42

MQ update

December 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.

Controlling access to MQSC facilities

In many organizations where MQ is installed as part of the corporate architecture there are often battles between application developers and the central MQ support team over access rights. On the one hand developers have a legitimate right to query what is going on within their application, whilst, on the other, central support people are usually reluctant to grant these developers group mqm access. This caution is understandable because any ID within that group becomes a super-user as far as MQ is concerned and could even delete the entire queue manager!

In recognition of this problem, and in an attempt to overcome it, IBM released SupportPac MS0E – *The MQSeries Administration Wrapper*.

After extracting the downloaded tar file there will be three files: the runmqadm executable, a configuration file, and an installation shell script, which copies files into their correct locations and sets the appropriate file permissions.

The config file is key to the utility because it sets appropriate authority levels for each potential MQSC command. It also assigns a security level to each designated user and a default for those not explicitly catered for. The authority level is expressed as a number from zero to nine with zero being the lowest and nine the highest (although the sample file that ships with it only uses zero to three). I find that the default config file that is part of the utility is a little too cautious and so I make quite a few changes to it.

One thing that developers frequently want to do is display all of the attributes of a queue (especially its current depth) or the status of a particular channel. As this is a read-only function I always make DISPLAY commands level zero – ie anyone can issue them.

This wrapper can also be used to allow users access to key primary MQ commands and utilities such as strmqm or strmqcsv.

TYPICAL CONFIG FILE

```
# RUNMQADM Configuration File
#*  Statement:     Licensed Materials - Property of IBM             *#
#*                 MSØE  - MQSeries Administration Wrapper          *#
#*                 (C) Copyright IBM Corp. 2ØØØ                     *#
# Stanzas in here define
# - some global settings
# - authorities for all the known MQSC commands
# - authorities for all the known MQSeries commands
# - authorities for user-specified commands
# - authority level for each user
GLOBAL:
  # Where's the log file to be put?
  LOGFILE = /var/mqm/config/runmqadm.log
  # For unknown users - what authority do they have? None by default.
  DEFAULTAUTHLEVEL = Ø
SCCOMMANDS:
          # What's the authority for unlisted MQSC commands?
          default = 3
          # Here are all the commmands known in MQSeries V5.1
          # Even OS/39Ø-specific commands are in here as we may be
          # doing remote administration.
          ALTER_CHANNEL    = 3
          ALTER_NAMELIST   = 3
          ALTER_PROCESS    = 3
          ALTER_QALIAS     = 3
          ALTER_QLOCAL     = 3
          ALTER_QMGR       = 3
          ALTER_QMODEL     = 3
          ALTER_QREMOTE    = 3
          ALTER_SECURITY   = 3
          ALTER_STGCLASS   = 3
          ALTER_TRACE      = 3
          ARCHIVE_LOG      = 3
          CLEAR_QLOCAL     = 2
          DEFINE_BUFFPOOL  = 3
          DEFINE_CHANNEL   = 3
          DEFINE_MAXSMSGS  = 3
          DEFINE_NAMELIST  = 3
          DEFINE_PROCESS   = 3
          DEFINE_PSID      = 3
          DEFINE_QALIAS    = 3
          DEFINE_QLOCAL    = 3
          DEFINE_QMODEL    = 3
          DEFINE_QREMOTE   = 3
          DEFINE_STGCLASS  = 3
          DELETE_CHANNEL   = 3
          DELETE_NAMELIST  = 3
          DELETE_PROCESS   = 3
          DELETE_QALIAS    = 3
          DELETE_QLOCAL    = 3
          DELETE_QMODEL    = 3
          DELETE_QREMOTE   = 3
          DELETE_STGCLASS  = 3
     DISPLAY_CHANNEL  = Ø
     DISPLAY_CHSTATUS = Ø
     DISPLAY_CLUSQMGR = Ø
     DISPLAY_CMDSERV  = Ø
     DISPLAY_DQM      = Ø
     DISPLAY_MAXSMSGS = Ø
     DISPLAY_NAMELIST = Ø
     DISPLAY_PROCESS  = Ø
     DISPLAY_QALIAS   = Ø
     DISPLAY_QCLUSTER = Ø
     DISPLAY_QLOCAL   = Ø
     DISPLAY_QMGR     = Ø
     DISPLAY_QMODEL   = Ø
     DISPLAY_QREMOTE  = Ø
     DISPLAY_QUEUE    = Ø
     DISPLAY_SECURITY = Ø
     DISPLAY_STGCLASS = Ø
     DISPLAY_THREAD   = Ø
     DISPLAY_TRACE    = Ø
     DISPLAY_USAGE    = Ø
     PING_CHANNEL     = Ø
     PING_QMGR        = Ø
     RECOVER_BSDS     = 3
          REFRESH_CLUSTER  = 3
          REFRESH_SECURITY = 3
          RESET_CHANNEL    = 3
          RESET_CLUSTER    = 3
          RESET_TPIPE      = 3
          RESOLVE_CHANNEL  = 3
          RESOLVE_INDOUBT  = 3
          RESUME_QMGR      = 3
          RVERIFY_SECURITY = 3
          START_CHANNEL    = 3
          START_CHINIT     = 3
          START_CMDSERV    = 3
          START_LISTENER   = 3
          START_QMGR       = 3
          START_TRACE      = 3
          STOP_CHANNEL     = 3
          STOP_CHINIT      = 3
          STOP_CMDSERV     = 3
          STOP_LISTENER    = 3
          STOP_QMGR        = 3
          STOP_TRACE       = 3
          SUSPEND_QMGR     = 3
OSCOMMANDS:
          # What's the authority for unlisted MQSeries commands?
          default = 3
          # Here are all the commmands known in MQSeries V5.1
      # Anyone can start runmqsc, as we check individual commands
later.
          runmqsc   = Ø
          crtmqm    = 3
          dltmqm    = 3
          dmpmqlog  = 3
          dspmqaut  = 3
          dspmqcsv  = 3
          dspmqfls  = 3
          dspmqtrc  = 3
          dspmqtrn  = 3
          endmqcsv  = 3
          endmqlsr  = 3
          endmqm    = 3
          endmqtrc  = 3
          rcdmqimg  = 3
          rcrmqobj  = 3
          rsvmqtrn  = 3
          runmqchi  = 3
          runmqchl  = 3
          runmqdlq  = 3
          runmqlsr  = 3
          runmqtmc  = 3
          runmqtrm  = 2
          setmqaut  = 3
          strmqcsv  = 3
          strmqm    = 3
          strmqtrc  = 3
          # Any extra locally-defined commands can be configured here.
          # Use a fully-qualified name unless it's in the same
directory
          # as the MQSeries commands (eg /opt/mqm/bin)
          /tmp/unknowncmd = 1
          /usr/lpp/mqm/samp/bin/amqsput= 1
     amqsput  = 2
     /opt/mqm/samp/bin/amqsput = 2
     endmqtrm = 2
          crtmqcvx = Ø
USERS:
          mqm = 3
     # App 1
     u763362 = 1
     u832ØØ2 = 1
     u759556 = 1
     u834321 = 1
     # App 2
     u87845Ø = 1
     oramap = 1
     u834135 = 1
     n4324Ø = 1
     # Test Group
     u6654ØØb=2
     u6654ØØc=3
```

You can see that the default authority for a user not designated in the 'USERS' stanza is zero, so this gives any system user the ability to DISPLAY the attributes of all objects within the queue manager. It also allows them to PING both channels and queue manager.

Once the utility has been correctly installed you can launch it by issuing the command **runmqadm**. Once within this command shell you can either use **runmqsc** or enter any other primary MQ command that you are authorized for.

