plain text that may be read by an operator.
*  Static version.
*/
  public static String resetChannelString(String channelName,
                                          PCFAgent agent) {

    int reason = resetChannel(channelName, agent);

    if (reason == 0) {
      return("Channel " + channelName + " Reset");
    }
    return("Channel " + channelName +
           " failed to reset, reason code : " + reason );
  }

/**
*  This method resets the channel's message sequence number to 0,
*  which means that the next message will be message 1. While MQSeries
*  allows any number to be specified as the new MSN, I prefer just to
*  reset both sides of a channel to 0. This is the static method
*  that actually makes the calls.
*/
  public static int resetChannel(String channelName, PCFAgent agent) {

    PCFParameter [] parameters = new PCFParameter [] {
            new MQCFST (MQCACH_CHANNEL_NAME, channelName),
            };

    PCFHashtable response = null;

    try {
      MQMessage [] pcfResponses = agent.send (MQCMD_RESET_CHANNEL,
                                              parameters);
      response = new PCFHashtable(pcfResponses[0]);
                                 // assume only the one response.

      if (response.isValid()) {
        return 0;
      }

    } catch (Exception e) {
      e.printStackTrace();
    }
    return response.getReasonCode();
  }
}
```

PINGCHANNELPCF

This class performs a simple operation. It’s not included for use in recovery from problems, but for use by systems administrators in manually testing whether a channel has network connectivity, etc. This class uses MQCMD_PING_CHANNEL PCF command and is in most respects very similar to the class for starting channels.

PINGCHANNELPCF.JAVA

```
package com.dmitri.pcf;
import com.ibm.mq.pcf.*;
import com.ibm.mq.*;

/**
*   @author Dmitri
*
*  This class is used to ping channels. The ping function may be
*  used to check communications to channels. A pingable channel is
*  not necessarily suitable for the transmission of messages,
*  though, as other factors, such as sequence numbers, may prevent
*  transmission.
*/
public class PingChannelPCF implements CMQCFC {

  private PCFAgent iAgent;

/**
*  Constructor, this requires a valid queue manager agent.
*/
  public PingChannelPCF(PCFAgent agent) {
    iAgent = agent;
  }

/**
*  This method pings the channel specified. The result returned
*  by this method is suitable for further processing.
*/
  public int pingChannel(String channelName) {

    return pingChannel(channelName, iAgent);
  }

/**
*  This method pings the channel specified. The string returned
*  contains a general description of the outcome. This is in plain
*  text and is aimed at humans, not other processes.
*/
  public String pingChannelString(String channelName) {

    return pingChannelString(channelName, iAgent);
  }

/**
*  This method pings the channel specified. The string returned
*  contains a general description of the outcome. This is in plain
*  text and is aimed at humans, not other processes. Static version.
*/
  public static String pingChannelString(String channelName,
                                         PCFAgent agent) {

    int reason = pingChannel(channelName, agent);

    if (reason == 0) {
      return("Channel " + channelName + " Pinged Sucessfully");
    }
    return("Channel " + channelName +
           " failed to ping, reason code : " + reason);
  }

/**
*  This static method pings the channel specified. The result is
*  suitable for further processing.
*/
  public static int pingChannel(String channelName, PCFAgent agent) {

    PCFParameter [] iParameters = new PCFParameter [] {
            new MQCFST (MQCACH_CHANNEL_NAME, channelName),
            };

    PCFHashtable response = null;
    try {
      MQMessage [] pcfResponses = agent.send (MQCMD_PING_CHANNEL,
                                              iParameters);
      response = new PCFHashtable(pcfResponses[0]);
                                  // assume only the one response.

      if (response.isValid()) {
        return 0;
      }

    } catch (Exception e) {
      e.printStackTrace();
    }
    return response.getReasonCode();
  }
}
```

ENQUIREQUEUEPCF

The enquire queue class has two main purposes. Firstly, it fetches information about a named queue (or all queues) and provides a shortcut method for querying queue depth. Secondly, it provides a method to re-enable both message GETs from the queue and triggering. Together, these two operations cover most of what is required to fix problems from the queue's perspective.

The following commands are used to implement these functions:

- MQCMD_INQUIRE_Q
- MQCMD_CHANGE_Q

These two functions may be used to enquire on or alter just about any attribute of a queue (for instance, MQIA_MAX_Q_DEPTH, MQIA_SHAREABILITY, MQIA_TRIGGER_TYPE, etc), though they are used in a fairly limited way here.

ENQUIREQUEUEPCF.JAVA

