38

MQ update

August 2002

In this issue

MQ Update

Published by
Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 38126
From USA: 01144 1635 38126
Fax: 01635 38345
E-mail: info@xephon.com

North American office
Xephon/QNA
Post Office Box 350100
Westminster CO 80035-0100, USA
Telephone: (303) 410 9344
Fax: (303) 438 0290

Editor
Madeleine Hudson
E-mail: MadeleineH@xephon.com

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 ($33.75) each including postage.

Contributions
When Xephon is given copyright, articles published in *MQ Update* are paid for at the rate of £170 ($260) per 1000 words and £100 ($160) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £50 ($80) per 100 lines. In addition, there is a flat fee of £30 ($50) per article. For more information about contributing an article you can download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

MQ Update on-line
Code from *MQ Update*, and complete issues in Acrobat PDF format, can be downloaded from our Web site at www.xephon.com/mq; you will need to supply a word from the printed issue.

MQSI exception processing: request/reply messages – part one

Before we go into the detail of how to code a request/reply message subflow let us first determine how to handle messages. Based on the message type we can make assumptions as follows.

- *Datagram*. Since this is a 'fire and forget' message, when an exception is encountered the requester or message originator does not expect a reply so there is nowhere to send the exception information. We can let 'bad' messages roll back to the backout queue after the backout count exceeds the threshold. For ease of debugging we can also write an accompanying 'error encountered' message to an error queue with the correlation ID copied from the 'bad' message.

- *Request/reply message*. The requester provides the reply information in the MQMD and expects a reply. The requester usually does an MQGET on the reply queue with a wait interval. When an exception is encountered the request message or the reply message should be appended with error information and sent back to the requester, instead of letting the requester time-out on getting the reply.

  Since there may be an update performed in the message flow the message should be rolled back, and eventually it will be stored in the backout queue (when the backout count is greater than or equal to the backout threshold of the queue). We should also write an error message accompanying the backout message, just as we did for the datagram.

For a datagram the requisite actions are:

- Write the reason queue with the Correl-ID copied from the Msg-ID of the failing message.

- Throw the failing message so that it will end up in the backout queue.

- If the exception/error is systemic, stop the message flow so that the message will not be put erroneously to the backout queue.

For a request/reply message the requisite actions are:

- Populate the error information and send it back to the reply queue.
- Write to the reason queue with the Correl-ID copied from the Msg-ID of the failing message if the message is a request (if it is a reply the server program should have copied it).
- Throw the failing message so that it will end up in the backout queue (in case there is an update on the message flow).

The main task in exception handling for a request/reply message type is to come up with a message standard to convey the error information from one platform/tier to another. Since this type of message involves at least three platforms/tiers – the requester, the message hub, and the server – you need an agreed message format that they all understand. One option may be to have a separate error message; another would be to insert an error information section into the message.

Let’s assume that:

- All messages are in XML format.
- The first tag will be <Message>.
- The message body will be <MsgBody>.
- Error information is stored in the tag <Exception> and its occurrence can be none.
- We also use the same error message information we created for the error queue and pass this along to the <Message><Exception> of the request/ reply message.

Here is an example of an input message.

```
<Message>
     <MsgBody>
          </customerNumber>
          </productNumber>
          </quantity>
     </MsgBody>
</Message>
```

Here is an example of a message with error information inserted.

```
<Message>
     <MsgBody>
```

```
          </customerNumber>
          </productNumber>
          </quantity>
     </MsgBody>
     <Exception>
          </Type>
          </Number>
          </SuspenseTimestamp>
          </InboundQueue>
          </Transaction>
          </Reason>
     </Exception>
</Message>
```

REQUEST/REPLY MESSAGES WITH MQSI

Figure 1 illustrates a standard MQ request/reply scenario. The requester application running on the front end performs an MQPUT to put a request message to the REQUEST.OUT queue on queue manager FEQM. The REQUEST.OUT will be resolved to the queue REQUEST.IN in the back-end queue manager BEQM. The requesting application will then do an MQGET and wait for a reply on the REPLY.IN queue.

The server program running on the back end will retrieve the request message from queue manager BEQM and process it. In order for the reply to correspond to the request message the server program will copy the message ID from the request message and store it in the correlation ID of the reply message. It will also retrieve the ReplyToQ and ReplyToQMgr from the MQMD of the requesting message and put the message back to the reply queue as specified in these two fields. The

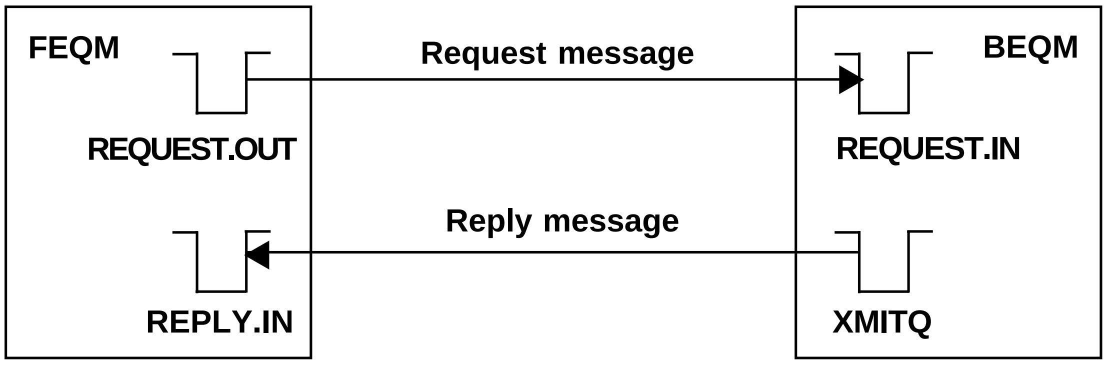


Figure 1: A standard MQ request/reply scenario

remote put will put the reply message via the transmit queue with the same name as the FEQM. This reply message will be shipped to the queue REPLY.IN on FEQM where the requester is waiting for it.

When data transformation is required between the front-end and back-end applications MQSI can be introduced between the requester and server applications, as Figure 2 illustrates.

In this case the requester will still put the request into queue REQUEST.OUT in the front-end queue manager FEQM. This time, though, the queue will be resolved to the queue HUB.REQUEST.IN on the MQSI queue manager. A message flow will pick up the request message, transform it, and pass it on to the server program. The MQSI will also set the ReplyToQ and ReplyToQMgr on the MQMD of the request message to queue HUB.REPLY.OUT on the MQSI message hub, where there will be a message flow waiting for the reply message, and transform it accordingly. The server program remains the same; it will process the request, copy the message ID into the correlation ID of the reply message, and reply to the queue as specified in the reply information of the MQMD from the incoming request message.

In the reply message flow of the hub the message flow has to have the intelligence to put the reply message back to the REPLY.IN queue on the FEQM.

Hard-coding the MQOutput node

As Figure 2 illustrates, at the end of the reply message flow we can hard-code the MQOutput queue name to REPLY.IN of queue manager

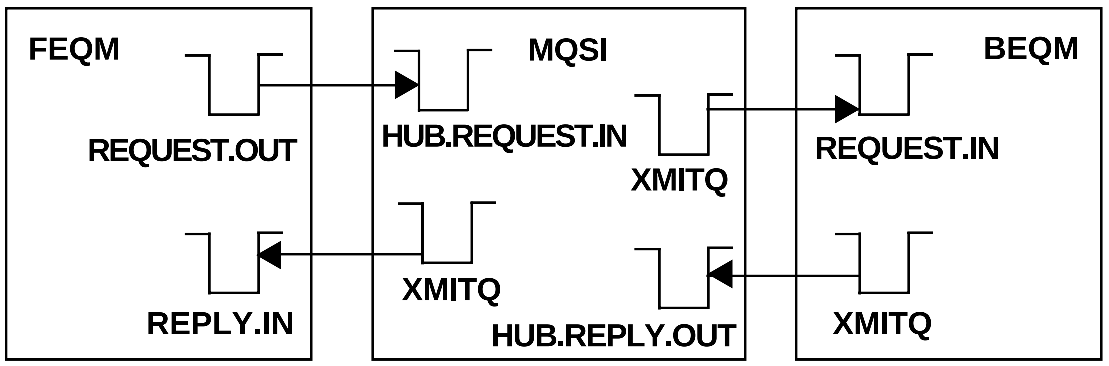


Figure 2: Data transformation with MQSI

FEQM. This will work nicely for most static request/reply message flows since we have knowledge of the whole flow, how it originates, and where it should end. But this hard-coding of the reply to the queue at the MQOutput node may result in a problem even though the message flow is static. Consider what may happen if another front-end queue manager is added due to an increase in workload, as Figure 3 illustrates.

Here the reply message has to be dynamically routed to the correct front-end queue manager where their reply queue names are the same. This dynamic nature implies that the original reply information from the request message has to be saved during the request flow, and then retrieved during the reply flow so that the reply will go back to the correct destination where the requester is expecting it. There are a number of ways to achieve this.