Here are some examples.

```
$ runmqadm
MSØE: MQSeries Administration Wrapper
(C) Copyright IBM Corp. 2ØØØ. ALL RIGHTS RESERVED
Username            :  u6654ØØd
Authorization Level:  Ø
MQADM >
```

Here we see that this user is not specifically listed within the config file and so has a default authority level of zero. They can **runmqsc** and display the attributes of various objects – but not alter or delete them.

```
MQADM >runmqsc SMPD9.BAW
Ø783889, 5765-B75 (C) Copyright IBM Corp. 1994, 2ØØØ.  ALL RIGHTS
RESERVED.
Starting MQSeries Commands.
RMQSC >display process(*)
     4 : display process(*)
AMQ84Ø7: Display Process details.
   PROCESS(NJP.PROC)
AMQ84Ø7: Display Process details.
   PROCESS(OCD.CHANGE_TYPE)
AMQ84Ø7: Display Process details.
   PROCESS(OCDPDT.CHANGE_TYPE)
AMQ84Ø7: Display Process details.
   PROCESS(OCDPDT.PROCESS)
AMQ84Ø7: Display Process details.
   PROCESS(OW.PROCESS)
AMQ84Ø7: Display Process details.
   PROCESS(SYSTEM.DEFAULT.PROCESS)
RMQSC >
RMQSC >alter qlocal(SYSTEM.DEFAULT.LOCAL.QUEUE) defpsist(yes)
```

```
RUNMQADM (E): Command is not authorized or not recognized
RMQSC >end
$
```

Another major benefit of running MQSC commands under this wrapper is that all actions within it are captured and written to a log. This can provide a useful audit trail for the queue manager. Here are the log entries for the actions above:

```
[18680] u665400d         Thu Jul 11 13:57:56 2002 1
** Started runmqadm session **
[18680] u665400d         Thu Jul 11 13:58:03 2002 1
runmqsc SMPD9.BAW
[18680] 0783889, 5765-B75 (C) Copyright IBM Corp. 1994, 2000.  ALL
RIGHTS RESER.
Starting MQSeries Commands.
[18680] u665400d         Thu Jul 11 13:58:08 2002 1
display process(*)
[18680]      1 : display process(*)
AMQ8407: Display Process details.
   PROCESS(NJP.PROC)
AMQ8407: Display Process details.
   PROCESS(OCD.CHANGE_TYPE)
AMQ8407: Display Process details.
   PROCESS(OCDPDT.CHANGE_TYPE)
AMQ8407: Display Process details.
   PROCESS(OCDPDT.PROCESS)
AMQ8407: Display Process details.
   PROCESS(OW.PROCESS)
AMQ8407: Display Process details.
   PROCESS(SYSTEM.DEFAULT.PROCESS)
[18680] u665400d         Thu Jul 11 13:58:20 2002 0
alter qlocal(SYSTEM.DEFAULT.LOCAL.QUEUE) defpsist(yes)
** NOT AUTHORIZED **
[18680] u665400d         Thu Jul 11 13:58:23 2002 1
end
[18680]      2 : end
One MQSC command read.
No commands have a syntax error.
All valid MQSC commands were processed.
[18680] u665400d         Thu Jul 11 13:58:26 2002 1
end
$
```

In conclusion, you can see that this admin wrapper allows you to give controlled and audited access to objects within a queue manager without giving group mqm or having to write utility PCF programs to get the information.

Chris Bell, Systems Consultant
British Airways (UK)

Using publish/subscribe in WebSphere MQ Integrator

OVERVIEW

Publish/subscribe is a function of WebSphere MQ Integrator (MQSI) V2 that allows a system to send a message to multiple recipients without needing to know who those recipients are. Thus, the sending application can publish a message to a pre-defined topic and whoever needs that message can subscribe to it. When new subscriptions are made to that topic the publishing application doesn't need to do anything to support it. Contrast this with a distribution list or a message flow, which needs to be modified each time a receiving queue is added or dropped.

MQSI directly supports the process of defining topics, and publishing messages to a topic isn't difficult once you know the process. However, MQSI offers absolutely no support (as of release 2.02) for subscribing to a topic and what documentation exists is extremely piecemeal.

This article attempts to remedy the situation by taking the reader by the hand and explaining each step of the process from publishing to subscribing.

This article assumes that you are using MQSI V2.02 or that your version supports RFH2 headers. It also assumes that your client environment is Windows NT 4.0 or later. The sample code is Visual Basic 6.0.

If you don't meet these requirements you may still get some value from the discussion of the underlying architecture, but I cannot promise that the exact same techniques will work in your environment.

Defining topics

To publish a message you must first have a topic defined to which you can publish. The process of creating a topic is well explained in the MQSI documentation but I will summarize it here.

The MQSI Control Centre has a tab labelled 'Topics', where all the topics are organized hierarchically as a tree structure. You can add a new topic underneath TopicRoot or you can add it to an existing topic. In the latter case it will act as a subset of the parent topic. To add a new topic:

- Check out the parent topic. Right-click it, and then select Check Out from the pop-up menu.
- Right-click the parent topic, and select Create/Topic… from the pop-up menu.
- You will get a dialog box where you can name your flow, define the access control list, and set permissions.

The MQSI message flow

Create a message flow in MQSI consisting of an input node, a compute node, and a publication node. The input node is assigned to the queue that you are publishing from and its Message Domain property should be set to XML. The publish node needs no property settings at all unless you wish to use subscription points. Put the following code into the compute node but use your own topic name.

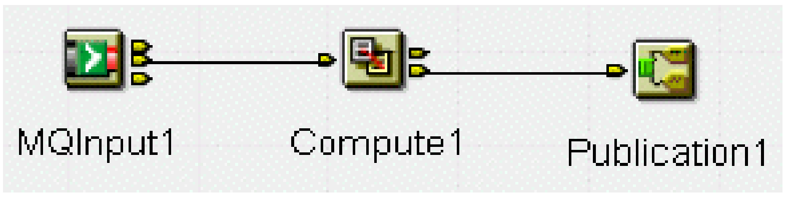

Figure 1: A sample publishing message flow

```
SET OutputRoot = InputRoot;
-- Enter SQL below this line. SQL above this line might be
regenerated, causing any modifications to be lost.
Set OutputRoot.Properties.Topic = 'FOO/BAR';
```

You might be wondering where the Properties node comes from in this example. The properties block, in addition to the MQMD and RFH2 blocks, specifies various pieces of header information about the message.

Once you deploy this message flow (see Figure 1) all messages put onto the input queue will be published to the topic you specify.

SUBSCRIBING TO TOPICS

Required information

When subscribing to a topic you must be prepared to supply certain pieces of information. While most of this may seem self-explanatory, too much MQSI documentation has lost value by assuming too much.

- *Topic* – required. This may be either a fully qualified topic name (ie Topic/Subtopic/Sub-subtopic) or else a wildcard expression where the hash sign (#) means 'give me all subtopics' (ie Topic/#).
- *Subscription point* – optional. When publishing, a message may be further qualified by specifying a subscription point. To get only those messages within a topic that have a particular subscription point you must specify it in the subscription.
- *Filter* – optional. You can specify a content filter using a subset of the filter node ESQL syntax.
- *MQSI queue manager* – this is the MQSeries Integrator queue manager supplying the topics, where the subscription will reside.
- *Subscriber queue manager* – this is the queue manager for the queue that will receive the published messages. This does not need to be the same as the MQSI queue manager.

- *Subscriber queue* – this is the name of the queue that the subscription will send the messages to.

To subscribe to a topic you must send a specially formatted message to the system queue SYSTEM.BROKER.CONTROL.QUEUE. The message you send contains a special data block in the MessageData buffer of the MQMD object called an RFH2 header. Following this header is an XML command message.

If you are used to using the MQSeries ActiveX interface (MQAX200) I recommend that you switch to the MQSeries Win32 API functions because they are more stable when handling binary data. Specifically, the ActiveX interface modified long integers embedded in the RFH2 header, causing the subscription message to fail; the Win32 API has given me no such problem.

Variable name	Data type	Length	Description
StrucId	String	4	"RFH "
Version	Long	4	2
StrucLength	Long	4	36 + NameValueLength
Encoding	Long	4	x'222'
CodedCharSetId	Long	4	-2
Format	String	8	Spaces
Flags	Long	4	0
NameValueCCSID	Long	4	1208 (UTF-8 or Unicode)
NameValueLength	Long	4	Length of the command XML padded to a multiple of four.
Command XML	String		An XML string containing the subscribe command, plus padding to make the length evenly divisible by four.

Table 1: The RFH2 header

The RFH2 header

No matter how large your RFH2 header is you must pad it so that its total length is a multiple of four. The header is structured in Table 1 below. If you are trying to adapt this to a non-Windows environment you will probably need different settings for the Encoding and CCSID fields.

The Visual Basic version of the RFH2 header is shown here.

```
Public Type RFH2
     StrucId         As String * 4   ' "RFH "
     Version         As Long         ' 2
     StrucLength     As Long         ' 36 + Len(xml command).
     Encoding        As Long         ' &H222
     CodedCharSetId  As Long         ' -2
     Format          As String * 8   ' Blank
     Flags           As Long         ' Zero
     NameValueCCSID  As Long         ' 12Ø8 = UTF-8 (Unicode)
     NameValueLength As Long         ' Length of XML command
                                     ' string (multiple of 4).
  End Type
```

IBM supplies in its sample code a C-include file called *cmqc.h,* which defines the RFH2 header and its related constants. For a more detailed description see the *MQSeries Integrator Programming Guide, Chapter 4 – MQRFH2 rules and formatting header*.