```
package com.dmitri.pcf;
import com.ibm.mq.pcf.*;
import com.ibm.mq.*;

/**
*   @author Dmitri
*
```

```
*  This class is used to query queue information. Queue depth is
*  of particular interest when the required queue is a transmission
*  queue (xmitq). In addition to querying, it is also possible to
*  re-enable (GET enable) queues using this class. This is useful
*  when developing systems management software, as channels often
*  disable their transmission queues when problems are encountered.
*/
public class EnquireQueuePCF implements CMQCFC, CMQC {

  private PCFAgent iAgent;

/**
*  The constructor. This requires a valid queue manager agent. It may
*  then be used to query any queue belonging to the queue manager.
*/
  public EnquireQueuePCF(PCFAgent agent) {

    iAgent = agent;
  }

/**
*  This method returns details of the queue in question.
*/
  public PCFHashtable queueDetails(String queueName) {

    return queueDetails(queueName, iAgent);
  }

/**
*  This method extracts the queue depth from the supplied queue
*  details.
*/
  public static int getQueueDepth(PCFHashtable details) {

    return details.getIntValue(MQIA_CURRENT_Q_DEPTH);
  }

/**
*  This method returns details of all queues belonging to the current
*  queue manager.
*/
  public PCFHashtable [] getAllQueues() {

    return getAllQueues(iAgent);
  }

/**
*  GET-enable a queue. The return value is zero for success,
*  positive for MQSeries errors, and negative for general errors.
*/
```

```
  public int enableQueueGet(String queueName) {
    return enableQueueGet(queueName, iAgent);
  }

/**
*  Re-enable queue triggering. The return value is zero for success,
*  positive for MQSeries errors, and negative for general errors.
*/
  public int enableQueueTriggering(String queueName) {
    return enableQueueTriggering(queueName, iAgent);
  }

/**
*  GET-enable a queue. This method may be used as a template for
*  other queue modification methods. The method returns zero if
*  successful.
*/
  public static int enableQueueGet(String queueName, PCFAgent agent) {

    try {

      // Check that it's appropriate to re-enable the queue.

      PCFHashtable details = queueDetails(queueName, agent);
      int queueType = details.getIntValue(MQIA_Q_TYPE);
      if (queueType != MQQT_ALIAS && queueType != MQQT_LOCAL) {
        return -1;  // not valid for remote queues etc.
      }

      if (details.getIntValue(MQIA_INHIBIT_GET) == MQQA_GET_ALLOWED) {
        return 0;  // nothing further to do, it's already GET-enabled.
      }

      PCFParameter [] parameters = new PCFParameter [] {
              new MQCFST (MQCA_Q_NAME, queueName),
              new MQCFIN (MQIA_Q_TYPE, queueType),
              new MQCFIN (MQIA_INHIBIT_GET, MQQA_GET_ALLOWED),
              };

      MQMessage [] pcfResponses = agent.send (MQCMD_CHANGE_Q,
                                              parameters);
      PCFHashtable response = new PCFHashtable(pcfResponses[0]);
                                    // assume only the one response.
      if (response.isValid()) {
        return 0;  // successful call.
      } else {
        return response.getReasonCode();
      }

    } catch (Exception ex) {
      ex.printStackTrace();
    }
    return -1;
  }

/**
*  Re-enables queue triggering. Returns zero if successful.
*/
  public static int enableQueueTriggering(String queueName,
                                          PCFAgent agent) {

    try {

      // Check that it's appropriate to re-enable the queue.

      PCFHashtable details = queueDetails(queueName, agent);
      int queueType = details.getIntValue(MQIA_Q_TYPE);
      if (queueType != MQQT_LOCAL) {
        return -1;  // only valid for local queues.
      }

      if (details.getIntValue(MQIA_TRIGGER_CONTROL) == MQTC_ON) {
        return 0;
               // nothing further to do, it's already trigger-enabled.
      }

      PCFParameter [] parameters = new PCFParameter [] {
              new MQCFST (MQCA_Q_NAME, queueName),
              new MQCFIN (MQIA_Q_TYPE, queueType),
              new MQCFIN (MQIA_TRIGGER_CONTROL, MQTC_ON),
              };

      MQMessage [] pcfResponses = agent.send (MQCMD_CHANGE_Q,
                                              parameters);
      PCFHashtable response = new PCFHashtable(pcfResponses[0]);
                                    // assume only the one response.
      if (response.isValid()) {
        return 0;  // successful call.
      } else {
        return response.getReasonCode();
      }