- *Database approach*. The reply information of the requesting message is stored in a database table with the message ID as key during the request flow. This reply information will be retrieved from the table on the reply flow by searching the table for a matching row ID with the Correl-ID of the incoming reply message

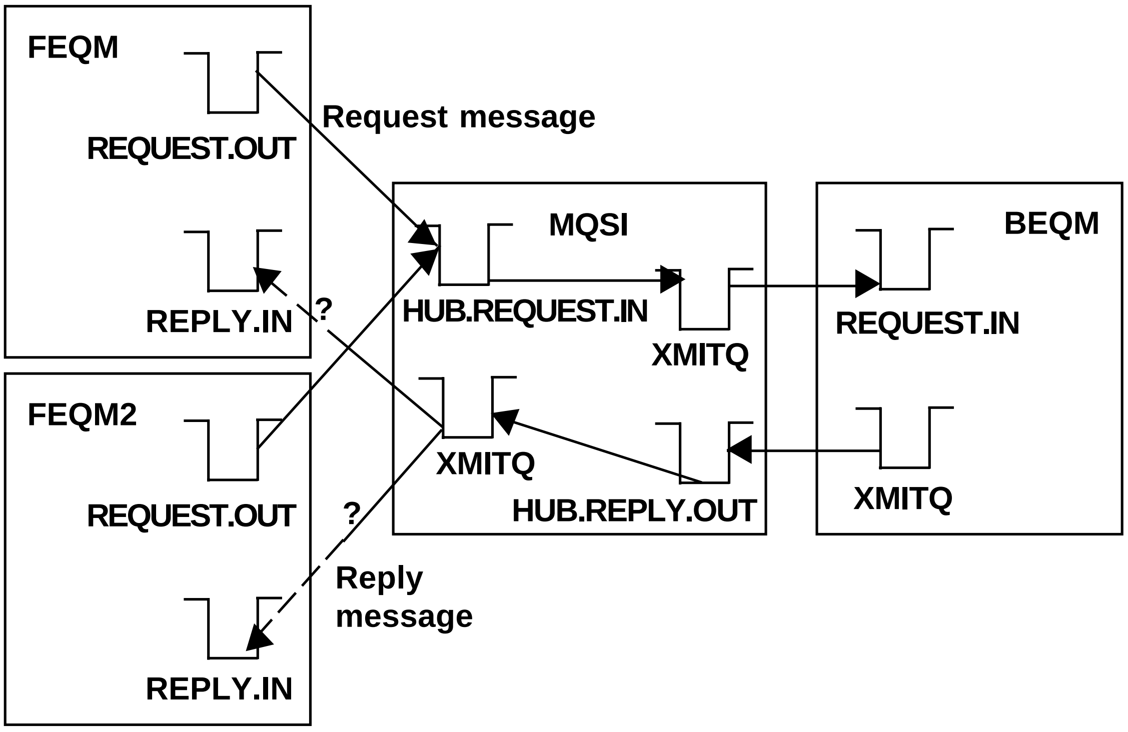


Figure 3: Adding another front-end queue manager

on the stored table. Once found, the reply information is retrieved and populated back to the MQMD fields. The row is then deleted.

This approach is quite expensive, with overheads that involve I/O reads and writes to and from a database table and a clean-up task for the rows stored that never receive a reply. It also complicates any recovery scenario since we have one more table to consider.

- *IBM SupportPacs*. There are two MQSI SupportPacs on the IBM Web site that allow message flows to save information and retrieve it later. SupportPac IA09 uses a queue to hold the saved reply information and a custom node MQGet to retrieve this saved reply information.

  SupportPac IA0H allows message flows to store the information in virtual memory using the custom node PostitCreate and retrieve information from memory with custom node PostitApply.

  Both SupportPacs are very easy to install and much 'cheaper' (in overhead terms) than the database approach.

The following section will give a brief explanation on implementing request/reply with these SupportPacs.

MQGET

Figure 4 illustrates usage of the SupportPac. In the request flow, the flow order node 'Save Reply Info first' makes sure that the reply information is saved first before processing the transformation of the message. The ESQL of the Save Reply Information compute node is given here.

```
SET OutputRoot.Properties.MessageDomain ='MRM';
SET OutputRoot.Properties.MessageSet ='DN8989C072001';
SET OutputRoot.Properties.MessageType ='m_ReplyInfo';
SET OutputRoot.Properties.MessageFormat ='CWF';
-- copy CorrelId from MsgId to the save reply info message for retrieval
SET OutputRoot.MQMD.CorrelId = InputRoot.MQMD.MsgId;
-- save the reply information
SET OutputRoot.MRM.ReplyToQ = InputRoot.MQMD.ReplyToQ;
SET OutputRoot.MRM.ReplyToQMgr = InputRoot.MQMD.ReplyToQMgr;
```

An MRM message is created to store the reply information containing two fields of 48 bytes. The information is saved in Save Queue output node. Once the reply information is saved the Request Transform compute node can then change the reply information to the queue that is located in the hub,

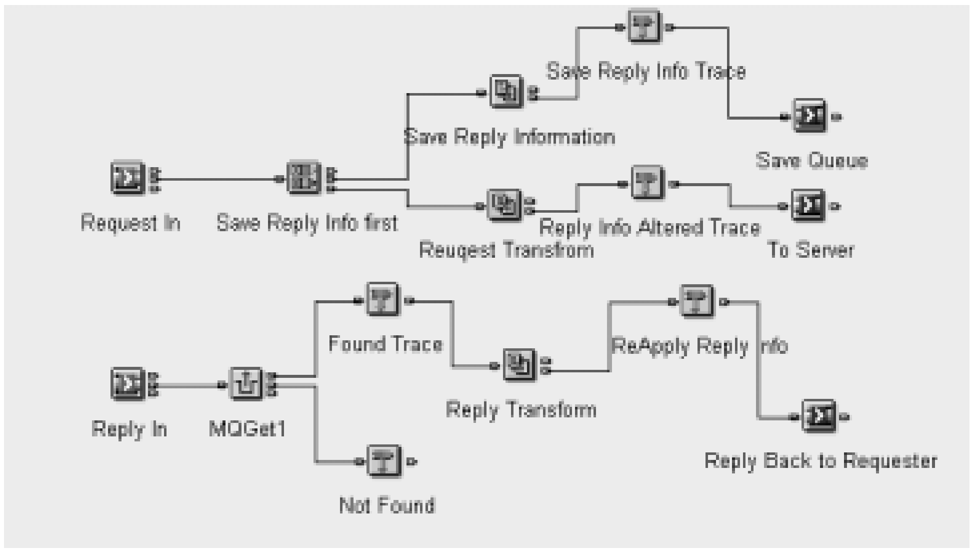


Figure 4: Sample message flow

where the reply message flow will process it.

```
SET OutputRoot.MQMD.ReplyToQ = 'ALEX_REPLY_IN;
SET OutputRoot.MQMD.ReplyToQMgr = ' ';
```

It then performs subsequent data transformation. Note that when the ReplyToQMgr is set to blank the Broker Queue Manager will populate its own name into this field as it puts the message out onto the queue in the MQOutput node.

On the reply message flow the first thing we need to do is restore the reply information: this is done by the MQGet node with the properties set as follows:

- Input queue name: ALEX_SAVE.
- Message tree location: *Root.queue.message*.
- Data type: string.
- Select by Correl-ID: yes.
- Transaction mode: automatic.

The key thing to note is that the message tree location is where we want the information retrieved from the queue to be put: in our case, it's *Root.queue.message*, a temporary scratch pad area within the message. The data type indicates that the retrieve information is a string and the matching criterion is 'Select by Correl-ID', where the Correl-ID of the input reply message is matched with the Correl-ID of the save message (note: not the Correl-ID of the input reply message matching the Msg-ID of the save message).

When successfully retrieved the reply information will be restored by the Reply Transform compute node.

```
SET OutputRoot.MQMD.ReplyToQ = SUBSTRING(InputRoot.queue.message FROM 1
FOR 48);
SET OutputRoot.MQMD.ReplyToQMgr = SUBSTRING(InputRoot.queue.message FROM
49 FOR 48);
```

Subsequent message transformation can continue after the above restore. The message flow trace, with the reply information in the MQMD shown in italics, can be found on the Web at www.xephon.com/extras/trace.txt.

POSTIT

A sample flow using the Postit node is shown in Figure 5. Usually the Postit node allows you to store one field of the message and retrieve it to any part of the message. If we are to save both the ReplyToQ and ReplyToQMgr we will require two PostitCreate Nodes to save and two PostitApply nodes to retrieve. The example below demonstrates a way that can be used for XML messages by grouping the information into one field like the MQGet flow example, using the scratch pad area concept.

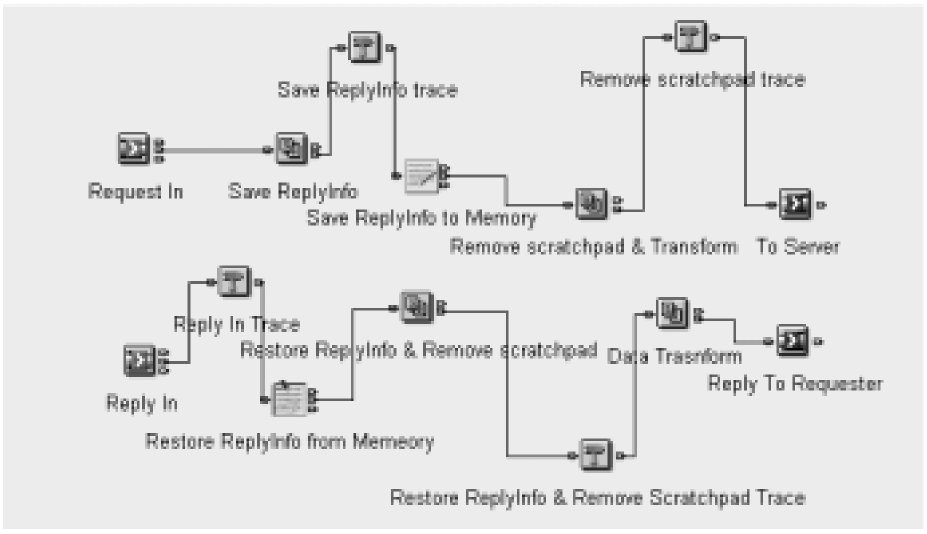


Figure 5: A sample flow using the Postit node

When the request arrives the message reply information will be saved. The ESQL for the Save ReplyInfo compute node is shown here.

```
DECLARE RTOQ CHAR;
DECLARE RTOQMGR CHAR;
-- blank out 48 spaces
SET RTOQ = '                                ';
SET RTOQMGR = '                               ';
-- overlay them with the reply information
SET OutputRoot.XML.scratchpad= OVERLAY(RTOQ PLACING
InputRoot.MQMD.ReplyToQ FROM 1) || OVERLAY(RTOQMGR PLACING
InputRoot.MQMD.ReplyToQMgr FROM 1);
```

The PostitCreate node Save ReplyInfo to Memory has the following properties:

- PostitName: SAVEREPLYINFO.
- CopyDataFrom: *Root.XML.scratchpad*.
- TimeoutSeconds: 0.
- NumberofApplications: 1.