The PSC command structure

There are several PSC commands but let’s focus on the RegSub command, which is used to create (register) a new subscription. A sample RegSub command looks similar to this:

```
   <psc>
   <Command>RegSub</Command>
   <Topic>Sport/Soccer/State/LatestScore/#</Topic>
   <QMgrName>foo</QMgrName>
   <QName>bar</QName>
   <SubPoint/>
   <Filter/>
   <RegOpt>PubOnReqOnly</RegOpt>
   <RegOpt>CorrelAsId</RegOpt>
   </psc>
```

There doesn’t appear to be any DTD or schema associated with

Tag Name	Req'd	Occurs	Description
psc	Y	1	This is the root element of the XML structure. Note that it is all lower case
Command	Y	1	"RegSub"
Topic	Y	1 - M	The topic you are subscribing to. Wildcards are supported. See subsequent discussion on wildcard syntax
SubPoint	N	1	Name of the subscription point. May be omitted if not used
Filter	N	1	Content filter written in a subset of the Filter node ESQL. See subsequent discussion on syntax differences
RegOpt	N	0 - M	Registration options. See the *MQSeries Programming Guide* for details
QMgrName	N	1	Name of the queue manager for the subscribing queue. If not specified, defaults to the ReplyToQMgr property of the MQMD. If that is not specified, it defaults to the MQSI queue manager
QName	N	1	Name of the subscribing queue. If not specified, the ReplyToQ property of the MQMD must specify the subscribing queue

Table 2: RegSub tags

Property	Value
Format	MQFMT_RF_HEADER_2 ("MQHRF2 ")
MsgType	MQMT_REQUEST (1)
ReplyToQ	This is the queue that will receive a response from the MQSI broker indicating the status of your request

Table 3: MQMD property settings

the command structure since the tags can appear in any order and most are optional. The tags are described in Table 2.

The RegSub command, along with all other PSC commands, is described in detail in the *MQSeries Programming Guide, Chapter 5 – Publish/Subscribe Command Messages*.

MQMD settings

The MQMD object must be set differently from a normal message in order for an RFH2 command to process properly. The required property changes are shown in Table 3.

Evaluating the results of a subscribe request

The MQSI broker puts a PSCR response message to each RFH2

Reason code	Reason name
2336	MQRC_RFH_COMMAND_ERROR
2337	MQRC_RFH_PARM_ERROR
2338	MQRC_RFH_DUPLICATE_PARM
2339	MQRC_RFH_PARM_MISSING
3072	MQRCCF_TOPIC_ERROR
3074	MQRCCF_Q_MGR_NAME_ERROR
3076	MQRCCF_Q_NAME_ERROR
3079	MQRCCF_INCORRECT_Q
3080	MQRCCF_CORREL_ID_ERROR
3081	MQRCCF_NOT_AUTHORIZED
3083	MQRCCF_REG_OPTIONS_ERROR
3150	MQRCCF_FILTER_ERROR
3151	MQRCCF_WRONG_USER

Table 4: Reason codes applicable to the subscribe function

request that it processes into the queue specified in the MQMD ReplyToQ property. This message is in XML format and is structured as follows:

```
<pscr>
    <Completion>ok</Completion>
</pscr>
<pscr>
    <Completion>error</Completion>
    <Response>
        <Reason>315Ø</Reason>
    </Reponse>
</pscr>
```

In addition to the standard set of MQSeries reason codes the reasons shown in Table 4 can also be used.

Viewing/deleting subscriptions

The MQSI Control Centre contains a Subscriptions tab. Whilst you can't create new subscriptions here you can view existing subscriptions and delete the ones that belong to you.

To delete a subscription, right-click on it and select Delete from the pop-up menu.

CONTENT FILTERING

When you subscribe to a topic you can also specify a content filter, which will screen out messages in which you have no interest. An example of this might be a message containing a system code. If there are messages for multiple systems and you are only interested in messages for your system you can specify a filter that only accepts messages with your system code.

Differences from the filter node

If you have worked with the filter node in MQSI message flows you will be familiar with filter syntax. However, subscription filters operate on a subset of this syntax, so functions you may be used to using may not apply here. Here is a summary of language elements that are not supported by subscription filters:

- Path element '*'.
- Array element 'LAST'.
- Explicit CAST operation.
- Only the Root, Body, and Properties correlation names are supported.

For more information on content filtering, see the *MQSeries Programming Guide, Appendix A – Using filters in content-based routing*.

Issues

In my own testing I have found that if a filter is coded wrongly it may affect not only your subscription but every other subscription to the same topic. The test I used had a filter similar to this:

```
FieldA <> ''
```

A subscription with a filter like this blocked all messages from the topic I subscribed to for all subscriptions. When I removed that subscription, the rest of the subscriptions worked normally again.

With this kind of sensitivity you should probably restrict the use of content filtering to an 'expert' who knows how to do it correctly.

VISUAL BASIC CODE SAMPLE

I have included a sample program, which can be found at http://www.xephon.com/extras/pub-sub.txt as a practical example of how to write a subscribing application. The language I used is Visual Basic 6.0. In general, VB6 is not an ideal language for handling binary data blocks and I did have to jump through some hoops to make it work, but you may not always have a C++ or Java programmer available.

For the sake of space I have included just what is necessary to create a subscription. Subscription options, form validation, and error handling are excluded but a competent VB programmer can add those features on their own.

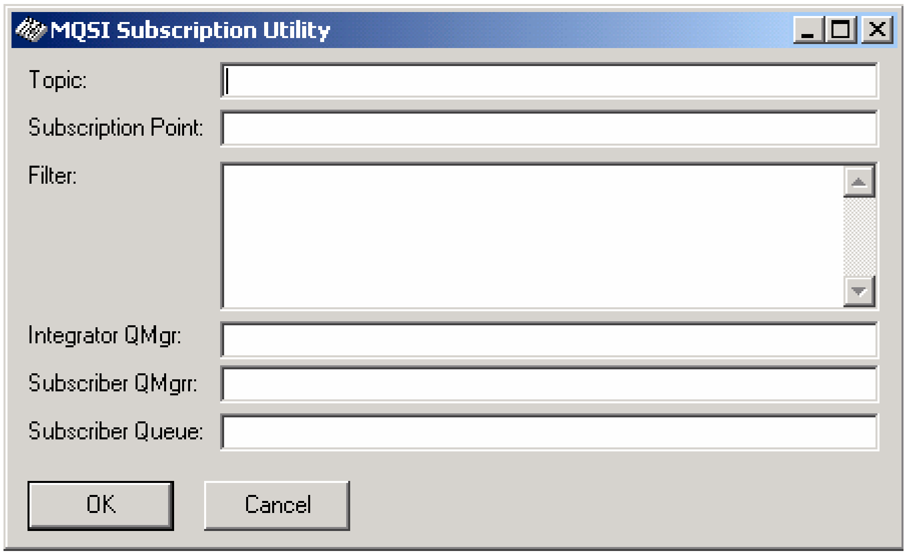

Figure 2: Form layout for the subscription program

Object	Property	Value
Form	Name	frmMain
	Caption	MQSI Subscription Utility
Topic text box	Name	txtTopic
Subscription Point	Name	txtSubPoint
Filter text box	Name	txtFilter
	MultiLine	True
	ScrollBars	2 - Vertical
Integrator QMgr	Name	txtMQSIQmgr
Subscriber QMgr	Name	txtQMgr
Subscriber Queue	Name	txtQueue
OK Button	Name	cmdOK
Cancel Button	Name	cmdCancel

Table 5: Property settings for the subscribe application

Setup

1 Create a new standard EXE project and give it an appropriate name.

2 In the Project/References… dialog window, select Microsoft XML, V2.0. If you don’t have this parser version installed use what you have, but you may have to adjust your class and object names accordingly.

3 Include the module CMQB.BAS. This is supplied with the MQSeries client and includes the API and constant declarations you will need. (If you are tempted to use MQAX200 instead be aware that I have seen it cause problems by corrupting the RFH2 headers you are trying to build.)

4 In the project properties window, on the Make tab, add ‘MqType = 2’ to the conditional Compilation Arguments field if you want this program to run on a client workstation. Use ‘MqType = 1’ if you want it to run on an MQ server.

5 Set up your form as shown in Figure 2.

Most fields on the form should be self-explanatory. The Topic, Subscription Point, and Filter fields are just what they appear to be. The Integrator System buttons select which Integrator server the subscription will go to and will internally translate to the appropriate queue manager names. The Queue Manager and Queue fields refer to the subscribing queue that is to receive the published messages. The Options fields refer to various options you can select for your subscription.

Table 5 shows the property settings for the subscribe application.

Code discussion

The module *CMQB.BAS* is included ‘as-is’ from IBM and won’t be discussed here for the sake of space.

Subroutine *cmdOK_Click* is where the subscribing takes place.

1 Declare all my data structures and variables.

2 Use the initialization routines supplied with *CMQB.BAS* to initialize my structures.

3 Build my PSC command XML string. Note that much of my XML manipulation logic is hidden in the BuildXML subroutine.

4 Transfer the finished XML from the XML object to the strCommand string variable.

5 Set my RFH2 header fields and convert the RFH2 header from a user-defined data type to a string. I used the **CopyMemory API** command to copy the header to an array of ASCII byte values. Then I converted each byte into a character and built up the string. Finally I combined this RFH2 string, the PSC XML command string, and any necessary padding to accommodate word boundaries into a single buffer string.