    } catch (Exception ex) {
      ex.printStackTrace();
    }
    return -1;
  }

/**
*   An alternative static queue enquiry method.
```

```
*/
  public static PCFHashtable queueDetails(String queueName,
                                        PCFAgent agent) {

    PCFParameter [] parameters = new PCFParameter [] {
            new MQCFST (MQCA_Q_NAME, queueName),
            };

    try {
      MQMessage [] pcfResponses = agent.send (MQCMD_INQUIRE_Q,
                                            parameters);

      PCFHashtable response = new PCFHashtable(pcfResponses[0]);
                                  // assume only the one response.
      return response;

    } catch (Exception e) {
      e.printStackTrace();
    }
    return null;
  }

/**
*  Alternative static version of the getAllQueues method.
*/
  public static PCFHashtable [] getAllQueues(PCFAgent agent) {

    try {
      PCFParameter [] parameters = new PCFParameter [] {
              new MQCFST (MQCA_Q_NAME, "*"),
              };

      MQMessage [] pcfResponses = agent.send (MQCMD_INQUIRE_Q,
                                            parameters);
      PCFHashtable [] details = new PCFHashtable [pcfResponses.length];

      for (int i = 0; i < pcfResponses.length; i++) {
        details[i] = new PCFHashtable(pcfResponses[i]);
      }
      return details;

    } catch (Exception e) {
      e.printStackTrace();
    }
    return null;
  }
}
```

PCFHASHTABLE

This utility wrapper class is created using a response message (for instance, the response from a ‘channel details’ inquiry), and then walks through a message and splits its components (such as MQCFINs or MQCFSTs) into entries in a hash table. Other classes are then able to query values without knowledge of parameter order. For example, if *iChannelDetails* is an instance of *PCFHashtable* (again created in response to a channel details query), the following code can be used to retrieve the channel type:

```
     int chltype = iChannelDetails.getIntValue(MQIACH_CHANNEL_TYPE);
```

PCFHASHTABLE.JAVA

```
package com.dmitri.pcf;

import com.ibm.mq.pcf.*;
import com.ibm.mq.*;
import java.util.*;
/**
*  @author Dmitri
*
*  This class extends Hashtable to provide PCF-specific processing.
*/
public class PCFHashtable extends Hashtable {

  private int iReasonCode;

/**
*  Constructor that sets up the data based on a message.
*/
  PCFHashtable(MQMessage message) {

    super();
    try {
      MQCFH cfh = new MQCFH(message);
      iReasonCode = cfh.reason;

      if (isValid()) {

        PCFParameter p;
        for (int i = 0; i < cfh.parameterCount; i++) {

          // Walk through the returned attributes
          p = PCFParameter.nextParameter (message);
          Integer key = new Integer(p.getParameter());
          put(key, p.getValue());
        }
      }
    } catch (Exception ex) {
      System.out.println(ex);
    }
  }

/**
*  Is the PCF request valid and, hence, may it be used?
*/
  public boolean isValid() {
    return iReasonCode == 0;
  }

/**
*  Returns the reason code for use by diagnostic processes.
*/
  public int getReasonCode(){
    return iReasonCode;
  }

/**
*  Returns the int value represented by the key supplied.
*/
  public int getIntValue(int key) {
    Integer i = (Integer) get(new Integer(key));
    return i.intValue();
  }

/**
*  Returns the int array represented by the key supplied.
*/
  public int [] getIntArray(int key) {
    return (int []) get(new Integer(key));
  }

/**
*  Returns the string value represented by the key supplied.
*/
  public String getStringValue(int key) {
    // mq returns padded strings.
    return ((String) get(new Integer(key))).trim();
  }

/**
*  Returns the string array represented by the key supplied.
*/
  public String [] getStringArray(int key) {
    String [] paddedStrings =  (String []) get(new Integer(key));
    String [] trimStrings = new String [paddedStrings.length];
    for (int i = 0; i < paddedStrings.length; i++) {
      trimStrings[i] = paddedStrings[i].trim();
    }
    return trimStrings;
  }
}
```

SUMMARY

Utility classes like these cannot hope to cater for every systems management scenario, nor can they hope to do away with all human intervention. They can, however, reduce the tedium and cost of many systems management activities. Whether you write or buy systems management software for MQSeries, using the techniques described in this article will almost certainly reduce the cost that manual procedures incur. I hope that these examples have reiterated the point that it is not excessively difficult to write your own automation procedures.

A system generator for MQSeries (part 2)

This is the second part of this article on generating an MQ system automatically (the first part appeared in last month's issue of *MQ Update*). The article concludes in next month's issue.

Below is MQSeries Generator's only message definition (add it as a member of *your.msgs.lib*).

MQG00