After the reply information is saved the compute node ‘Remove scratchpad and Transform’ can do the normal data transformation, set the reply information for the reply message to come back to the hub, and remove the scratch pad area.

```
-- clean up the scratchpad area
SET OutputRoot.XML.scratchpad = null;
-- alter the reply Information to point to Hub
SET OutputRoot.MQMD.ReplyToQ = 'ALEX_REPLY_IN';
SET OutputRoot.MQMD.ReplyToQMgr = ' ';
-- for testing of Postit
SET OutputRoot.MQMD.CorrelId = InputRoot.MQMD.MsgId;
```

On the reply trip the first action is to retrieve the reply information from the virtual memory by the PostitApply node Restore ReplyInfo from Memory, using the following properties:

- PostitName: SAVEREPLYINFO.
- CopyDataTo: *Root.SML.scratchpad*.
- MatchOn: Correlid=OldMsgid.

Note that the PostitName should be the same as that of the PostitCreate Node. The information stored will be put into the message tree *Root.XML.scratchpad*.

The compute node Restore ReplyInfo and Remove scratchpad will be responsible for moving the reply information. The ESQL is shown here.

```
SET OutputRoot.MQMD.ReplyToQ = SUBSTRING(InputRoot.XML.scratchpad FROM
1 FOR 48);
SET OutputRoot.MQMD.ReplyToQMgr = SUBSTRING(InputRoot.XML.scratchpad
FROM 49 FOR 48);
-- clean up the scratch pad area
SET OutputRoot.XML.scratchpad = null;
```

The trace file record, showing how the reply information is stored and restored in this message flow, can be found on the Web at www.xephon.com/extras/reply.txt.

Editor's note: this article concludes next month.

Alex Au
IT Architect, IBM Global Services (USA)

Performance Event Queue Viewer: MQ for OS/390

To install the Performance Event Queue Viewer you need to send the REXX code to your mainframe and store it as a library member (RECFM=FB, LRECL=80) named MQEVNPER. After customization you can run the Performance Event Queue Viewer against your SYSTEM.ADMIN.PERFM.EVENT queue. The job has been tested with OS/390 V2.8 and MQSeries for OS/390 V2.1.

MQSERIES PERFORMANCE EVENTS

MQSeries has the ability to gather events that relate to its work. There are three types of event – called instrumentation events – and each category has its own event queue.

- Queue manager events (SYSTEM.ADMIN.QMGR.EVENT).
- Channel events (SYSTEM.ADMIN.CHANNEL.EVENT).
- Performance events (SYSTEM.ADMIN.PERFM.EVENT).

Performance events are basically notifications that a threshold condition has been reached by a queue, ie a queue is full. There are two types of performance event:

- Queue depth events: these are related to the number of messages on a queue; ie how full or empty the queue is.
- Queue service interval events: these concern whether or not messages are processed within a user-specified time interval.

The specific performance events are:

- *Queue Depth High*. An MQPUT or MQPUT1 call has caused the queue depth to be incremented to or above the limit specified in the QdepthHi attribute.
- *Queue Depth Low*. An MQGET call has caused the queue depth to be decremented to or below the limit specified in the QDepthLo attribute.
- *Queue Full*. On an MQPUT or MQPUT1 call, the call failed because the queue is full. That is, it already contains the maximum number of messages possible. This is set by MaxQDepth attribute.
- *Queue Service Interval High*. No successful gets or puts have been detected within an interval greater than the limit specified in the QSvcInt (queue service interval) attribute.
- *Queue Service Interval OK*. A successful get has been detected within an interval less than or equal to the limit specified in the QSvcInt attribute.

Performance events must be enabled on the queue manager otherwise no performance events can occur. You can then enable the specific performance events by setting the appropriate queue attribute. You also have to specify the conditions that give rise to the event. Performance events are never generated for the event queues themselves. To enable performance events:

- Set the queue manager attribute PERFMEV to ENABLED.
- For queue depth events, set the appropriate queue attribute to ENABLED:
  - QDPHIEV for Queue Depth High events
  - QDPLOEV for Queue Depth Low events
  - QDPMAXEV for Queue Full events.

- For queue service interval events, set the queue attribute QSVCIEV to HIGH or OK.
- Set the threshold queue attributes to appropriate values:
  - QSVCINT (time in milliseconds) for service interval events.
  - QDEPTHLO and QDEPTHHI (percentage of maximum queue depth) for queue depth events.

Performance event statistics are reset when:

- A performance event occurs (statistics are reset on all active queue managers).
- A queue manager stops and restarts.

Messages that are put to event queues have a special format, namely an additional header that is appended in the front of the application data. The format of a message sent to an event queue is set to MQFMT_EVENT.

RUNNING THE PERFORMANCE EVENT QUEUE VIEWER

The Performance Event Queue Viewer can be invoked either from the supplied job or run from a TSO or ISPF line command. These last two options are not recommended since, depending on the number of messages on the performance event queue, the Performance Event Queue Viewer can produce a lot of output. The job to run Performance Event Queue Viewer is shown below.

```
//MQENVPER JOB NOTIFY=&SYSUID
//*
//* parameters in SYSTSIN:
//*  QMgrName
//*  PerformanceEventQueueName
//*
//GETENV    EXEC PGM=IKJEFTØ1,
//          PARM='%MQENVPER'
//STEPLIB   DD DSN=MA18.LOAD,DISP=SHR
//          DD DSN=MQM.SCSQAUTH,DISP=SHR
//SYSEXEC   DD DSN=YOUR.DSN,DISP=SHR
//SYSTSPRT  DD SYSOUT=*
//SYSTSIN   DD *
QmgrName
PerformanceEventQueueName
/*
```

Tailoring the above code

- Line one: supply a valid job card.
- Line nine: MA18 load library.
- Line ten: MQSeries hlq.SCSQAUTH library.
- Line 11: Library containing the MQENVPER source.
- Line 12: a class for sysout (the Performance Event Queue Viewer reports will be printed there).
- Parameters for the Performance Event Queue Viewer:
  - line 14: Queue Manager Name (required)
  - line 15: Performance Event Queue Name (optional).

Parameters for the Performance Event Queue Viewer must come in the order shown, the last one may be omitted (leave a blank line), but the queue manager name is required – it has no default. The default values for the Performance Event Queue Viewer parameters are:

- QMgr: none
- PerformanceEventQueueName: none, if blank it defaults to SYSTEM.ADMIN.PERFM.EVENT.

OUTPUT

The Performance Event Queue Viewer will produce a report for every message it finds on the performance event queue. The following information will be given:

- The message descriptor MQMD. I will not describe here the full message descriptor (please refer to *MQSeries Application Programming Reference*), but several fields need a comment since they are subject to change when the message is put to the dead letter queue. They are:
  - format: always set to MQFMT_EVENT
  - type: always set to MQMT_DATAGRAM
  - PutTime and PutDate: time and date when the message was put to the performance event queue.

- The contents of the event header. There are fields listed below:
  - type: always set to EVENT
  - reason: reason for the event; it is one of the following:
    - MQRC_Q_DEPTH_HIGH
    - MQRC_QDEPTH_LOW
    - MQRC_Q_FULL
    - MQRC_Q_SERVICE_INTERVAL_HIGH
    - MQRC_Q_SERVICE_INTERVAL_LOW
  - Queue Name: name of the queue for which an event has been generated
  - Queue Manager Name: name of the queue manager
  - High Queue Depth: maximum number of messages on the queue since the queue statistics were last reset
  - Msg Enqueue Count: the number of messages put on the queue since the queue statistics were last reset
  - Msg Dequeue Count: the number of messages removed from the queue since the queue statistics were last reset
  - Time Since Reset: the time in seconds since the statistics were last reset.

After reading all messages from the performance event queue the Performance Event Queue Viewer will print their number and quit.