6 Set up MQSeries and send the subscribe request:

- Connect to the user-specified MQSI queue manager.
- Open queue SYSTEM.BROKER.CONTROL.QUEUE.
- Set up the necessary MQMD fields. For the ReplyToQ field, I chose to use a static queue created for the purpose of replying to subscribe requests. You will either have to create a queue with this name or change the code to match one of your queues.
- Put onto the queue the message we so lovingly crafted.

7 Read and interpret the broker's response.

- Open the queue that you specified as the ReplyToQ for input.
- Set the MQMD and MQGMO fields to wait up to ten seconds for the reply and to get only a message whose correlation-ID matches the message ID of the request message you just sent
- Get the reply message. This message will contain an

RFH2 header followed by a PSCR XML string containing status information.

- Strip the RFH2 header from the message data and parse the remaining XML.
- Extract the text value of the Completion element. If it is OK, then you're done.
- Extract the text value of the Reason element. It represents the numeric reason code of the error.
- I created a function called ReasonName(), which looks up the error message based on the reason code.
- Display the appropriate status message. If not OK, include the reason code and reason name in the message.

Mills Perry, IT Consultant/Instructor
ZyQuest (USA)

Getting started with SSL support in WebSphere MQ for Windows

In this article we focus on how to set up a simple test environment for the SSL functionality in WebSphere MQ (WSMQ) V5.3 on the Microsoft Windows platforms.

WHY FOCUS ON THE WINDOWS PLATFORMS?

All WSMQ V5.3 platforms provide SSL support over their channels. An important aspect of using SSL involves manipulating the certificates and keys that are central to the SSL protocol. Some of the operating systems supported by this version of the WSMQ product include support for management of these keys and certificates and, in general, WSMQ has used these tools where they exist.

On the Windows platforms WSMQ's key and certificate management is used in combination with that provided by the operating system. The ways in which the two components work and how they interoperate are not simple and those new to the area are likely to benefit from the detailed usage example provided here.

WSMQ SSL SUPPORT ON WINDOWS PLATFORMS

The V5.3 Gold code released in June 2002 supports SSL channels on WSMQ clients and queue managers running on Windows NT and on Windows 2000 platforms. There is no support for SSL channels on the Windows 98 platform, which is otherwise a supported V5.3 client platform; also, there is no plan to include such support in the future. Windows XP is not a supported V5.3 platform on the June release.

ENCRYPTION STRENGTH

There are two encryption strengths for which the Windows platforms can be configured: 56 bit and 128 bit. You will probably want the option to use high-strength encryption, at least on some of your channels. If your system only has 56-bit encryption support you cannot do this. The encryption level is handled differently between Windows 2000 and Windows NT platforms.

On the Windows 2000 platform you can find out the current encryption level for SSL support by going to *hkey_local_machine\system\currentcontrolset\control\securityproviders\schannel\ciphers* in the registry. If some of the ciphers listed have the number 128 by them you have 128-bit encryption. If you need to upgrade the encryption level on the Windows 2000 platform, download the Windows 2000 High Encryption Pack from Microsoft. Note that on the Windows 2000 platform Internet Explorer can be at a different encryption level from the base operating system; WSMQ uses the base operating system to provide support.

On Windows NT you can find out your current encryption level by selecting the Help button on Internet Explorer and then selecting About Internet Explorer in the pull-down. The resulting display contains a Cipher Strength field. If you find you have 56-bit encryption you should upgrade Internet Explorer to V6 or greater (alternatively, if you are on Internet Explorer V5.5 you can apply Internet Explorer 5.5 Service Pack 2). These upgrades can be downloaded from Microsoft.

CHANGING THE CHANNEL DEFINITION

We assume for the purposes of this article that you already have a working single TCP/IP non-SSL sender/receiver channel from one Windows queue manager to another on a single Windows system. We also assume that you have an appropriate remote queue definition on the sender end to allow you to use **amqsput** to successfully put a message to a local queue on the receiving queue manager.

We refer here to the receiving queue manager (the SSL server) as QM2 and the sending queue manager (the SSL client) as QM1.

It is very simple to enable SSL on this existing channel definition:

- Using WSMQ Explorer, right-click on the sender channel definition. Click on Properties. Click on the SSL tab.
- Click on the pull-down for CipherSpec (or Cipher Specification): Standard Settings. Select a CipherSpec (in this case let's use RC4_MD5_US).
- Repeat this sequence exactly for the matching receiver channel. But, additionally, for the receiver channel only, clear the tick in the 'Always authenticate parties initiating connections to this channel definition' field.

  This implies that the SSL client (the sender end of the channel) does not have to provide a certificate. We do this in this example so we can simplify the certificate setup.

We now have the channel definitions the way we want them, but if we attempt to start the channel it fails because we haven't yet provided the keys and certificates that are required.

KEYS AND CERTIFICATES

We have set up our channel so that it makes use of server authentication only. It requires:

- A personal certificate (which contains a private key) in the key repository at the SSL server end (QM2, the end with the receiver channel definition).
- A root CA certificate at the client end (QM1, the end with the sender channel definition).
- Possibly, intermediate certificates at the client end, which form a full, valid chain from the CA that signed the personal certificate to the root CA certificate.

The certificates described above are used to authenticate the certificate that is received on the SSL client from the SSL server.

OBTAINING THE CERTIFICATES NEEDED FOR TESTING

There are several ways to obtain test certificates to work with; for example, you can:

- Use the MakeCert tool, available as part of the Microsoft SDK.
- Create an internal Microsoft Certification Authority of your own and generate test certificates from there.
- Use IBM's iKeyman on the Unix platforms (or on the Windows platforms if you have or download a version of it) to generate self-signed certificates.
- Go to an external certification authority and request a free trial certificate. Examples of Web sites you might try are *www.globalsign.com*, *www.thawte.com,* and *www.digsigtrust.com*.

I give precise instructions here for obtaining a free trial certificate and the associated CA certificates from Digital SignatureTrust (DST, *digsigtrust.com*). The process for obtaining a trial certificate from DST is relatively simple and the certificates provided have a relatively long currency.

1 Go to Web site: *www.digsigtrust.com* (using Internet Explorer).

2 Click on Products/Services.

3 Click on Get a TrustID Demo Certificate.

4 Fill in the identification form and click Continue.

5 Check your form contents and click Continue.

6 You are asked to select from the options below a mechanism for storing your TrustID digital certificate. You should select browser (not roaming) and click on Continue.

7 Click Accept on the certificate agreement.

8 Next panel: leave 1024 as the Key Bit Length and select Microsoft Enhanced Cryptographic Provider. Click Continue.

9 Next panel: you don't have to do anything (you will already

have the DST root certificate, the root of the CA certificate chain, so you don't need to download this).

10 You will receive a URL in your e-mail with an activation code.

11 Access the URL through Internet Explorer.

12 It will give you the activation code; just type in the passcode you gave on the identification form at step 4 above. Click Retrieve.

13 You now have your personal certificate; click Continue.

14 Go into Internet Explorer, Select Tools->Internet Options->Content->Certificates.

15 Select the Personal Tab and you will see a certificate with your name in the Issued To column and DEMO CA A6 in the Issued By column.

16 Double-click (left mouse button) on that certificate. This brings up a display of certificate information.

17 Click on the Certification Path tab. This shows you the chain of CA certificates that you need in order to validate this personal certificate. In the Certificate status box at the bottom of this panel the status is This certificate is OK, which indicates that the CA certificate chain is complete and valid.

You now have, and can see that you have, the certificates you need.

You next want to add your personal certificate to the queue manager's store and assign it to the queue manager so it is the certificate the queue manager uses.

ADDING THE PERSONAL CERTIFICATE ON THE SSL SERVER

The simplest way to do this is detailed below.

1 Go into WSMQ Explorer.

2 Expand the Queue Managers folder.

3 If it is not currently running, start the SSL server queue manager (QM2) to which you wish to assign the personal certificate.

4 Right-click the QM2 queue manager; select Properties.

5 Click on the SSL tab.

5 Click on the Manage SSL Certificates button.

6 Click Add... on the Manage SSL Certificates panel.

7 Access the Certificate Store pull-down; select MY(Current User).

8 Select the certificate in your name issued by DEMO CA A6 that you just received from DST. You will notice that it has a picture of a key on its icon, which indicates that it has a private key associated.

9 Click on the Add button.

If this works, you now have the personal certificate in your queue manager store.

Unfortunately, if you are adding a certificate for the first time and you are running a WSMQ version at a low service level and you are running on Windows NT or on a low service level of Windows 2000, this Add will fail with the message AMQ9680 'a problem was encountered with the specified certificate file'.