```
MQG001 'Enter a valid value .' .ALARM=YES
'WRONG !                 '
MQG002 'space for a second message........' .ALARM=YES
'EMPTY!                  '
```

MQSeries Generator skeleton definitions – the following skeleton JCLs are invoked:

MQSDEFA

```
//&USERID.P JOB    (ACCT#),'INSTALL',CLASS=A,MSGCLASS=X,
//          NOTIFY=&USERID
//******************************************************************/
/*ROUTE  XEQ &LPAR
//******************************************************************/
//DEFLIBS  EXEC PGM=IEFBR14
//DD1      DD DISP=(NEW,CATLG,DELETE),
// DSN=&SYSID..SCSQPROC,
// SPACE=(CYL,(1,1,20)),UNIT=SYSDA,VOL=SER=&VOL,
// DCB=(BLKSIZE=3120,RECFM=FB,LRECL=80,DSORG=PO)
//*************************************************
//*    ADD MEMBERS TO SCSQPROC                    *
//*************************************************
//SCSQPROC EXEC PGM=IEBUPDTE,PARM=NEW
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSUT2   DD DISP=SHR,DSN=&SYSID..SCSQPROC
//SYSIN    DD DATA
./  ADD NAME=CSQBDEFV
********************************************************************/
***                                                                */
*** MODULE NAME:  CSQBDEFV                                         */
***                                                                */
*** DESCRIPTIVE NAME = MQM MVS/ESA Batch adapter:                  */
***                    Subsystem definition table                  */
***                                                                */
********************************************************************/
***                    (C) Copyright INTERPAY                      */
********************************************************************/
***                                                                */
*** FUNCTION =   This table defines the MQ default                 */
***              subsystem for a batch application.                */
***              It is generated using CSQBDEF.                    */
***                                                                */
********************************************************************/
*
CSQBDEFV CSECT
*
* -----------------------------------------------------------------*/
*     Define default system for connect.
* -----------------------------------------------------------------*/
         CSQBDEF  NAME=&SYSID
*
         END CSQBDEFV
********************************************************************/
./  ADD NAME=CSQ4INP1
*********************************************************************
*                                                                   *
*                COPYRIGHT INTERPAY                                  *
*                                                                    *
* Status:        Version 1 Release 1                                 *
*                                                                    *
**********************************************************************
*                       MQSeries for MVS/ESA
* CSQINP1 sample
**********************************************************************
*
* Commands to define objects that are not recoverable should be
* specified in a dataset identified by the CSQINP1 DD concatenation
* in the queue manager started task procedure. As these objects are
* not recoverable, they should be defined every time the queue
* manager is started using a dataset referenced by CSQINP1.
*
**********************************************************************
* BUFFPOOL DEFINITIONS
**********************************************************************
*
* Define the buffer pool sizes.

DEFINE BUFFPOOL( 0 ) BUFFERS( 10000 )
DEFINE BUFFPOOL( 1 ) BUFFERS( 10000 )
DEFINE BUFFPOOL( 2 ) BUFFERS( 10000 )
DEFINE BUFFPOOL( 3 ) BUFFERS( 10000 )

*
**********************************************************************
* PAGE SET DEFINITIONS
**********************************************************************
*
* Define pagesets and associate each pageset with a buffer pool.

DEFINE PSID( 00 ) BUFFPOOL( 0 )
DEFINE PSID( 01 ) BUFFPOOL( 1 )
DEFINE PSID( 02 ) BUFFPOOL( 1 )
DEFINE PSID( 03 ) BUFFPOOL( 1 )
DEFINE PSID( 04 ) BUFFPOOL( 1 )
DEFINE PSID( 05 ) BUFFPOOL( 1 )
DEFINE PSID( 06 ) BUFFPOOL( 1 )
DEFINE PSID( 07 ) BUFFPOOL( 1 )
DEFINE PSID( 08 ) BUFFPOOL( 1 )
DEFINE PSID( 09 ) BUFFPOOL( 1 )
DEFINE PSID( 10 ) BUFFPOOL( 1 )

*
**********************************************************************
* OTHER DEFINITIONS
**********************************************************************
*
```

```
* MAXSMSGS definition.

DEFINE MAXSMSGS( 10000 )

*
* SECURITY definition.