PROGRAMMING REMARKS

I have written the Performance Event Queue Viewer in REXX. Since the standard MQSeries API is not available for REXX I have used SupportPac MA18.

```
/* REXX                                            by Marcin Grabinski */

MaxMsgLen = 192                      /* Performance event message length*/

PARSE EXTERNAL QMgr
PARSE EXTERNAL EventQ

IF EventQ = '' THEN
```

```
  EventQ = 'SYSTEM.ADMIN.PERFM.EVENT'
SAY
SAY 'Performance Event Queue Viewer written by Marcin Grabinski'
SAY
/* Initialize the interface */
RXMQVTRACE = ''
rcc= RXMQV('INIT')
SAY rcc
/* Connect to Queue Manager */
RXMQVTRACE = ''
rcc = RXMQV('CONN', QMgr)
SAY rcc
IF WORD(rcc, 5) <> 'FAILED' THEN /* Connect OK */
DO
  /* Open Queue for Input */
  RXMQVTRACE = ''
  oo = MQOO_INPUT_AS_Q_DEF + MQOO_INQUIRE
  rcc = RXMQV('OPEN', EventQ, oo , 'h2', 'ood.' )
  SAY  rcc
  IF WORD(rcc, 5) <> 'FAILED' THEN /* Open OK */
  DO
    /* Get Current Queue Depth */
    RXMQVTRACE = ''
    rcc = RXMQV('INQ', h2, MQIA_CURRENT_Q_DEPTH, 'depth')
    SAY  rcc
    /* Read messages                  */
    RXMQVTRACE = ''
    DO i = 1 TO depth
       g.Ø      = MaxMsgLen
       g.1      = ''
       igmo.opt = MQGMO_ACCEPT_TRUNCATED_MSG
       rcc = RXMQV('GET', h2,'g.','igmd.','ogmd.','igmo.','ogmo.')
       SAY  rcc
       IF ( WORD(rcc,1) = 2Ø33 ) THEN LEAVE
       SAY
       SAY 'Message #'i':'
       SAY
       SAY '  The event header is 'g.Ø' bytes long'
       SAY '  Message Descriptor: '
       SAY '     MsgId:            'ogmd.msgid
       SAY '     CorelId:          'ogmd.cid
       SAY '     Report:           'ogmd.rep
       SAY '     MsgType:          'ogmd.msg
       SAY '     Expiry:           'ogmd.exp
       SAY '     Feedback:         'ogmd.fbk
       SAY '     Encoding:         'ogmd.enc
       SAY '     CodedCharSetId:   'ogmd.ccsi
       SAY '     Format:           'ogmd.form
       SAY '     Priority:         'ogmd.pri
       SAY '     Persistence:      'ogmd.per
       SAY '     BackoutCount:     'ogmd.bc
       SAY '     ReplyToQ:          'ogmd.rtoq
       SAY '     ReplyToQMgr:       'ogmd.rtoqm
       SAY '     UserId:            'ogmd.uid
       SAY '     AccountingToken:   'ogmd.at
       SAY '     ApplIdentityData: 'ogmd.aid
       SAY '     PutApplType:       'ogmd.pat
       SAY '     PutApplName:       'ogmd.pan
       SAY '     PutDate:           'ogmd.pd
       SAY '     PutTime:           'ogmd.pt
       SAY '     ApplOriginData:    'ogmd.aod
       SAY
       /* Extract the event header  */
       rcc = RXMQV('EVENT', 'g.', 'x.')
       SAY  rcc
       IF WORD(rcc, 1) = Ø THEN /* Event header ok */
       DO
         INTERPRET 'rtext = RXMQV.RCMAP.'x.REA
         SAY '  Event header:'
         SAY '     Type:                'x.TYPE
         SAY '     Reason:              'x.REA' ('rtext')'
         SAY '     Queue Name:          'x.BQN
         SAY '     Queue Manager Name: 'x.QM
         SAY '     High Queue Depth:    'x.HQD
         SAY '     Msg Enqueue Count:  'x.MEC
         SAY '     Msg Dequeue Count:  'x.MDC
         SAY '     Time Since Reset:    'x.TSR
         SAY
      END /* Event header OK */
    END /* DO i=1 TO depth*/
    SAY 'There were 'depth' messages on the event queue 'EventQ
    SAY
    /* Stop access to a Queue */
    RXMQVTRACE = ''
    rcc = RXMQV('CLOSE', h2, mqco_none)
    SAY  rcc
  END /* Open OK */
  /* Disconnect from the QM */
  RXMQVTRACE = ''
  rcc = RXMQV('DISC', )
  SAY  rcc
END /* Connect OK */
/* Remove the Interface functions from the Rexx Workspace ... */
RXMQVTRACE = 'TERM'
rcc = RXMQV('TERM', )
SAY  rcc
RETURN
```

Marcin Grabinski
System Engineer, PUP SPIN (Poland) © PUP SPIN 2002

Configuring a Web Client on Windows NT using IIS PWS Web Server – part one: configuration

INTRODUCTION

This article is intended to assist developers who would like to set up Web Client once they have successfully configured MQSeries Workflow on Windows NT in stand-alone mode. IBM suggests that you use WebSphere and HTTP server, so why did I choose IIS and Tomcat?

The main reasons were constraints on budget and time. Tomcat and IIS PWS are freeware and I thought I would save time and money; but configuring these products proved to be more difficult than anticipated, requiring a considerable amount of effort. Ultimately, and with much assistance from the IBM Lab in Germany, I configured them successfully.

I hope this article will be of use to others in a similar situation. I must take this opportunity to thank IBM and APACHE for their extensive support and documentation.

CONFIGURATION

In order to use Web Client with MQ Workflow you need the following:

- MQ Workflow 3.3.
- Web browser with JavaScript enabled (Internet Explorer 5.1 or higher, for preference: WebCredit does not work on Netscape.)
- A servlet and JSP-enabled Web server, ie IBM's WebSphere.
- Application Server or IIS-PWS 4.0 with Tomcat 3.3 Redirection DLL.
- Java 2 JDK 1.2.*x* or 1.3.
- If running MQSeries 5.2 or higher, the MQSeries Java Support must be installed (SupportPac MA88).

Configuring Web Client on Windows is easy if you are using IBM's WebSphere. Since we are going to use IIS PWS 4.0 Webserver, take the following steps.

- Install and configure MQ Workflow.
- Install JDK 1.3.1.
- Install and configure IIS-PWS , Tomcat 3.3, and Redirection DLL.
- Configure Web Client.
- Test with the WebCredit example.

Installing and configuring MQ Workflow

To install and configure MQ Workflow follow the steps described in the manual *Stand-alone client/server setup on Windows NT*.

Since we are using MQSeries 5.2 we have to install MQSeries classes for Java and MQSeries classes for Java Message Service for Windows NT. Download and install the software (*ma88_win.zip*) from *http://www-4.ibm.com/software/ts/mqseries/txppacs/ma88.html*. If you have unzipped the file in a separate folder, ie TMP, you will need to copy the contents of the folder MQSeries located under *C:\TMP\program files\IBM* to *<MQSeries Instl Dir>*.

Note that *<MQSeries Instl Dir>* will be *C:\Program Files\MQSeries* if you have followed the default installation settings.

Installing JDK 1.3.1

- Download Java 2 SDK V1.3.1 for Windows from *http://java.sun.com/j2se/1.3*.
- Add *C:\jdk1.3.1\bin* to the CLASS PATH variable.
- Add *C:\jdk1.3.1\bin* to the PATH variable.
- Create the user variable JAVA_HOME with the value *C:\jdk1.3.1*.

(See Appendix A: *How to add/create environment variables in Windows NT*.)

Installing and configuring IIS-PWS, Tomcat 3.2.3, and Redirection DLL

- Download and install the Windows NT 4.0 Option Pack for Windows NT, which contains IIS-PWS (Personal Web Server) from *http://*

www.microsoft.com/msdownload/ntoptionpack/askwiz.asp.

- IIS does not provide JSP or servlet support so we need to download the Tomcat servlet container, which understands the JSP and Servlet requests, and Redirection DLL to direct such requests from IIS-PWS to TOMCAT.
- Installing Tomcat:
  - download Tomcat *jakarta-tomcat 3.2.3.zip* from *http://jakarta.apache.org/builds/jakarta-tomcat/release/v3.2.3/bin/*
  - for detailed information about installing and running Tomcat point your browser at the file *doc/uguide/tomcat_ug.html* under the directory into which you unpacked the Tomcat distribution (eg *C:\jakarta-tomcat-3.2.3*).
- Check that Tomcat is installed properly by trying a couple of examples in the JSP tutorial at *http://java.sun.com/products/jsp/docs.html*.

MAKING TOMCAT WORK WITH IIS-PWS

Normally IIS cannot execute servlets and Java Server Pages (JSPs). Configuring IIS to use the Tomcat Redirector plugin *isapi_redirect.dll* will enable IIS to send servlet and JSP requests to Tomcat (and this way, serve them to clients).

Conventions and assumptions

<tomcat_home> is the root directory of Tomcat (eg *C:\jakarta-tomcat-3.2.3*). Your Tomcat installation should have the following subdirectories:

- *<tomcat_home>\conf* – where you can place various configuration files.
- *<tomcat_home>\webapps* – containing example applications.
- *<tomcat_home>\bin* – where you place Web server plugins.

In all the examples in this article *<tomcat_home>* will be *C:\jakarta-tomcat-3.2.3*.

A worker is defined to be a Tomcat process that accepts work from the IIS server.

Installation

As of Tomcat 3.2, a pre-built version of the ISAPI Redirector server plugin *isapi_redirect.dll* is available under the *win32/i386* directory from where you downloaded Tomcat 3.2.3 (*http://jakarta.apache.org/builds/jakarta-tomcat/release/v3.2.3/bin/*).

For those using a Netscape browser, try downloading a zipped version of the file if available, as there can be problems using Netscape to download DLL files. You can also build a copy locally from the source in Tomcat's source distribution.

The Tomcat Redirector requires three entities:

- *isapi_redirect.dll* – the IIS server plugin; either obtain a pre-built DLL or build it yourself.
- *workers.properties* – a file that describes the host(s) and port(s) used by the workers (Tomcat processes). A sample *workers.properties* can be found under the conf directory.
- *uriworkermap.properties* – a file that maps URL-path patterns to workers. A sample *uriworkermap.properties* can also be found under the conf directory.

The installation includes the following parts:

- Configuring the ISAPI Redirector with a default /examples context and checking that you can serve servlets with IIS.
- Adding more contexts to the configuration.

Configuring the ISAPI Redirector

In this article it is assumed that *isapi_redirect.dll* is placed in *C:\jakarta-tomcat-3.2.3\bin\win32\i386\isapi_redirect.dll* and that the properties files you created (*workers.properties, uriworkermap.properties*) are in *C:\jakarta-tomcat-3.2.3\conf.*

- In the registry, create a new registry key named *HKEY_LOCAL_MACHINE\SOFTWARE\Apache Software foundation\Jakarta Isapi Redirector\1.0*.
- Add a string value with the name *extension_uri* and a value of */jakarta/isapi_redirect.dll*.