If you encounter this problem you have to use the **amqmcert** command line tool to add in the personal certificate you have obtained. This involves:

1 Listing the certificates in the Microsoft MY store: **amqmcert -k MY -l**.

2 Adding the personal certificate you have obtained into your queue manager store, eg *amqmcert -a 14002 -m QM2*, where 14002 is the handle for the certificate in the amqmcert list and QM2 is the queue manager name for the SSL server queue manager.

Note that in all environments you can add further certificates using the GUI once you have added this one using **amqmcert**.

We have now added the certificate but we still need to assign it as the one our server queue manager will use.

ASSIGNING THE CERTIFICATE TO THE QUEUE MANAGER

This is achieved quite simply as follows:

1 If you are not already on the Manage SSL Certificates panel for your SSL server queue manager, follow the previous instructions to get there.

2 Click Assign.

3 You move on to the Assign Queue Manager Certificate panel.

3 Select your new personal certificate.

4 Click Assign.

5 You return to the Manage SSL Certificates panel with confirmation that the Assigned Certificate is the one you chose.

So we have the personal certificate in the SSL server queue manager store and it is assigned to that queue manager. All we need do now is to add the CA certificates to the SSL client queue manager store so that the SSL client can authenticate this personal certificate.

ADDING THE CA CERTIFICATES ON THE SSL CLIENT

Fortunately, in this scenario both queue managers are running on the same machine so we already have the CA certificates in the Microsoft system stores on the machine on which the SSL client queue manager runs. (If the queue managers were running on different machines you could export and ftp the CA certificates, or get them again in a second request to DST.)

1 Use the WSMQ Explorer to access the Manage SSL Certificates panel on QM1, the SSL client queue manager.

2 Click Add...
3 Access the Certificate Store pull-down; select ROOT.
4 Select the DST RootCA X1 certificate.
5 Click on the Add button.
6 You are back on the Manage SSL Certificates panel; click on Add... again.
7 Access the Certificate Store pull-down; select Certification Authorities (CA).
8 Select the DST Root CA X3 certificate.
9 Click on the Add button.
10 You are back on the Manage SSL Certificates panel; click on Add... again.
11 Access the Certificate Store pull-down; select Certification Authorities (CA) again.
12 Select the DEMO CA A6 certificate.
13 Click on the Add button.

You have now set up all the certificates you need.

RUN YOUR SSL CHANNEL

Start the channel as normal for a non-SSL channel. It starts up with SSL authentication and key exchange using the DST certificates. Because of this extra processing it takes somewhat longer to start than it did when it was not an SSL channel. You may need to click on the refresh icon to update the channel display to running. Once it has started, use **amqsput** to transfer data over it. Data is transferred using 128-bit RC4 encryption and MD5 hashing.

You now have a working SSL channel.

Mike Horan
WebSphere MQ Base Development, IBM Hursley (UK)

CSQ4BVJ1

CSQ4BVJ1 is a utility on z/OS that enables a Get on a queue object. Messages are printed to DD sysprint.

CSQ4BVJ1

```
//userid     jobcard,MSGCLASS=X
//           CLASS=S,
//           NOTIFY=userid,USER=userid
//*  CUSTOMIZE THIS JCL HERE FOR YOUR INSTALLATION
//*  YOU MUST DO GLOBAL CHANGES ON THESE PARAMETERS USING YOUR EDITOR
//*  THIS JOB RUNS THE GET SAMPLE PROGRAMS, CSQ4BVK1/CSQ4BVJ1
//*       Replace
//*       SYS3.MQSERIES.TGT      -  The high level qualifier of the
                                    MQSeries target library data sets.
//*       asl1.templib           -  User dataset qualifiers for CSQ4BVx1
                                    load module
//*       ++HLQ.USERCOB2++       -  User system dataset qualifiers for
VS
                                    COBOL II library
//*    PROGRAM CSQ4BVJ1 ISSUES MQGET ON A QUEUE.
//*     (EXEC and PARM statements are commented out as shipped)
//*     - FIRST PARM   (QMC)     QUEUE MANAGER NAME
//*     - SECOND PARM  (QMC.TESTING.DEBLOCK.QUEUE)  QUEUE NAME
//*     - THIRD PARM   (3Ø)      THE NUMBER OF MESSAGES TO GET-(9999)
//*     - FOURTH PARM  (B)       GET TYPE-(B)ROWSE/(D)ESTRUCTIVE
//*     - FIFTH PARM   (S)       (S)YNCPOINT/(N)O SYNCPOINT
//GETMSGS  EXEC  PGM=CSQ4BVJ1,REGION=1Ø24K,
//  PARM=('QMGRname,SYSTEM.ADMIN.CHANNEL.EVENT,ØØ2Ø,D,S')
//STEPLIB  DD   DSN=ASL1.TEMPLIB,DISP=SHR
//         DD   DSN=SYS1.SCSQAUTH,DISP=SHR
//         DD   DSN=SYS1.SCSQLOAD,DISP=SHR
//SYSDBOUT DD   SYSOUT=*
//SYSABOUT DD   SYSOUT=*
//SYSPRINT DD   SYSOUT=*
//SYSOUT   DD   SYSOUT=*
//*        DD   DSN=++HLQ.USERCOB2++.COB2LIB,DISP=SHR
```

Saida Davies
IBM (UK)

Using queue manager alias

HOW QUEUE MANAGER ALIAS WORKS

When a queue is opened using MQOPEN/MQPUT1, the queue name resolution takes place; this determines the destination queue name and queue manager (qmgr) name and the xmitq to be used. If the message needs to be placed on the xmitq because the message is for the remote queue manager, MQXQH (the transmission queue header) is prefixed with MQMD, followed by the message.

On the local queue manager, if there is a queue manager alias definition with the same name as the destination queue manager of the message, the fields from qmgr alias are copied to MQXQH fields and the xmitq specified (if any) are used.

When the receiving channel receives the message, the information in the remote queue manager field of MQXQH is checked. If there is a queue manager alias definition with the same name as the remote qmgr field of the MQXQH structure, the values from the rqmname field from the qmgr alias are copied to the remote qmgr field of the MQXQH.

If the remote qmgr field of MQXQH is the same as the local queue manager to which the receiving channel is connected, the message is put to the remote queue name present in the MQXQH header. If the queue is not present, the message is put to the dead letter queue (if specified).

If the remote qmgr field in MQXQH is not the same as the local queue manager to which the receiving channel is connected, the messages are placed on the xmitq. From here the messages are transferred to the destination queue manager using appropriate channels.

WHEN AND HOW TO USE QUEUE MANAGER ALIAS

Let’s look at four different scenarios:

- Without queue manager alias.

Using queue manager alias to:

- Redirect messages by specifying the xmit queue name.
- Redirect messages by changing the qmgr of the incoming message so that it is the same as the local qmgr.
- Redirect messages by remapping the qmgr name of outgoing messages.

Note that if the remote queue definition exists for a particular queue, the queue manager alias definition will be ignored.

Scenario one: without queue manager alias

In Figure 1 messages from QM1 reach Q2 on QM2. The following definitions are required at MQ1 and QM2 to enable this setup (assuming that queue managers QM1 and QM2 and the requisite listeners are running).

- QM1.mqsc:
  - def ql(QM2) usage (xmitq)
  - define chl(TO.QM2) chltype(sdr) trptype(tcp) conname('QM2_IPaddr(port)') xmitq(QM2).
- QM2.mqsc:

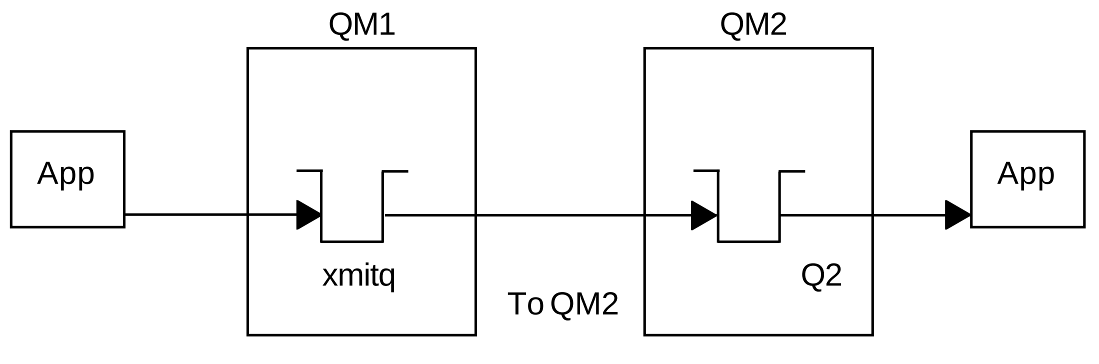

Figure 1: Without queue manager alias

- def ql(Q2)
- def chl(TO.QM2) chltype(rcvr) trptype(tcp).