ALTER SECURITY TIMEOUT( 54 ) INTERVAL( 12 )

*
*********************************************************************
* End of CSQ4INP1
*********************************************************************
./  ADD NAME=CSQ4INP2
*********************************************************************
*                                                                   *
*********************************************************************
*
*                       MQSeries for MVS/ESA
*                 CSQINP2 sample for system objects
*
*********************************************************************
*
* This sample dataset contains a set of object definitions starting
* with SYSTEM.DEF, SYSTEM.COMMAND, SYSTEM.ADMIN, and &SYSID.
*
* These objects must be created the first time a queue manager is
* started. This is done by including the dataset in the CSQINP2
* DD concatenation in the queue manager started task procedure.
*
* Once objects are successfully created, there is no need to define
* them again if the queue manager is restarted, and so the dataset
* can be removed from the CSQINP2 DD concatenation. If the dataset
* is not removed from CSQINP2, the DEFINEs will fail with an error
* message stating that the object already exists. Alternatively,
* if the definitions are reset on every restart, you can add the
* keyword REPLACE to each command.
*
* Objects in the dataset that start with the reserved word
* SYSTEM cannot be deleted once created. If changes are needed
* to these objects, use ALTER or DEFINE with REPLACE commands.
*
*********************************************************************
* For more information, see the comments below on dead letter queues.
*
*********************************************************************
* SYSTEM.DEF DEFINITIONS
*********************************************************************
*
* SYSTEM.DEF objects are used by queue managers to define the
* default attributes for the following objects:
*
*  LOCAL QUEUES
*  MODEL QUEUES
*  ALIAS QUEUES
*  REMOTE QUEUES
*  PROCESSES
*  NAMELISTS
*  CHANNELS
*
* These are used by DEFINE commands if no LIKE parameter is used.
* The system default object names should not be changed, though
* you may change the default attribute settings, if required.
*
******
DEFINE QLOCAL( 'SYSTEM.DEFAULT.LOCAL.QUEUE' ) +

* Common queue attributes
         DESCR( ' ' ) +
         PUT( ENABLED ) +
         DEFPRTY( 0 ) +
         DEFPSIST( NO ) +

* Local queue attributes
         GET( ENABLED ) +
         NOSHARE +
         DEFSOPT( EXCL ) +
         MSGDLVSQ( PRIORITY ) +
         RETINTVL( 999999999 ) +
         MAXDEPTH( 999999999 ) +
         MAXMSGL( 4194304 ) +
         NOHARDENBO +
         BOTHRESH( 0 ) +
         BOQNAME( ' ' ) +
         STGCLASS( 'DEFAULT' ) +
         USAGE( NORMAL ) +

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

* Trigger attributes
         NOTRIGGER +
         TRIGTYPE( FIRST ) +
         TRIGDPTH( 1 ) +
         TRIGMPRI( 0 ) +
         TRIGDATA( ' ' ) +
         PROCESS( ' ' ) +
         INITQ( ' ' )
*
******
DEFINE QMODEL( 'SYSTEM.DEFAULT.MODEL.QUEUE' ) +

* Common queue attributes
         DESCR( ' ' ) +
         PUT( ENABLED ) +
         DEFPRTY( 0 ) +
         DEFPSIST( NO ) +

* Model queue attributes
         DEFTYPE( TEMPDYN ) +

* Local queue attributes
         GET( ENABLED ) +
         NOSHARE +
         DEFSOPT( EXCL ) +
         MSGDLVSQ( PRIORITY ) +
         RETINTVL( 999999999 ) +
         MAXDEPTH( 999999999 ) +
         MAXMSGL( 4194304 ) +
         NOHARDENBO +
         BOTHRESH( 0 ) +
         BOQNAME( ' ' ) +
         STGCLASS( 'DEFAULT' ) +
         USAGE( NORMAL ) +

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

* Trigger attributes
         NOTRIGGER +
         TRIGTYPE( FIRST ) +
         TRIGDPTH( 1 ) +
         TRIGMPRI( 0 ) +
         TRIGDATA( ' ' ) +
         PROCESS( ' ' ) +
         INITQ( ' ' )
*
******
```

```
DEFINE QALIAS( 'SYSTEM.DEFAULT.ALIAS.QUEUE' ) +

* Common queue attributes
         DESCR( ' ' ) +
         PUT( ENABLED ) +
         DEFPRTY( 0 ) +
         DEFPSIST( NO ) +

* Alias queue attributes
         GET( ENABLED ) +
         TARGQ( ' ' )
*
******
DEFINE QREMOTE( 'SYSTEM.DEFAULT.REMOTE.QUEUE' ) +