- Add a string value with the name *log_file* and a value pointing to where you want your log file to be (eg *C:\jakarta-tomcat-3.2.3\logs\isapi.log*).
- Add a string value with the name *log_level* and a value for your log level (debug, info, error, or emerg).
- Add a string value with the name *worker_file* and a value which is the full path to your *workers.properties* file (eg *c:\jakarta-tomcat-3.2.3\conf\workers.properties*).
- Add a string value with the name *worker_mount_file* and a value which is the full path to your *uriworkermap.properties* file (eg *c:\jakarta-tomcat-3.2.3\conf\uriworkermap.properties*).
- Using the IIS management console, add a new virtual directory to your IIS/PWS Web site. The name of the virtual directory must be *jakarta*. Its physical path should be the directory where you placed *isapi_redirect.dll* (in our example it is *c:\jakarta-tomcat-3.2.3bin\win32\i386*). While creating this new virtual directory, assign it execute access. (See Appendix A: *How to add virtual directory in IIS/PWS*.)
- Using the IIS management console add *isapi_redirect.dll* as a filter in your IIS/PWS Web site. The name of the filter should reflect its task (I use the name *jakarta*); its executable must be our *c:\jakarta-tomcat-3.2.3\bin\win32\i386\isapi_redirect.dll*. For PWS you'll need to use regedit and add/edit the Filter DLLs key under *HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\W3SVC\Parameters*. This key contains a comma-separated list of dlls (full paths): you need to insert the full path to *isapi_redirect.dll*.
- Restart IIS (stop and start the IIS service) and make sure that the jakarta filter is marked with a green upwards-pointing arrow. Note that in Windows NT/2000 the stop/start feature of the Microsoft Management Console does not actually stop and start the IIS service. You need to use the services control panel (or the **net** command) to stop and start the World Wide Web Publishing Service.
- In the *uriworkermap.properties* file (not the *AUTO* file) add the following statements:

```
# Mount the servlet context to the ajp12 worker
/servlet/*=ajp12
# Mount the examples context to the ajp12 worker
/examples/*=ajp12
# Chandra's add for webclient configuration "FMC1"
/MQWFClient-FMC1/*=ajp12
```

- In the *workers.properties* file make the following corrections:
  - *# workers.tomcat_home* should point to the location where you installed Tomcat. This is where you have your conf, webapps, and lib directories.

    ```
    # workers.java_home=c:\jdk1.2.2 to
    workers.java_home=c:\jdk1.3.1
    ```

  - *# workers.java_home* should point to your Java installation. Normally you should have bin and lib directories beneath it.

    ```
    # workers.tomcat_home=c:\jakarta-tomcat to
    workers.tomcat_home=c:\jakarta-tomcat-3.2.3
    ```

  - You should configure your environment slash... ps=\ on NT, / on Unix, and maybe something different elsewhere.

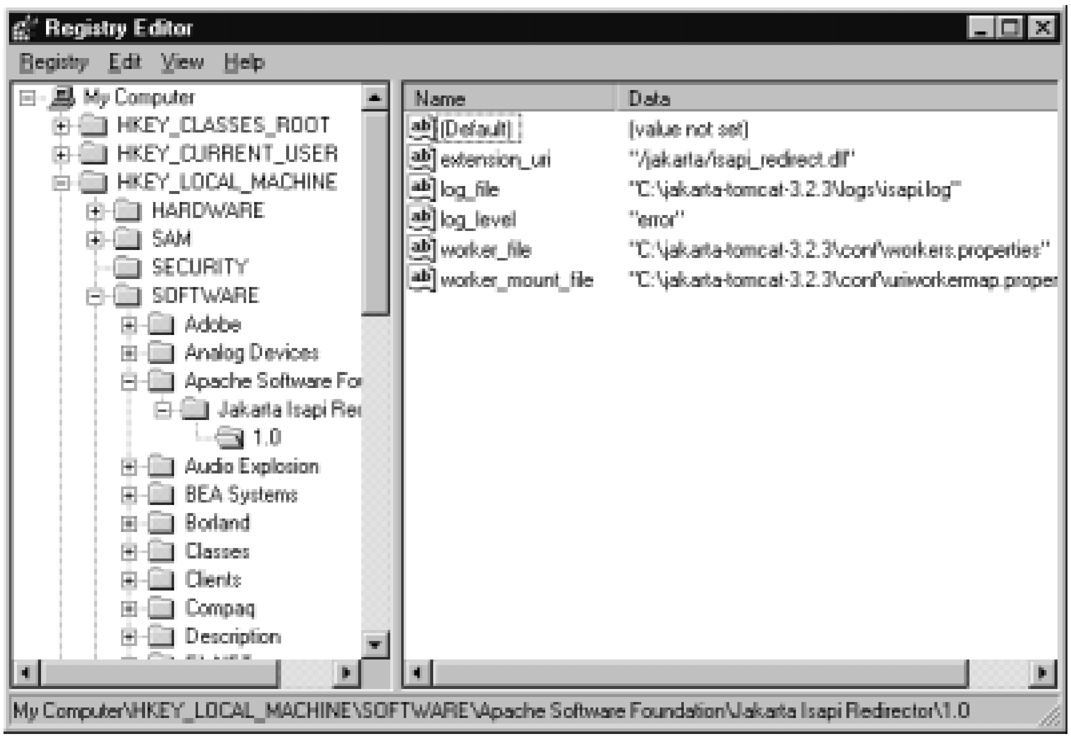
    ```
    #
    ps=\
    # ps=/
    ```

- You should now start Tomcat and ask IIS to serve you the /examples context. Try *http://localhost/examples/jsp/index.html* for example, and execute some of the JSP samples. If this does not work successfully refer to the *Troubleshooting* section for help.
- At the end of our modifications the Registry screen looks something like Figure 1.

Adding contexts

The examples context is useful for verifying your installation but you will also need to add your own contexts. Adding a new context requires two operations:

- Adding the context to Tomcat (not covered here).
- Adding the context to the ISAPI Redirector, which is simple. All you need to do is edit your *uriworkermap.properties* and add a line that looks like:



Name	Data
(Default)	(value not set)
extension_uri	"/jakarta/isapi_redirect.dll"
log_file	"C:\jakarta-tomcat-3.2.3\logs\isapi.log"
log_level	"error"
worker_file	"C:\jakarta-tomcat-3.2.3\conf\workers.properties"
worker_mount_file	"C:\jakarta-tomcat-3.2.3\conf\uriworkermap.proper

My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Apache Software Foundation\Jakarta Isapi Redirector\1.0

Figure 1: The Registry screen

```
/context/*=worker_name
```

Workers and their names are defined in *workers.properties*. By default *workers.properties* comes with a single preconfigured worker named ajp12. As an example, if you want to add a context named shop, the line that you should add to *uriworkermap.properties* will be:

```
/shop/*=ajp12
```

After saving *uriworkermap.properties* restart IIS and it will serve the new context. You must stop and start the IIS Admin Service from the service control panel; the MMC will not pick up the changes to the *uriworkermap.properties* file.

As a new feature in Tomcat 3.2 a *uriworkermap.properties-auto* is automatically written each time Tomcat is started. This file includes settings for each of the contexts that Tomcat will serve during its run. Each context has settings to have Tomcat handle servlet and JSP requests, but by default static content is left to be served by IIS. Each context also has a commented-out setting to have Tomcat handle all requests to the context. You can rename this file (so it won't be overwritten the next time Tomcat is started) and uncomment this setting or make other customizations. You can also use this file as is, in your *worker_mount_file* setting.

TROUBLESHOOTING

Sometimes the ISAPI Redirector will not work the first time you try to install it. If this happens, follow these steps; they're not guaranteed to cover all problems but they should help find the most common errors. If you make any corrections during these steps restart the IIS service as described above.

These guidelines assume your *worker_mount_file* setting points to an unmodified copy of the *uriworkermap.properties* file. Results may be misleading if *worker_mount_file* points to a modified *uriworkermap.properties* or the *uriworkermap.properties-auto* file. It is also assumed that the */examples* context works correctly if you access Tomcat directly.

WinNT

- Make sure Website activity is being logged. For PWS 4.0 make sure 'Save Website Activity Log' is checked in the Advanced Options of the Personal Web Manager.
- Start the World Wide Web Publishing Service and Tomcat.
- Check for the presence of the ISAPI Redirector log file you specified in the *log_file* setting. If it is not found, check the following:
  - check the executable you set for the filter in the IIS Management Console and make sure the path is correct.
  - check the spelling of the *HKEY_LOCAL_MACHINE\SOFTWARE\Apache Software Foundation\Jakarta Isapi Redirector\1.0* key. Case isn't important, but an incorrect letter will prevent the *isapi_redirect.dll* from finding its registry settings.
  - check the *log_file* setting for typos, name, and data. Also ensure that the directory in which the log file will appear already exists.

If the above are set correctly the ISAPI Redirector should be able to create the log file.

- Check the Jakarta filter you added and make sure its status shows a green upward-pointing arrow. If not:

  – check the *worker_file* setting for typos, name, and data.

  – check the *worker_mount_file* setting for typos, name, and data.

  If the above are set correctly the green up-arrow should appear even if the other settings are wrong.

- Invoke the URL *http://localhost/examples/jsp/index.html* in your browser. Case is important in Tomcat 3.2. The characters following 'localhost' in the URL must be lower case. If the page fails to appear examine the last line in the IIS server log file in found in *SYSTEM32/LogFiles/W3SVC1*:

  – the last line should contain something like: GET "/jakarta/isapi_redirect.dll HTTP1.1", which indicates the ISAPI redirector is recognizing that it should handle the request.

  – if the number following GET "/..." is 404 make sure you entered the URL correctly.

  – if the number following GET "/..." is 500 make sure the virtual directory created was called 'jakarta'.

  – make sure that the extension_uri setting is correct.

  – check the *workers.properties* file and make sure the port setting for *worker.ajp12.port* is the same as the port specified in the *server.xml* for the Apache AJP12 support.