(Note that remote queue definition is not required here. The application needs to specify both the object name and object queue manager name fields of the MQOD structure before issuing an MQOPEN call. If the object queue manager name is left blank and you are using the remote queue definition, queue manager alias will not work.)

- Put application connecting to QM1:
  - OD.ObjectName = Q2
  - OD.ObjectQmgrName = QM2.

Table 1 shows the MQXQH fields, ie the destination queue name and destination queue manager name.

Scenario two: redirecting the messages by changing the xmit queue

In the first scenario, as depicted in Figure 1, if channel TO.QM2 terminates (because of network or other problems) and the messages need to reach their destination because they are mission-critical, we need to find an alternative way to ensure that messages reach QM2, and without modifications to any applications.

One way to achieve this is to use queue manager alias, as shown in Figure 2. Queue manager aliases are defined as remote queue definitions with a blank rname. Let us assume in this second scenario that there is another queue manager – QMTRANS – and we can set up a channel between QM1 and QMTRANS and

Qmgr name	QM1	QM2
Object name	Q2	Q2
Object queue manager name	QM2	QM2

Table 1: MQXQH details on QM1 and QM2

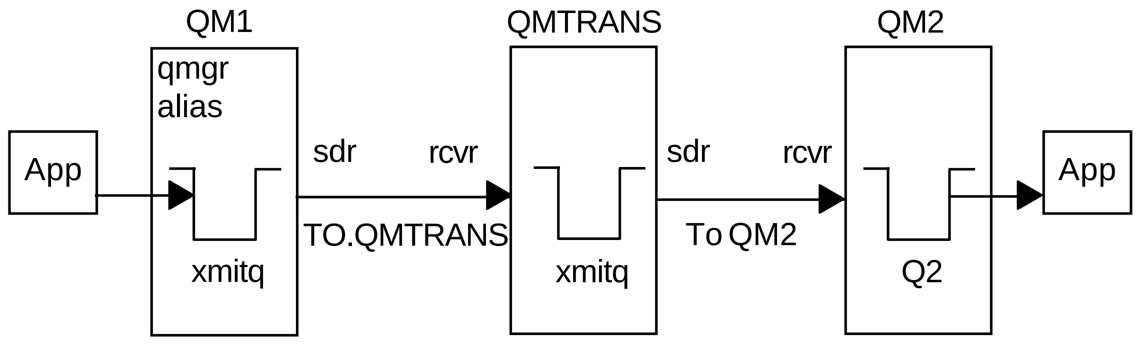

Figure 2: Using queue manager alias to redirect messages

between QMTRANS and QM2.

Since the direct communication between QM1 and QM2 is not working we want the messages to reach QM2 through QMTRANS. This implies that QM1 needs to send the messages to QMTRANS and QMTRANS should in turn forward the messages to QM2.

To achieve this we need the following definitions at QM1, QMTRANS, and QM2.

- QM1.mqsc:
  - def ql(QM2) usage (xmitq)
  - define chl(TO.QM2) chltype(sdr) trptype(tcp) conname('QM2_IPaddr(port)') xmitq(QM2)
  - del ql(QM2)
  - def qr(QM2) rname(' ') rqmname(QM2) xmitq(QMTRANS)
  - def ql(QMTRANS) usage(xmitq)
  - def chl(TO.QMTRANS) chltype(sdr) trptype(tcp) conname('QMTRANS_IPaddr(port)) xmitq(QMTRANS).
- QMTRANS.mqsc:
  - def ql(QM2) usage(xmitq)
  - def chl(TO.QM2) chltype(sdr) trptype(tcp) conname('QM2_IPaddr(port)) xmitq(QM2)

  - def chl(TO.QMTRANS) chltype(rcvr) trptype(tcp).
- QM2.mqsc:
  - def ql(Q2)
  - def chl(TO.QM2) chltype(rcvr) trptype(tcp).

In QM1.mqsc "Def rq(QM2) rname(' ') rqmname(QM2) xmitq(QMTRANS)" is the queue manager alias definition. This definition will enable the queue manager QM1 to put the messages for QM2 to xmitq QMTRANS. The channel TO.QMTRANS will transfer this message to QMTRANS, QMTRANS qmgr will put this message to xmitq QM2, and from there the message will be transferred to the queue Q2 on QM2. Table 2 shows the MQXQH details for this second scenario.

If the network problem is corrected and the channel between QM1 and QM2 is up, queue manager alias on QM1 can be deleted to restore the original setup where the messages will go directly from QM1 to QM2.

Scenario three: changing the incoming message qmgr so that it is the same as the local qmgr

Figure 2 shows how messages from QM1 reach QM2 through QMTRANS. If QM2 is down for some reason and you want the messages to reach the queue on QMTRANS, Figure 3 illustrates how to achieve this, using queue manager alias and the definitions as detailed below.

- QM1.mqsc:
  - def ql(QM2) usage (xmitq)

Qmgr name	**QM1**	**QMTRANS**	**QM2**
Object name	Q2	Q2	Q2
Object queue manager name	QM2	QM2	QM2

Table 2: MQXQH details for scenario two

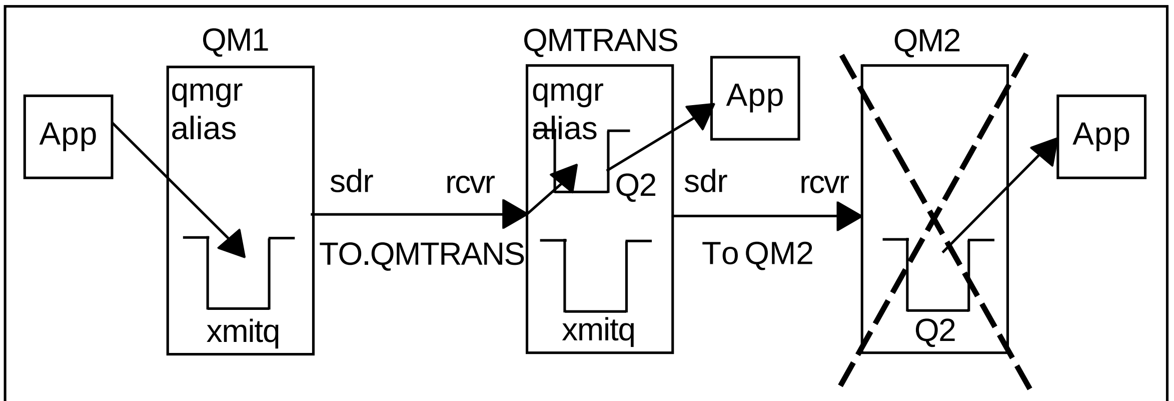

Figure 3: Changing the incoming message qmgr

  - define chl(TO.QM2) chltype(sdr) trptype(tcp) conname('QM2_IPaddr(port)') xmitq(QM2)
  - del ql(QM2)
  - def qr(QM2) rname(' ') rqmname(QM2) xmitq(QMTRANS)
  - def ql(QMTRANS) usage(xmitq)
  - def chl(TO.QMTRANS) chltype(sdr) trptype(tcp) conname('QMTRANS_IPaddr(port)) xmitq(QMTRANS).
- QMTRANS.mqsc:
  - def ql(QM2) usage(xmitq)
  - def chl(TO.QM2) chltype(sdr) trptype(tcp) conname('QM2_IPaddr(port)) xmitq(QM2)
  - def chl(TO.QMTRANS) chltype(rcvr) trptype(tcp)
  - del ql(QM2)
  - def qr(QM2) rname(' ') rqmname('QMTRANS)
  - def ql(Q2).

The queue manager alias definition on QMTRANS will change the destination queue manager name in the xmitq header to QMTRANS, which is the same as the queue manager to which the receiving channel is connected. This will make the receiving channel put the messages to Q2 on QMTRANS. The MQXQH details are shown in Table 3.