* Common queue attributes
         DESCR( ' ' ) +
         PUT( ENABLED ) +
         DEFPRTY( 0 ) +
         DEFPSIST( NO ) +

* Remote queue attributes
         RNAME( ' ' ) +
         RQMNAME( ' ' ) +
         XMITQ( ' ' )
*
******
DEFINE PROCESS( 'SYSTEM.DEFAULT.PROCESS' ) +

* Process attributes
         DESCR( ' ' ) +
         APPLTYPE( CICS ) +
         APPLICID( ' ' ) +
         USERDATA( ' ' ) +
         ENVRDATA( ' ' )
*
******
DEFINE NAMELIST( 'SYSTEM.DEFAULT.NAMELIST' ) +

* Namelist attributes
         DESCR( ' ' ) +
         NAMES( )
*
******
DEFINE CHANNEL( 'SYSTEM.DEF.SENDER' ) +
         CHLTYPE( SDR ) +

* Sender channel attributes
         DESCR( ' ' ) +
         TRPTYPE( LU62 ) +
         XMITQ( ' ' ) +
         CONNAME( ' ' ) +
         MCAUSER( ' ' ) +
         BATCHSZ( 50 ) +
         DISCINT( 6000 ) +
         SHORTRTY( 10 )       SHORTTMR( 60 ) +
         LONGRTY( 999999999 ) LONGTMR( 1200 ) +
         SCYEXIT( ' ' )       SCYDATA( ' ' ) +
         MSGEXIT( ' ' )       MSGDATA( ' ' ) +
         SENDEXIT( ' ' )      SENDDATA( ' ' ) +
         RCVEXIT( ' ' )       RCVDATA( ' ' ) +
         SEQWRAP( 999999999 ) +
         MAXMSGL( 4194304 )
*
******
DEFINE CHANNEL( 'SYSTEM.DEF.SERVER' ) +
         CHLTYPE( SVR ) +

* Server channel attributes
         DESCR( ' ' ) +
         TRPTYPE( LU62 ) +
         XMITQ( ' ' ) +
         CONNAME( ' ' ) +
         MCAUSER( ' ' ) +
         BATCHSZ( 50 ) +
         DISCINT( 6000 ) +
         SHORTRTY( 10 )       SHORTTMR( 60 ) +
         LONGRTY( 999999999 ) LONGTMR( 1200 ) +
         SCYEXIT( ' ' )       SCYDATA( ' ' ) +
         MSGEXIT( ' ' )       MSGDATA( ' ' ) +
         SENDEXIT( ' ' )      SENDDATA( ' ' ) +
         RCVEXIT( ' ' )       RCVDATA( ' ' ) +
         SEQWRAP( 999999999 ) +
         MAXMSGL( 4194304 )
*
******
DEFINE CHANNEL( 'SYSTEM.DEF.RECEIVER' ) +
         CHLTYPE( RCVR ) +

* Receiver channel attributes
         DESCR( ' ' ) +
         TRPTYPE( LU62 ) +
         MCAUSER( ' ' ) +
         BATCHSZ( 50 ) +
         SCYEXIT( ' ' )       SCYDATA( ' ' ) +
         MSGEXIT( ' ' )       MSGDATA( ' ' ) +
         SENDEXIT( ' ' )      SENDDATA( ' ' ) +
         RCVEXIT( ' ' )       RCVDATA( ' ' ) +
         PUTAUT( DEF ) +
         SEQWRAP( 999999999 ) +
         MAXMSGL( 4194304 )
*
******
DEFINE CHANNEL( 'SYSTEM.DEF.REQUESTER' ) +
         CHLTYPE( RQSTR ) +

* Requester channel attributes
         DESCR( ' ' ) +
         TRPTYPE( LU62 ) +
         CONNAME( ' ' ) +
         MCAUSER( ' ' ) +
         BATCHSZ( 50 ) +
         SCYEXIT( ' ' )       SCYDATA( ' ' ) +
         MSGEXIT( ' ' )       MSGDATA( ' ' ) +
         SENDEXIT( ' ' )      SENDDATA( ' ' ) +
         RCVEXIT( ' ' )       RCVDATA( ' ' ) +
         PUTAUT( DEF ) +
         SEQWRAP( 999999999 ) +
         MAXMSGL( 4194304 )
*
DEFINE CHANNEL( 'SYSTEM.DEF.SVRCONN' ) +
         CHLTYPE( SVRCONN ) +