  – if the number following GET "/..." is 200 or 403 make sure you have checked 'Execute Access' for the jakarta virtual directory in the 'Advanced Options' of the Personal Web Manager.

  If the above settings are correct the *index.html* page should appear in your browser. You should also be able to click the 'Execute links' to execute the JSP examples.

For detailed information about making Tomcat work with IIS-PWS point your browser at the file *doc/uguide/tomcat-iis-howto.html* under the directory into which you unpacked the Tomcat distribution (eg C:\jakarta-tomcat-3.2.3).

CONFIGURING WEB CLIENT

If you want to use the Web Client on a Windows platform you must configure it as you would on a Unix platform, using the command line **configuration utility fmczutil**.

- Log on as an administrator.
- Open a command prompt window.
- Enter the command **fmczutil at C:\**.
- You should see the Configuration Commands menu *FMC33201I Configuration Commands Menu*.
  - l ...List
  - s ...Select
  - c ...Create
  - d ...Change default configuration
  - x ...Exit Configuration Commands Menu
- Enter *c* to create a new configuration.
- Enter a new configuration identifier, eg FMC1.
- Enter *w* to select the Web Client.
- If you want to use 'bindings' as the locator policy enter *j* to select the Java agent.
- I selected the following categories while working on a new configuration, *ID FMC1*:
  - s: Server
  - i: Runtime database utilities
  - c: Client with queue manager
  - j: Java agent
  - w: Web Client

```
FMC33210I Select Category Menu:
     s ... (X) Server
     i ... (X) Runtime Database Utilities
     b ... (  ) Buildtime
     c ... (X) Client with queue manager
     j ... (X) Java Agent
     w ... (X) Web Client
     a ... all
     n ... none
     x ... Exit Select Category Menu
-  Configuration of Runtime database ...
     u ... ( ) Use an existing Runtime database
     n ... (X) Create a new Runtime database
     DB2 instance            : [DB2]
     DB2 database            : [FMCDB1]
     DB2 user ID of database administrator       : [ChandraU]
     DB2 database layout file : [c:\program files\mqseries
workflow\cfgs\fmc1\fmcdblay.ini]
     DB2 database location    : [C:]
     DB2 container location   : [C:\Program Files\MQSeries
Workflow\rt_db\DB2\FMCDB1]
     DB2 log files location   : [C:\Program Files\MQSeries
Workflow\rt_db\DB2\FMCDB1]
   FMC33526I Select space management ...:
     s ... (X) Managed by system
     d ... ( ) Managed by database
-  FMC33749I Selected Space management : Managed by system
     DB2 user ID to access Runtime database      : [ChandraU]
     System group name       : [FMCGRP1]
     System name             : [FMCSYS1]
     Queue manager name      : [FMCQM1]
     Queue prefix            : [FMC]
-  Configuration of queue manager ...
   FMC33513I Select log type ...:
     c ... (X) Circular log
     l ... ( ) Linear log (prerequisite for backup)
-  FMC33749I Selected Log type : Circular log
     Queue manager log files location     : []
     Channel definition table file        : [c:\program files\mqseries
workflow\chltabs\mqwfchl.tab]
   FMC33507I Select communication protocol ...:
     t ... (X) TCP/IP
     n ... ( ) NetBios
     a ... ( ) APPC
-  FMC33749I Selected Communication protocol : TCP/IP
     TCP/IP address          : [c030824]
     TCP/IP port number      : [14000]
     Principal name          : [fmc]
-  FMC33749I Selected Communication protocol : TCP/IP
     TCP/IP address          : [c030824]
     TCP/IP port number      : [14000]
     Principal name          : [fmc]
     Cluster name            : [FMCGRP1]
   FMC33537I Select repository type ...:
     f ... (X) 'FMCQM1' is the first queue manager in cluster 'FMCGRP1'
     a ... ( ) 'FMCQM1' is an additional queue manager in cluster
'FMCGRP1'
-  FMC33749I Selected Repository type : 'FMCQM1' is the first queue
manager in cluster 'FMCGRP1'
   FMC33632I Transaction coordination will be used between MQSeries and
DB2.
   FMC33633I The queue manager 'FMCQM1' will connect to the database
'FMCDB1'.
     DB2 user ID of transaction coordinator      : [ChandraU]
-  Configuration of client ...
-  Configuration of Java Agent ...
-  FMC33749I Selected Locator Policy : Local bindings
   FMC33606I Specify information about garbage collection (reaper) ...:
     Agent cycle (in seconds)             : [300]
     Client threshold (number of objects) : [1000]
     Client cycle (in % of agent cycle)   : [90]
-  Configuration of Web Client ...
   FMC33777I Select application server ...:
     w ... ( ) WebSphere
     o ... (X) Other
     Code Version                         : [3300]
     c ... Create configuration profile for 'FMC1' now
     s ... Save input to file
     r ... Review/change input
     x ... Exit (input for configuration 'FMC1' will be lost)
c
Enter password for user ID 'ChandraU'   : [] ********
     Confirm password for user ID 'ChandraU' : [] ********
FMC33136I Generating database layout.
FMC33153W The managed by value for tablespaces belonging to group INDEX
is not c
ustomizable.
FMC33110I The database manager is already active.
FMC33115I Creating the database - FMCDB1
FMC33116I Please wait... This may take a while.
FMC33117I Database FMCDB1 has been created.
FMC33120I Updating the database configuration.
FMC33120I Updating the database configuration.
FMC33132I Creating tablespaces.
FMC33133I Creating tables.
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbact.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbadm.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbadt.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbad2.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbatr.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbblk.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbccn.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbctr.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbdcn.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbepi.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdblst.bnd
```

```
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbmat.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbmod.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbopr.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbprc.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbqmg.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbses.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbsgo.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbstf.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbsvs.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbtop.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbtpl.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbwcs.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbwit.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbwiv.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcpqe01.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcpqe02.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcpqe03.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcpqe04.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcpqe05.bnd
FMC33143W Warning during bind occurred. See file c:\program
files\mqseries workf
low\cfgs\fmc1\log\fmcpqe05.msg for details.
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcpqe06.bnd
FMC33143W Warning during bind occurred. See file c:\program
files\mqseries workf
low\cfgs\fmc1\log\fmcpqe06.msg for details.
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcpqe07.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcpqe08.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcpqe09.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcddsql.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbcln.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbntf.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbqry.bnd
FMC33126I Binding c:\program files\mqseries workflow\bnd\fmcdbwcs.bnd
FMC33130I Initializing the database.
FMC33003I fmczbstr -gFMCGRP1 -sFMCSYS1 -xFMC -dFMCDB1 -uChandraU
FMC24500I fmczbstr is starting.
FMC24560I fmczbstr finished and found 0 errors 0 warnings. RC = 0
FMC33131I Loading reference FDL.
FMC20500I Start parsing c:\program files\mqseries
workflow\cfgs\fmc1\fdl\fmczref
.fdl.
FMC25100I CREATE LEVEL '0' finished.
FMC25100I CREATE LEVEL '1' finished.
FMC25100I CREATE LEVEL '2' finished.
FMC25100I CREATE LEVEL '3' finished.
FMC25100I CREATE LEVEL '4' finished.
FMC25100I CREATE LEVEL '5' finished.
FMC25100I CREATE LEVEL '6' finished.
FMC25100I CREATE LEVEL '7' finished.
FMC25100I CREATE LEVEL '8' finished.
```

```
FMC25100I CREATE LEVEL '9' finished.
FMC25100I CREATE STRUCTURE 'Default Data Structure' finished.
FMC25100I REPLACE DOMAIN 'DOMAIN' finished.
FMC25100I REPLACE GROUP 'FMCGRP1' finished.
FMC25100I REPLACE SYSTEM 'FMCSYS1' finished.
FMC25100I REPLACE PERSON 'ADMIN' finished.
FMC25100I REPLACE ROLE 'System administrator' finished.
FMC25100I CREATE SERVER 'CLEANSVR.FMCSYS1.FMCGRP1' finished.
FMC25100I CREATE SERVER 'EXECSVR.FMCSYS1.FMCGRP1' finished.
FMC25100I CREATE SERVER 'SCHEDSVR.FMCSYS1.FMCGRP1' finished.
FMC25100I CREATE SERVER 'PESERVER.FMCSYS1.FMCGRP1' finished.
FMC25100I CREATE QUEUE_MANAGER 'FMCQM1' finished.
FMC20510I Finished parsing c:\program files\mqseries
workflow\cfgs\fmc1\fdl\fmcz
ref.fdl.
-  FMC33911I The new Runtime database FMCDB1 was created successfully.
-  Do you want to create the queue manager 'FMCQM1' now?
     y ... Yes
     n ... No
y
MQSeries queue manager created.
Creating or replacing default objects for FMCQM1.
Default objects statistics : 28 created. 1 replaced. 0 failed.
Completing setup.
Setup completed.
MQSeries queue manager 'FMCQM1' started.
MQSeries queue manager ending.
MQSeries queue manager ended.
Creating or replacing default objects for FMCQM1.
Default objects statistics : 1 created. 28 replaced. 0 failed.
Completing setup.
Setup completed.
Not executing on a domain controller
-  FMC33736I The queue manager FMCQM1 has been updated successfully.
FMC34010I: Configuration checker version 3.3.0.100 started.
FMC34600I: ==> Executing commands.
FMC34631I: The service 'MQ Workflow - FMC1' has been created
successfully.
FMC34100I: Messages have been written to c:\program files\mqseries
workflow\cfgs
\fmc1\log\@fmczchk.log.
FMC34999I: Configuration checker ended: 0 error(s), 0 warning(s), rc =
0.
```

For further information please refer to chapter nine, *Configuring on Unix*, in the *Installation Guide*.

This article concludes next month with a WebCredit test example.