Qmgr name	**QM1**	**QMTRANS**	**QM2**
Object name	Q2	Q2	–
Object queue manager name	QM2	QMTRANS	–

Table 3: MQXQH details for scenario three

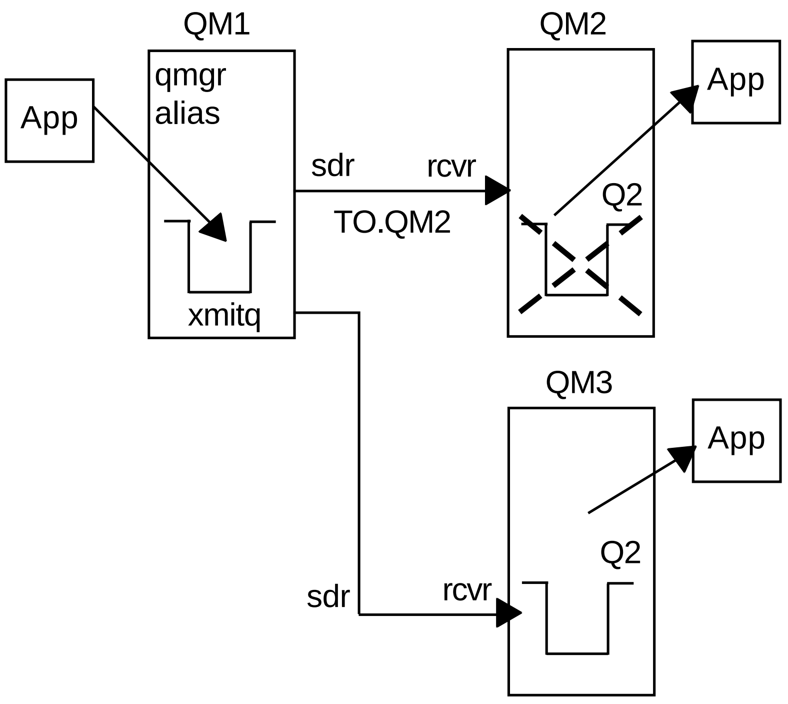

Figure 4: Remapping the qmgr name of outgoing messages

Scenario four: remapping the qmgr name of outgoing messages

In the first scenario, if messages cannot be put to the queue on QM2 (perhaps because the queue is full), we may need to redirect messages to another queue manager. Again, this can be achieved with queue manager aliases. For example, as Figure 4 illustrates, we can redirect the outgoing messages at QM1 to QM3.

The following definitions are required:

- QM1.mqsc:
  - def ql(QM2) usage (xmitq)

- define chl(TO.QM2) chltype(sdr) trptype(tcp) conname('QM2_IPaddr(port)') xmitq(QM2).
- def qr(QM2) rname(' ') rqmname(QM3)
- def ql(QM3) usage(xmitq)
- def chl(TO.QM3) chltype(sdr) trptype(tcp)

Qmgr name	**QM1**	**QM2**	**QM3**
Object name	Q2	–	Q2
Object queue manager name	QM3	–	QM3

Table 4: MQXQH details for scenario four

  conname('QM3m/cname(port)') xmitq('QM3).

- QM3.mqsc:
  - def ql(Q2)
  - def chl(TO.QM3) chltype(rcvr) trptype(tcp).

The MQXQH details are shown in Table 4.

Geetha Adinarayan, Software Engineer
IBM (India)

The MQLSX is dead: long live MQJava for Notes

MQSeries is, of course, an excellent infrastructure tool that enables simple, industrial-strength, guaranteed message delivery between any combination of the AIX, iSeries, HP-UX, Linux, Sun Solaris, z/OS and OS/390, and Windows platforms. The LotusScript

Extensions for MQSeries (MQLSX) were provided as a useful and free download from IBM’s SupportPac MA7D Web site *(http://www-3.ibm.com/software/ts/mqseries/txppacs/ma7d.html)* and extended the power of MQSeries to Notes agents.

In fact MQLSX was so useful that I was able to create a number of production Notes applications using it, details of which were published in several journals. These applications merged the ease of use of the Notes and PC platforms with the raw power and huge databases available on the mainframe.

However, IBM has pulled the plug on MQLSX and it is no longer downloadable. IBM’s official statement from the MA7D Web site is “The function provided in this SupportPac was withdrawn on 31 December 2001. If you are considering developing applications involving Lotus Notes/WebSphere MQ interaction it is recommended that you use MQSeries classes for Java provided in SupportPac MA88 (*http://www-3.ibm.com/software/ts/mqseries/txppacs/ma88.html*). Any service updates for the SupportPac will be made available at the WebSphere MQ Support site (*http://www-3.ibm.com/software/ts/mqseries/support/summary*).”

Apparently, IBM figures Java will be good for the longer term, while the LotusScript Extensions are no longer strategic. How disappointing. I put a lot of time and effort into Notes development with the MQLSX.

Somewhat saddened, I began to convert our Notes MQLSX agents to Notes Java agents. I quickly ran into a problem because there were no sample programs that did what I needed. I found a sample MQSeries Java application but it contained no Notes coding. I also found a sample Notes Java agent with no MQSeries calls.

This article is a description of how to combine all three – Notes, Java, and MQSeries – with a list of the environment issues that must be addressed and a sample demonstration program. You can use this sample to get yourself started on your conversion from MQLSX to MQJava or as a tool to learn more about MQSeries and Java.

First, download the MQ Java classes from IBM SupportPac MA88, *The MQSeries Classes for Java and MQSeries Classes for Java Message Service*. The Web site address is *http://www-4.ibm.com/software/ts/mqseries/txppacs/ma88.html*.

Release 1.1.4 was posted here on 5 April 2002 and it will be supported until 31 December 2003.

You must install Java – either the JDK or JRE – on any Notes server that will use the MQ Java classes and on any developer's PC. The MQ Java classes require JDK 1.3.0 or higher. I installed JDK 1.3.1 onto a test server.

Here's how to install the MA88 MQ Java classes on a Windows NT server:

- Unzip MA88 into a temporary directory. Keep the name and directory structure intact.
- Run setup.exe. The classes will be installed into C:\Program Files\IBM\MQSeries\Java. I could not see a way to select a different directory.
- Update the CLASSPATH on the target PC to add the following:

  ```
  C:\Program Files\IBM\MQSeries\Java\lib\com.ibm.mq.jar;
  C:\Program Files\IBM\MQSeries\Java\lib\com.ibm.mqiiop.jar;
  C:\Program Files\IBM\MQSeries\Java\lib\;
  C:\Program Files\IBM\MQSeries\Java\samples\base\;
  .;
  d:\notes\data\domino\java\NCSO.jar;
  d:\notes\Notes.jar
  ```

  Make sure the 'dot' entry is in the CLASSPATH ahead of the directory containing Notes.jar.
- Update the PATH environment variable on the target PC to add the following:

  ```
  installdir\lib;
  installdir\bin
  ```

- If you are planning to use the Java Message Service, JMS, set the NT environment variable MQ_JAVA_INSTALL_PATH to the directory where JMS is installed. JMS is not needed to connect MQSeries with Notes using Java so it plays no further part in this article.

- Update NOTES.INI to include the following:

```
ALLOW_NOTES_PACKAGE_APPLETS=1
JavaUserClasses=c:\program
files\IBM\MQSeries\Java\lib\com.ibm.mq.jar;c:\program
files\IBM\MQSeries\Java\Lib
```

JavaUserClasses is Domino's version of ClassPath for non-Domino classes and is both important and poorly documented. The second '\java\lib' is needed to avoid getting the error message 'Unable to load message catalog – mqji' each time you run an MQ Java agent.

The Notes Java Programmer's Guide (*javapg.nsf*) is a great source of ideas, references, samples, etc. Also, when you install MA88 you get some sample Java source code, which is very useful. You can get another excellent source of information, the *MQSeries Using Java* reference manual, SC34-5456, from the MA88 download Web page.

If you have a Notes agent that uses the MQLSX and you want to convert it to MQJava, you will need to go to your MQ queue manager server computer and change the MQServerDEFS.TXT MQ definitions to alter the NOTES.AGENT.PROCESS to trigger your 'New Java Agent' instead of the old 'MQLSX Notes Agent'. Then issue the **runmqsc** command to reset your MQSeries definitions.

What about the trigger monitor? The MA7K Trigger Monitor for Notes Agents is another key part of this puzzle. I wondered if it would also be going away sometime soon so I e-mailed the author at IBM to find out his thoughts. He said he was not aware of any move afoot to drop MA7K; it was a separate entity from the MQLSX.

The MA7K Trigger Monitor does not care what kind of Notes agent it is triggering. In fact MA7K works really well and triggers both MQLSX and MQJava Notes agents.

The MQJava agent given here is a sample and a proof of concept demonstration application that should give you a base from which you can build your own production Notes MQJava agents.

The sample code demonstrates how to start a Notes session within a Java agent. It then connects to an MQSeries queue manager and writes data, a message, onto a queue. It then reads data back from that same queue. Then it creates a Notes document inside the current database and mails that document to a predefined address. It writes information to a Notes log database, then closes down.

You could use this sample to read all kinds of messages from an MQSeries queue and perform various Notes processes on them. Or you could use this sample to write all kinds of Notes data onto an MQSeries queue for pickup and processing elsewhere, for example to send data to the mainframe. An MQSeries infrastructure makes this kind of inter-platform communication easy to implement.

The application that inspired this work is a mainframe CICS process that writes 'To' and 'From' addressing data, plus message text, onto a queue for delivery to an agent on the Notes PC server. The Notes agent picks up the addressing information and message text and combines it with a server file, then e-mails the resulting document to the indicated recipients.