* Server connection channel attributes
         DESCR( ' ' ) +
         TRPTYPE( LU62 ) +
         MCAUSER( ' ' ) +
         SCYEXIT( ' ' )       SCYDATA( ' ' ) +
         MSGEXIT( ' ' )       MSGDATA( ' ' ) +
         SENDEXIT( ' ' )      SENDDATA( ' ' ) +
         RCVEXIT( ' ' )       RCVDATA( ' ' ) +
         MAXMSGL( 4194304 )
*
******
DEFINE CHANNEL( 'SYSTEM.DEF.CLNTCONN' ) +
         CHLTYPE( CLNTCONN ) +

* Client connection channel attributes
         DESCR( ' ' ) +
         TRPTYPE( LU62 ) +
         MODENAME( ' ' )      TPNAME( ' ' ) +
         CONNAME( ' ' ) +
         SCYEXIT( ' ' )       SCYDATA( ' ' ) +
         MSGEXIT( ' ' )       MSGDATA( ' ' ) +
         SENDEXIT( ' ' )      SENDDATA( ' ' ) +
         RCVEXIT( ' ' )       RCVDATA( ' ' ) +
         USERID( ' ' )        PASSWORD( ' ' ) +
         QMNAME( ' ' ) +
         MAXMSGL( 4194304 )
*
```

```
*********************************************************************
* SYSTEM.COMMAND DEFINITIONS
*********************************************************************
*
* SYSTEM.COMMAND objects are used by the queue manager to define
* the input queue and reply-to queue for system commands issued
* using the command server. These objects must be defined before
* operations and control panels can be used to issue commands to
* a queue manager.
*
*  SYSTEM.COMMAND.INPUT
*  SYSTEM.COMMAND.REPLY.MODEL
*
* The object names and queue types should not be changed. All the
* essential attributes are specified so that the definitions are
* not dependent on the default definitions.
*
* For normal operation, the following SYSTEM.COMMAND.INPUT
* attributes should not be changed:
*
*  PUT( ENABLED )
*  GET( ENABLED )
*  MAXMSGL( 32762 )
*  USAGE( NORMAL )
*  DEFSOPT( EXCL )
*  NOTRIGGER
*
* Changes to the other SYSTEM.COMMAND.INPUT queue attributes could
* be used to control the operation of the command server.
*
* Any of the SYSTEM.COMMAND.REPLY.MODEL attributes may be changed,
* but for normal usage, the following should be set:
*
*  PUT( ENABLED )
*  GET( ENABLED )
*  USAGE( NORMAL )
*  NOTRIGGER
*  MAXMSGL( at least 13000 )
*
******
DEFINE QLOCAL( 'SYSTEM.COMMAND.INPUT' ) +

* Common queue attributes
         DESCR( 'System-command input queue' ) +
         PUT( ENABLED ) +
         DEFPRTY( 5 ) +
         DEFPSIST( NO ) +

* Local queue attributes
         GET( ENABLED ) +
         NOSHARE +
         DEFSOPT( EXCL ) +
         MSGDLVSQ( PRIORITY ) +
         RETINTVL( 999999999 ) +
         MAXDEPTH( 999999999 ) +
         MAXMSGL( 32762 ) +
         NOHARDENBO +
         BOTHRESH( 0 ) +
         BOQNAME( ' ' ) +
         STGCLASS( 'SYSTEM' ) +
         USAGE( NORMAL ) +

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

* Trigger attributes
         NOTRIGGER +
         TRIGTYPE( NONE ) +
         TRIGMPRI( 0 ) +
         TRIGDPTH( 1 ) +
         TRIGDATA( ' ' ) +
         PROCESS( ' ' ) +
         INITQ( ' ' )
*
******
DEFINE QMODEL( 'SYSTEM.COMMAND.REPLY.MODEL' ) +

* Common queue attributes
         DESCR( 'System-command reply-to queue' ) +
         PUT( ENABLED ) +
         DEFPRTY( 0 ) +
         DEFPSIST( NO ) +

* Model queue attributes
         DEFTYPE( TEMPDYN ) +

* Local queue attributes
         GET( ENABLED ) +
         SHARE +
         DEFSOPT( SHARED ) +
         MSGDLVSQ( FIFO ) +
         RETINTVL( 999999999 ) +
         MAXDEPTH( 999999999 ) +
         MAXMSGL( 4194304 ) +
         NOHARDENBO +
         BOTHRESH( 0 ) +
         BOQNAME( ' ' ) +
         STGCLASS( 'SYSTEM' ) +
         USAGE( NORMAL ) +