Chandra Upadhyayula
Programmer Analyst, (USA)

How secure are your channels?

So, you've gone to great lengths to control who has access to your queues, but would you care if someone could see the contents of your messages as they were transported across the network to another queue manager? Have you thought about encrypting the traffic across your channels? Would you like to be able to authenticate the partner queue manager automatically? Would you like all this to be integrated with your channels and provided free with WebSphere MQ?

In the new release of WebSphere MQ (formerly MQSeries) V5.3, a protocol known as Secure Sockets Layer (SSL) has been introduced into MQ channels. It is a protocol widely used in many products which transport information across insecure networks; you are probably already using it with your Web browser, for example. The SSL protocol provides us with the security benefits of partner authentication, encryption, and message integrity.

BUILDING BLOCKS

The SSL protocol relies on several different technologies combined to provide these benefits. These building blocks are symmetric and asymmetric encryption techniques and hash functions, and are described below.

Symmetric encryption

Symmetric or shared key encryption is a relatively fast encryption mechanism but has problems which stem from the need to keep the key completely secret. Only the partners exchanging information can know the key; otherwise the encryption can be broken.

Asymmetric encryption

Asymmetric or public/private key encryption by comparison with symmetric key encryption is not as fast, but has the benefit that the public key portion of the public/private key pair can be published to the world. The encryption algorithm used is based upon a mathematical function known as a trapdoor function. A trapdoor function is one that cannot be reversed without the use of brute force (ie attempting every possible combination until the correct result is discovered).

Asymmetric encryption can be done using the public key by anyone, ensuring that only the owner of the private key can decrypt the message.

Asymmetric encryption can also be done using the private key by the owner of that key, ensuring that anyone can decrypt the message using the public key. This means that the message must have originated from the owner of the private key.

Hash function

A hash function is a way of producing a small fixed-size number that is a representation of the message being delivered. If the message is changed in any way the likelihood that the value produced by the hash function is the same is very small. This provides a means to detect tampering. The small fixed-size number is known as the hash value, the message authentication code (MAC), or the message digest.

COMBINING THESE BUILDING BLOCKS

These building blocks alone are not enough to provide a secure protocol but it is the manner in which they are combined that provides the security benefits that we want.

Digital signature

A hash function allows detection of tampering. However, someone tampering with a message could also tamper with the hash value so that it matched the new message. By encrypting the hash value using a private key the sender creates what is known as a digital signature. The receiver checks that the digital signature is valid by decrypting it using the corresponding public key pair to obtain the sender's hash value. This value is then compared with a hash value for the message received, which has been calculated by the receiver. If the values match, the message has not been tampered with and the origin of the message can be trusted.

Certificate

The use of public/private key pairs is clearly a very important part of the

SSL protocol. However, how can you be sure you trust the public key you are using? You may have been sent a public key to use but can you be sure you have the correct key?

In a ‘man-in-the-middle’ attack the public key you are sent is intercepted and replaced with the public key of the attacker. The solution to this problem is the use of items known as certificates. They are issued by a well known trustworthy third party known as a Certification Authority (CA) and contain the identity of the owner of the public/private key pair and the public key from that pair.

As certificates are delivered across an insecure network they can be thought of as just another message that could be tampered with. In order to protect against tampering with certificates, they are also signed with a digital signature made from the CA’s private key, which can be decrypted by anyone using the CA’s public key in order to ensure the validity of the certificate.

The identity in a certificate is stored in a particular format known as a Distinguished Name (DN). A DN contains several subfields allowing the identity, job description, and address of the entity to be specified.

Authentication

Partner authentication is achieved by both partners exchanging certificates and validating that these certificates have been correctly signed by the CA and can, therefore, be trusted. Optionally, the responder can choose to allow anonymous connections but the responder will always be authenticated. In other words you can have anonymous clients but you cannot have anonymous servers.

Encryption and message integrity

There are a variety of different symmetric key encryption algorithms to use and also several hash functions. The combination of these two algorithms is known as a CipherSpec and is negotiated as part of the handshake that creates a secure session using SSL. The problem of keeping the symmetric key secret between the two partners is solved in SSL by using asymmetric encryption to deliver the shared secret key, which is then used for all subsequent traffic on that session.

Certificates

To make use of SSL you must acquire a certificate for each party that wishes to communicate securely. In the case of WebSphere MQ this means one certificate for each queue manager when using MCA channels, and also one certificate for each logged-on user ID when using MQI channels.

The location of this certificate varies from platform to platform. It can be specified as a queue manager's certificate by using the **ALTER QMGR** command to indicate the location, or can be specified for use with a client application either by using an environment variable or through MQCONNX.

The label of the certificate is also important and indicates which queue manager or logged-on user ID owns the certificate. The label is the queue manager name or logged-on user ID appended to the end of *ibmWebSphereMQ*. This string is all in lower case on Unix platforms.

CipherSpecs

A particular encryption algorithm and hash function is known as a CipherSpec. CipherSpecs are specified on a per-channel basis using a parameter on the channel definition called SSLCipherSpec (SSLCIPH). For a secure channel to start up, both ends must specify the same CipherSpec.

Partner authentication

To specify whether this responding channel is willing to accept anonymous connections you use a parameter on the channel definition called SSLClientAuthentication (SSLCAUTH), which can be set to one of two values; REQUIRED or OPTIONAL.

Identity filtering

You may wish to ensure that only certain queue managers can connect to a particular channel. You can filter out unwanted entities by means of their DN. To specify the DNs that are allowed to connect you use a parameter on the channel definition called SSLPeerName (SSLPEER), which can contain a DN filter that is compared with the DN in the partner’s certificate.

BY EXAMPLE

In the following example we have a z/OS queue manager (MQ46) and an AIX queue manager (MORAGAIX) and we are going to set up secure channels between them. Instead of using a CA to sign the certificates for this example, we are going to use self-signed certificates. The only way to authenticate a self-signed certificate is to have a copy of it. So in each case, after the certificate has been created, we will ftp it to the other machine. This example has been tested as it appears here apart from the connection names, which have been changed to make clear which address needs to be used. Be sure to be accurate with the case of the certificate labels.

Certificate creation

We will show examples of the RACF commands needed to create a self-signed certificate on z/OS and how to use the iKeyMan tool to create a self-signed certificate on AIX. If you want to try this out on other platforms there are specific details in the *WebSphere MQ Security* manual.

Certificate creation on z/OS

Create a certificate in RACF using the following command (where the user ID we’re using (HUGHSON) must be the channel initiator user ID).

```
RACDCERT ID(HUGHSON) GENCERT
SUBJECTSDN(CN('Morag Hughson')
           T('Software Engineer')
           OU('WebSphere MQ Devt')
           O('IBM')
           L('Hursley')
           SP('Hampshire')
           C('UK'))
WITHLABEL('ibmWebSphereMQMQ46')
```

Export the certificate to a dataset so that it can be ftp’d to the AIX box, using:

```
RACDCERT ID(HUGHSON) EXPORT(LABEL('ibmWebSphereMQMQ46'))
DSN('HUGHSON.ZOS.CERT') FORMAT(CERTB64)
```

Create a keyring in RACF using the following command:

```
RACDCERT ID(HUGHSON) ADDRING(MQ46RING)
```

Connect the certificate you created to the keyring using the following command:

```
RACDCERT ID(HUGHSON) CONNECT(ID(HUGHSON)
LABEL('ibmWebSphereMQMQ46') RING(MQ46RING) USAGE(PERSONAL))
```

Once you have ftp'd your AIX queue manager's certificate to z/OS, import it into RACF using the following command (you may want to come back to this point later):

```
RACDCERT ID(HUGHSON) ADD('HUGHSON.AIX.CERT')
WITHLABEL('ibmWebSphereMQMORAGAIX')
```

and connect it to your keyring using the following command:

```
RACDCERT ID(HUGHSON) CONNECT(ID(HUGHSON)
LABEL('ibmWebSphereMQMORAGAIX') RING(MQ46RING) USAGE(PERSONAL))
```

Check that your keyring looks as it should with the following command:

```
RACDCERT ID(HUGHSON) LISTRING(MQ46RING)
```

which gives the following output:

```
Digital ring information for user HUGHSON:

   Ring:
        >MQ46RING<
   Certificate Label Name        Cert Owner     USAGE       DEFAULT
   ----------------------        ----------     -----       -------
   ibmWebSphereMQMQ46            ID(HUGHSON)    PERSONAL      NO
   ibmWebSphereMQMORAGAIX        ID(HUGHSON)    PERSONAL      NO
```

Certificate creation on AIX

Before starting the iKeyMan tool you must set an environment variable:

```
export JAVA_HOME=/usr/mqm/ssl/jre
```

Then start the iKeyMan tool using the command **gsk6ikm**.

Make a new Key Database File of type CMS with a file name of *key.kdb* and a location of */var/mqm/qmgrs/MORAGAIX/ssl* and when prompted for a password make sure you check the box to stash it to a file.

Now create a new self-signed certificate with a key label of *ibmwebspheremqmoragaix*. Take special care to have the key label all in lower case. Use a version of X509 V3, fill in common name and organization at the very least, and select your country.

Now extract the certificate you have just created to a file (I used *cert.arm*) so that it can be ftp'd to your z/OS system. I used a data type of Base64 encoded ASCII data**.**

Now ftp to your z/OS system and put your AIX certificate there and retrieve your z/OS certificate. Use the following ftp settings:

```
ftp> ascii
ftp> quote site recfm=vb
ftp> put cert.arm AIX.CERT
ftp> get ZOS.CERT zoscert.arm
```

Select 'Signer Certificates' instead of 'Personal Certificates' and add your z/OS certificate file, giving it a label.

Ensure the mqm group has read permissions on the *key.kdb* and *key.sth* files you have just created.

Remember to go back to the z/OS certificate management steps and import the AIX certificate into RACF.