The sample also writes information to a Notes log database. It requires that you first create a *javalog.nsf* database, using the standard Notes Log template. The System.out.println() method will put data into your existing Notes log but I included the log code to show another Java option for tracking the progress of the agent.

The agent is set to run 'Manually from Action Menu', and 'Run once (@Commands may be used)'.

Finally, some comments on garbage collection. Java has built-in garbage collection, where storage occupied by objects that are no longer in use will be recovered and freed by the system. MQJava objects participate fully in this garbage collection process.

Unfortunately, Notes objects inside a Java application do not participate in garbage collection. You must call the recycle() method for Notes objects you create or the database where your agent resides will gradually grow over time. However, you don't have to recycle absolutely everything since recycling a ‘parent’ object will remove the ‘child’ object(s).

SAMPLE MQ, JAVA, AND DOMINO AGENT

```
import lotus.domino.*;              // standard Domino Java classes
import java.io.PrintWriter;         // use for debug output writing
import com.ibm.mq.*;                // use for MQ classes
// Sample program copyright 2ØØ2 by Joe Larson, proof of concept for
MQ, Java, and Notes.
// Write a message to an MQ queue, read back from that queue, then
create and send a Notes message containing the data read from the
queue.
// Log the results to a javalog.nsf database.
public class MQJDSamp extends AgentBase {
   // Set up MQ string values for later use.
  private String hostname = "MQQMGRNAME.FQDN.COM";
  private String channel = "CLIENT.MQQMGRNAME";
  private String qManager = "MQQMGRNAME";
  private MQQueueManager qMgr;
 // All Notes Java agents must start with a NotesMain() method.
 public void NotesMain() {
  try {
  //  Session is the base NotesSession object. Everything starts here.
  Session s = getSession();   // establish a session object
  //  All agent information comes from the AgentContext object.
  AgentContext agentContext = s.getAgentContext(); // Get the agent
context.
  // Pick up the current database, the one this agent resides in.
  Database db = agentContext.getCurrentDatabase();
  // Start by creating a new document in the current database.
  // Begin building the document here, then add to it as you go along.
  Document doc = db.createDocument();  // Create a new document in the
current database.
  // In this sample, we hard code the recipient.
  // Another way would be to read the recipient name from the MQ queue.
  // The recipient name must be in a format the Address Book
recognizes.
  // start creating fields, first the SendTo
  String sendto = "Joseph Larson";
  doc.replaceItemValue("SendTo", sendto);  // put field into document
  // We do not want to actually save each document we mail out from
this
  // agent,so we set the "setSaveMessageOnSend" property to "false" to
prevent this from happening.
  doc.setSaveMessageOnSend(false); // do not save after sending
  // We need to set these other properties of the sendTo field as well:
  sendTo.setNames(true);
  sendTo.setSummary(true);
  // We are creating a Memo form.
  String form = "Memo";
  doc.replaceItemValue("Form", form);
  // Start building the body field here.
  // Build onto it as you go along, so that it shows you the progress
of your agent.
  String body = "Results from MQ calls: \n";
  // Set up MQ environment variables here.
    MQEnvironment.hostname = hostname;
  MQEnvironment.channel = channel;
  // Connect to MQSeries queue manager.
    qMgr = new MQQueueManager(qManager);
  // Set up open options.
    int openOptions = MQC.MQOO_INPUT_AS_Q_DEF | MQC.MQOO_OUTPUT;
  // Specify the queue we want to open.
  MQQueue system_default_local_queue =
qMgr.accessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                      openOptions,
                      null,   // default qmgr
                      null,   // no dynamic queue
                      null);   // no alternate user id
  // Define a simple MQ message.
  MQMessage hello_world = new MQMessage();
  // The writeString() method is how you add text to a message.
  // We have hard-coded some sample text here, but
  // you could, of course, get this information from a Notes database,
  // now that you know how to connect to Notes and MQ at the same time.
  hello_world.writeString("Hello Zaphod");
  // Specify put-message-options.
  // Here we simple take the defaults.
  MQPutMessageOptions pmo = new MQPutMessageOptions();
  // Now put the message onto the queue.
  system_default_local_queue.put(hello_world, pmo);
  // Record your progress in the "body" text you are building.
  body = body + "Successful put to queue. \n";
  // Now read that message back from the queue.
  // First define a new MQ Message buffer to read into.
  MQMessage retrievedMessage = new MQMessage();
  // Set the message ID so you get the old message right back again.
  retrievedMessage.messageId = hello_world.messageId;
  // Now set the get-message-options.
  // Again we take the defaults.
  MQGetMessageOptions gmo = new MQGetMessageOptions();
  // Now get the message from the queue.
  // We will accept at most 100 characters.
  system_default_local_queue.get(retrievedMessage,
            gmo,
            100);      // max message size
  // Record your progress so far in the "body" text.
  body = body + "Successful get from queue. \n";
  // Now pull out a string of text from the message.
  // The readString() method is how you extract the string
  // contents of a message.
  // There are other read methods for other types of data.
  int msgLen = 11;   //  How long a string do I want
  String msgText = retrievedMessage.readString(msgLen);
  //  Add the retrieved text to the "body" text.
  body = body + "Retrieved message = " + msgText;
  // Close the queue, you're done with it.
  system_default_local_queue.close();
  // Disconnect from queue manager.
  // This method also syncpoints and commits any messages written or
read so far.
  qMgr.disconnect();
  //  Now finish building the soon-to-be-mailed Notes document.
  doc.replaceItemValue("Body", body);
  // Send out the document via email to those in SendTo field.
  doc.send(sendto);
  // Print out status messages.
  // First pick up the current time from Domino.
  DateTime now = s.createDateTime("Ø");
  now.setNow();
  // System.out.println() write information the Notes server log.
  System.out.println("Some Useful Information For The Log and the time:
" + now.getLocalTime());
  // If you want to log your own data, here's how:
  // create a Notes Log object and write data to it.
  Log log = s.createLog("Notes Java Agent");
  log.openNotesLog("", "javalog.nsf");
  log.logAction Some Useful Information For The Log And The Time: "
  + now.getLocalTime());
  log.close();
  //  Free up objects.
  //  Java does its own garbage collection. Notes does not. You must
recycle() anything you create.
  log.recycle();
  now.recycle();
  doc.recycle();
  db.recycle();
  s.recycle();
  }       // End of method's try block
  // Capture MQ errors here.
  catch (MQException ex)
  {
   System.out.println("MQ error: comp code " +
         ex.completionCode +
         " Reason code: " + ex.reasonCode);
  }
  // Java IOException error.
  // The MQ readString() method will generate Java IOException errors
```

```
when you, for example,
  //   try to read past the end of a message.
  // Not that you would ever goof up like that.
  catch (java.io.IOException ex)
  {
   System.out.println("Error occurred writing to message buffer: " +
ex);
  }
  // All other errors.
  catch (Exception e)
  {
   e.printStackTrace();
  }
}   // end of NotesMain()
}     // end of class
```

Joe Larsen (USA) © Xephon 2002

MQ news

Nastel Technologies has recently announced AutoPilot for WebSphere/Transaction Monitor, which automates the detection and correction of problems in the message flow. Used in conjunction with Nastel's AutoPilot, the Transaction Monitor module enables users to monitor, record, track, and troubleshoot WebSphere MQ messages transmitted across integrated networks.

Nastel claims that AutoPilot for WebSphere/ Transaction Monitor performs all of the tasks required for complete message flow management. It automatically determines message types and message formats and can pinpoint any message anywhere in the network.

Additionally, it provides a time stamp between sender and receiver for each message, gathers statistics about message volume and traffic through the network's channels, calculates the performance of message throughput, and stores all of this message-related information in an SQL database.

For more information contact:
Nastel Technologies, 48 South Service Road, Melville, New York 11747, USA.
Tel: (631) 761 9100.
Fax: (631) 761 9101.
Web: www.nastel.com

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

* * *

IBM has announced Version 2.0 of its WebSphere MQ Everyplace, which extends its facilities to small footprint and mobile platforms and devices through its messaging capabilities. It links directly to the WebSphere integration brokers, including MQ Event Broker and MQ Integrator Broker, for managing the flow of information routing and distributing information between applications.

The software also enables robust pervasive messaging for WebSphere Everyplace Access.

It now has native support for Pocket PC and the Java implementation supports point-to-point messaging via JMS and also extends Java programming to smaller devices that can run Java Micro Edition.

A native C implementation is available on Windows Pocket PC, including Compaq iPAQ support. Other additions include scalability and administration improvements. Additional platforms, including Windows XP, are also supported.

Separately, IBM has announced the release of WebSphere Business Integration for Financial Networks, which was previewed as WebSphere Financial Network Integrator in the October issue of *MQ Update*.

For more information contact your local IBM representative.

* * *

xephon