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

* Trigger attributes
         NOTRIGGER +
         TRIGTYPE( NONE ) +
         TRIGMPRI( 0 ) +
         TRIGDPTH( 1 ) +
         TRIGDATA( ' ' ) +
         PROCESS( ' ' ) +
         INITQ( ' ' )
*
*
*********************************************************************
* SYSTEM.ADMIN DEFINITIONS
*********************************************************************
*
* The SYSTEM.ADMIN objects are used by queue managers to define
* queues for event reporting.
*
*  SYSTEM.ADMIN.QMGR.EVENT      queue manager-related events
*  SYSTEM.ADMIN.PERFM.EVENT     performance-related events
*  SYSTEM.ADMIN.CHANNEL.EVENT   channel-related events
*
* The reporting of queue manager and performance-related events is
* controlled by various queue and queue manager attributes.
*
* There are no controls for channel related events - only define
* the SYSTEM.ADMIN.CHANNEL.EVENT queue if you require the reporting
* of channel-related events, which are reported only when using
* distributed queuing without CICS.
*
* The object names should not be changed, though they can be
* defined as remote queues instead of local queues.
*
* If defined as local queues, the following attributes
* should not be changed:
*
```

```
*  PUT( ENABLED )
*  GET( ENABLED )
*  MAXMSGL( 4194304 )
*  USAGE( NORMAL )
*  DEFSOPT( EXCL )
*  QDPMAXEV( DISABLED )
*  QDPHIEV( DISABLED )
*  QDPLOEV( DISABLED )
*  QSVCIEV( NONE )
*
******
DEFINE QLOCAL( 'SYSTEM.ADMIN.QMGR.EVENT' ) +

* Common queue attributes
         DESCR( 'System queue manager-related event queue' ) +
         PUT( ENABLED ) +
         DEFPRTY( 0 ) +
         DEFPSIST( YES ) +

* Local queue attributes
         GET( ENABLED ) +
         NOSHARE +
         DEFSOPT( EXCL ) +
         MSGDLVSQ( PRIORITY ) +
         RETINTVL( 999999999 ) +
         MAXDEPTH( 999999999 ) +
         MAXMSGL( 4194304 ) +
         NOHARDENBO +
         BOTHRESH( 0 ) +
         BOQNAME( ' ' ) +
         STGCLASS( 'DEFAULT' ) +
         USAGE( NORMAL ) +

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

The remaining code for *MQSDEFA* is published in next month's issue.

Paul Jansen
Systems Programmer
Interpay (The Netherlands)

MQ news

Computer Associates has announced an MQSeries monitoring and management agent for its forthcoming Unicenter TNG for OS/390. Other agents to be available at around the same time include ones for the OS/390 base system, CICS, OS/390 Unix Services, CA-IDMS, CA-Datacom, DB2, Lotus Notes/Domino, SAP R3, PeopleSoft, Baan, Oracle, and Novell NetWare for System/390.

The company also announced Release 3.0 of its Unicenter TNG Option for MQSeries, an operational and configuration management tool for MQSeries-based networks and applications. Message management features include the ability to add, modify, and delete messages, and other features include ones to monitor all MQSeries objects and report performance statistics, queue and channel generation automation using configurable templates, and generation of MQSeries definitions.

Pricing and availability weren't announced for either product.

For further information contact:
Computer Associates International, 1 Computer Associates Plaza, Islandia, NY 11788, USA
Tel: +1 516 342 5224
Fax: +1 516 342 5734
Web: http://www.cai.com

Computer Associates Plc, Computer Associates House, 183-187 Bath Road, Slough, Berks SL1 4AA, UK
Tel: +44 1753 577733
Fax: +1 1753 825464

New Era of Networks has released NEONtrack Version 3.1, which monitors and controls transactions and messages within MQSeries queues, working with NEON's Integration Server and IBM's MQSeries. New features include centralized message management, viewing of messages passing through hubs, and secure message repair and reprocessing by flagging failed messages and allowing users to edit and repair them.

Reporting and performance optimization comes via a record of historical data for auditing transactions, which reports statistics on volume, processing speed, and error trends using Seagate Crystal Reports. Another new feature is real-time message tracking, which allows users to see the state and status of every message in the enterprise. This provides information on the progress of messages.

It's out now for NT 4.0, and supporting NEON Integration Servers running on MVS, AIX, NT, Solaris, and HP/UX servers. Prices start at US$60,000.

For further information contact:
New Era of Networks, 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