MQ definitions

We will show examples of MQSC commands to set up the MQ definitions on a z/OS queue manager and on an AIX queue manager.

MQ definitions on z/OS

Specify the location of the queue manager's certificate and define some necessary TCBs for use in the channel initiator address space.

```
ALTER QMGR SSLKEYR(MQ46RING) SSLTASKS(5)
```

Add *sslhlq.SGSKLOAD* to your CHINIT STEPLIB and start the channel initiator and a listener.

```
START CHINIT
START LSTR TRPTYPE(TCP) PORT(1546)
```

Define a sender channel to the AIX queue manager.

```
DEFINE CHANNEL(TO.MORAGAIX) CHLTYPE(SDR) TRPTYPE(TCP) CONNAME('aix-
mach(1414)') XMITQ(MORAGAIX) SSLCIPH(RC4_MD5_US)
DEFINE QLOCAL(MORAGAIX) USAGE(XMITQ)
```

Define a receiver channel.

```
DEFINE CHANNEL(TO.MQ46) CHLTYPE(RCVR) TRPTYPE(TCP)
SSLCIPH(DES_SHA_EXPORT) SSLPEER('OU="WebSphere MQ Devt"')
```

MQ definitions on AIX

Enter the following MQSC commands using **runmqsc MORAGAIX**.

We have used the default location of the key database file. You can check it using:

```
1 : display qmgr sslkeyr
AMQ8408: Display Queue Manager details.
SSLKEYR(/var/mqm/qmgrs/MORAGAIX/ssl/key)
QMNAME(MORAGAIX)
```

Define a sender channel to the z/OS queue manager.

```
DEFINE CHANNEL(TO.MQ46) CHLTYPE(SDR) TRPTYPE(TCP) CONNAME('mvs-
mach(1546)') XMITQ(MQ46) SSLCIPH(DES_SHA_EXPORT)
DEFINE QLOCAL(MQ46) USAGE(XMITQ)
```

Define a receiver channel

```
DEFINE CHANNEL(TO.MORAGAIX) CHLTYPE(RCVR) TRPTYPE(TCP)
SSLCIPH(RC4_MD5_US) SSLPEER('OU="WebSphere MQ Devt"')
```

Remember to start your listener.

```
runmqlsr -m MORAGAIX -t tcp -p 1414
```

Start your secure channels

Once you have created all these definitions, and have remembered to ftp your self-signed certificate to the other machine and add it to the relevant key repository, you are ready to start your secure channel.

Start your secure channel from the z/OS queue manager to the AIX queue manager and display its channel status once started.

```
START CHANNEL(TO.MORAGAIX)
DISPLAY CHSTATUS(TO.MORAGAIX) RQMNAME SSLPEER
CSQM201I !MQ46 CSQXDMSG  DIS CHSTATUS DETAILS 518
CHSTATUS(TO.MORAGAIX)
CHLDISP(PRIVATE)
XMITQ(MORAGAIX)
CONNAME(xxx.xxx.xxx.xxx)
CURRENT
STATUS(RUNNING)
STOPREQ(NO)
```

```
RQMNAME(MORAGAIX)
SSLPEER(CN=Morag Hughson;OU=WebSphere MQ
Devt;O=IBM;L=Hursley;SP=Hampshire;C=GB;)
CSQXDMSG END CHSTATUS DETAILS
```

Start your secure channel from the AIX queue manager to the z/OS queue manager and display its channel status once started.

```
START CHANNEL(TO.MQ46)
     3 : dis chs(TO.MQ46) RQMNAME SSLPEER
AMQ8417: Display Channel Status details.
   CHANNEL(TO.MQ46)                        XMITQ(MQ46)
   CONNAME(winmvs41(1546))                 CURRENT
   CHLTYPE(SDR)                            STATUS(RUNNING)
   SSLPEER(CN=Morag Hughson,T=Software Engineer,OU=WebSphere MQ
Devt,O=IBM,L=Hursley,ST=Hampshire,C=UK)
   RQMNAME(MQ46)
```

Note that in the channel status you can see the DN from the certificate of the partner so you can see exactly who is connected to your channel.

CONCLUSION

WebSphere MQ V5.3 provides a mechanism to secure your channels that is integrated into the product. It does not involve any changes to your MQ applications in order to use it. It is set up as part of your channel configuration and provides a built-in mechanism to prevent eavesdropping, tampering, and impersonation using a well known and extensively tested security protocol.

I have provided just a basic introduction to the new features; more detail can be found in the *WebSphere MQ Security* manual that is part of the WebSphere MQ V5.3 product.

Morag Hughson
WebSphere MQ for z/OS Development, IBM Hursley (UK)

Checking Solaris Kernel parameters automatically

I got tired of manually checking the kernel parms on our Sun Solaris machines so I automated it. The enclosed Korn Shell script is based on IBM’s latest recommended values for MQSeries servers.

It looks in both */etc/system* and *sysdef -i*, compares the values against IBM's, and prints out a nice table. These values were originally given in the *MQSeries Quick Beginnings* book for Solaris and subsequently updated by IBM.

Here's what the output of the script looks like.

$ KERNELPARMS.PL

```
Parm    Std     Sysdef  /etc
msgmap  1026    0       1026
msgmax  4096    0       2048    LOW!
msgmnb  4096    0       none    LOW!
msgmni  50      0       none    LOW!
msgseg  2048    0       none    LOW!
msgssz  8       0       none    LOW!
msgtql  1024    0       none    LOW!
semaem  16384   16384   16384
semmap  1026    1026    1026
semmni  1024    160     1024
semmns  32767   16384   16384   LOW!
semmnu  2048    2048    2048
semmsl  128     200     200
semopm  128     100     100     LOW!
semume  256     256     256
semvmx  23767   32767   none
shmmax  4194304 4294967295      4294967295
shmmni  1024    1024    1024
shmseg  1024    54      54      LOW!
```

SOURCE CODE OF SCRIPT (KERNELPARMS.PL)

```
#!/usr/local/bin/perl
use strict;
# IBM Recommended Values
my %StdParm = (
    msgmap => 1026,
    msgmax => 4096,
    msgmnb => 4096,
    msgmni => 50,
    msgseg => 2048,
    msgssz => 8,
    msgtql => 1024,
    semaem => 16384,
    semmap => 1026,
    semmni => 1024,
    semmns => 32767,
    semmnu => 2048,
    semmsl => 128,
    semopm => 128,
    semume => 256,
    semvmx => 32767,
    shmmax => 4194304,
    shmmni => 1024,
    shmseg => 1024,
    );
# Declare hashes to store system values
my %Sysdef = ();
my %EtcSystem = ();
# Only  valid for SunOS
my $OS = `uname`;
die "$0: Not a Solaris server $OS\n" unless $OS eq
"SunOS\n";
# Go get values from /etc/system
open(SYSTM, "</etc/system") or die "$0: No /etc/system
file found\n";
while (<SYSTM>) {
    chomp;
    s/\s*//g;
    my @Foo = split /[_=]/;
    my ($Value, $Parm) = reverse @Foo;
    $EtcSystem{$Parm} = $Value if $StdParm{$Parm};
}
close SYSTM;
# Go get values from sysdef -i
foreach (`/etc/sysdef -i`) {
    chomp;
    s/^\s*//;
    s/\)//g;
    my @Bar = split /[\s(]/;
    my ($Value) = @Bar;
    my $Parm = lc(pop @Bar);
    $Sysdef{$Parm} = $Value if $StdParm{$Parm};
}
# Print values
#
print "Parm\tStd\tSysdef\t/etc\n";
foreach (sort keys %StdParm) {
    print "$_\t$StdParm{$_}\t",
        exists $Sysdef{$_} ? $Sysdef{$_} : "none",
        "\t",
        exists $EtcSystem{$_} ? $EtcSystem{$_} : "none";
    my $Greatest =  $Sysdef{$_} > $EtcSystem{$_} ?
$Sysdef{$_} : $EtcSystem{$_};
    print "\tLOW!" if $Greatest < $StdParm{$_};
    print "\n";
}
```

T Robert Wyatt (USA) © Xephon 2002

MQ news

Middleware management tools provider Nastel Technologies has introduced AutoPilot for WebSphere/MQ, a suite of management technologies designed for IBM WebSphere/MQ middleware components.

Currently available components include AutoPilot for WebSphere/MQ, AutoPilot for WebSphere/MQSI, and AutoPilot for WebSphere/WebSphere Application Server.

AutoPilot for WebSphere/MQ provides a centralized dashboard to control, configure, monitor, and automate MQSeries and WebSphere/MQ-based distributed middleware infrastructures.

It features a 'self-healing' management environment designed to catch problems before they affect operations, and it provides graphical views of business events and processes to determine the operational impact of middleware on related applications.

For more information contact:
Nastel, 48 South Service Road, Melville, New York 11747, USA.
Tel: +1 800 375 6144.
Fax: +1 631 761 9101.
Web: http://www.nastel.com

Nastel Technologies, 3 Tannery House, Tannery Lane, Send, Surrey, GU23 7EF, UK.
Tel: + 44 207 872 5412.
Fax: + 44 207 753 2848.

* * *

The recently published IBM Redbook SG24-6509-00, entitled *WebSphere MQ Integrator Deployment and Migration,* is now available from the Redbooks home page at *http://ibm.com/redbooks.*

Authors of this Redbook are: Geert Van de Putte, Chris Griego, Brian McCarty, Peter Toogood, and Ralf Ziegler.

* * *

IBM has announced a portfolio of adapters for use with its integration middleware. WebSphere Business Integration Adapters connect a customer's various application software packages to IBM's WebSphere Business Integration, WebSphere MQ Integrator (formerly MQSeries Integrator), or IBM CrossWorlds products.

For more information contact your local IBM representative.
Web: http://www.ibm.com/websphere.

* * *

xephon