47

MQ update

May 2003

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

Disclaimer

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

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.

Printed in England.

Controlling WSMQ resources in a Windows environment

WebSphere MQSeries (WSMQ) historically has supplied a robust set of identical cross-platform capabilities. Nonetheless, this has not prevented IBM from offering platform-specific capabilities where warranted, eg ISPF panels on S/390 platforms, MQ-specific SMIT panels on AIX, and Microsoft Management Console (MMC) facilities on Windows. First introduced with MQSeries V5.0, MQServices and MQ Explorer represent two related and independent uses of the MMC technology.

With WSMQ V5 R2 in a Windows environment IBM has introduced **amqmdain**, a new command to control MQ resources represented as services within a Windows Snap-in domain. The **amqmdain** facility is the newest member of a set of MQ control facilities previously consisting of Programmable Command Format (PCF), MQSeries Command Interpreter (MQSC), Control Commands (eg **crtmqm** or **strmqm**), and, in a Windows environment only, MQ Explorer and MQServices.

Just what new capabilities does **amqmdain** provide and what does it offer with respect to managing MQ resources? How does it relate to the other MQ control facilities?

This article describes the scope of control and granularity with which you can control MQSeries resources in a Windows NT and 2000 environment. In the absence of understanding how all of these facilities relate to each other you can get yourself into quite a tangled mess.

During a recent client engagement with a major global financial institution based in the Boston, MA (USA) area, we examined the pros and cons of implementing **amqmdain** in a production environment. One of our goals was to take a set of Perl scripts currently used in a production Unix environment and adapt them to a Windows environment, exploiting any platform-specific capabilities available to us. Initially, we thought that we might be able to use **amqmdain** capabilities from within the Perl scripts

but we discovered that **amqmdain** did not control resources at a suitable level of granularity.

It's beneficial to view each of the various MQ resources as a Service. Our aim is to exert control at a highly granular level in order to maximize operational control over those Services. What are the MQ resources that we wish to treat as Services and over which we wish to exert such control?

- Queue Manager.
- Command server.
- Channel initiator.
- Trigger monitor.
- Channel listener.
- Dead letter handler.
- Channels (sender and receiver).

How can we control these resources? Six facilities are at our disposal:

1 Native MQ commands (MQSC or MQSeries commands).

2 Services within an MQ Snap-in group.

3 **amqmdain**.

4 *srvany.exe* utility.

5 Custom-built SCM services.

6 Perl scripts that exercise any of the five facilities listed above.

Let's now introduce each of these facilities. Please refer to Figure 1. This diagram shows a variety of mechanisms available to control MQ resources. On the left-hand side just inside the box is a list of several Windows Services. Next, note the box labelled 'MQSeries Snap-in Services'. Two Snap-in groups are represented – QM1 and QM2 – where QM1 shows greater detail about the resources within its span of control.

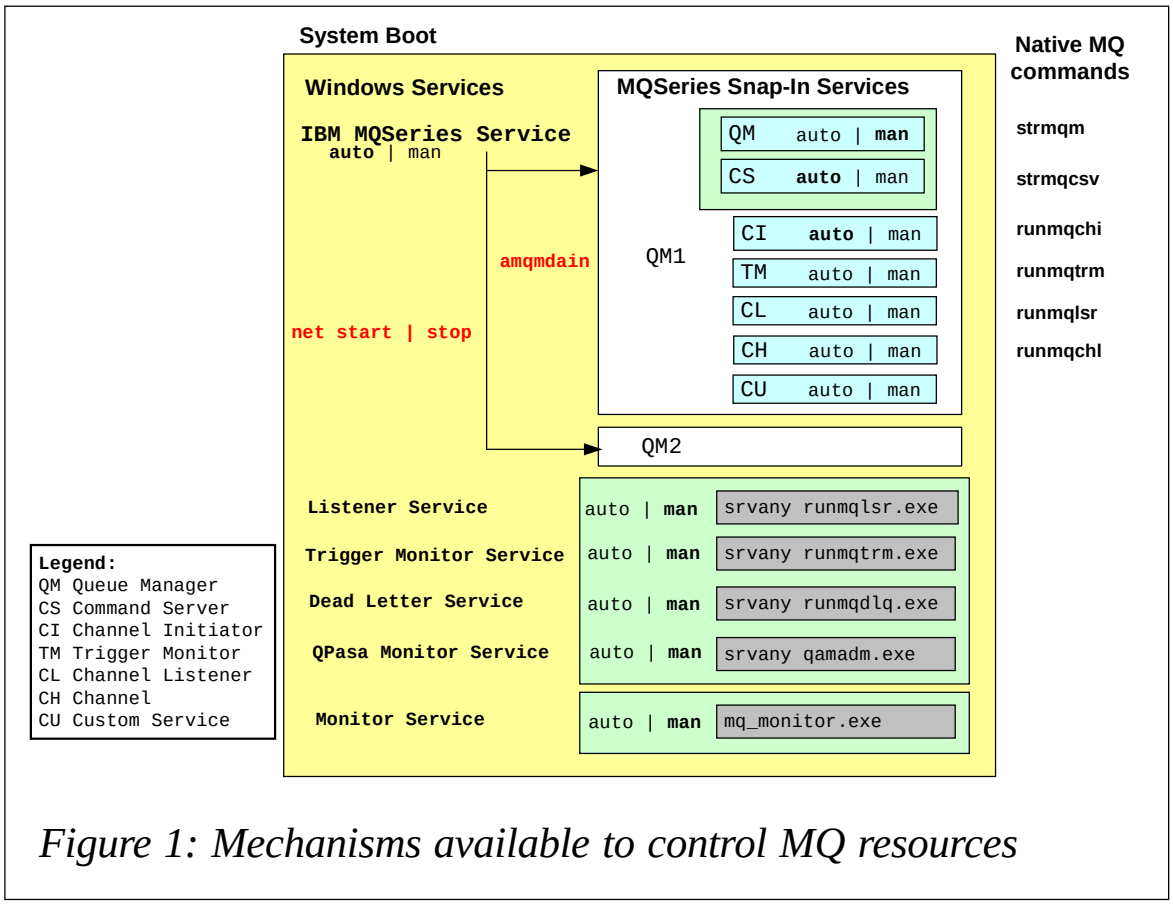

Figure 1: Mechanisms available to control MQ resources

Inside the QM1 Snap-in services box you will see various MQ resources at a granular level. And just outside the box on the right-hand side, under Native MQ commands, is a list of native MQ commands used to control the MQ resources at the most granular level. At the far right is a list of Perl scripts, a set of custom-built programs written in the Perl language. Let's discuss each of these mechanisms in greater detail.

Windows Services

A Windows Service is similar to a Unix daemon process. Windows Services can be viewed through the Services window, which you can access in one of the following ways:

- NT:
  - My Computer => Control Panel => Services.
- Windows 2000:
  - My Computer => Control Panel => Administrative Tools =>Services.

The coarsest span of control for MQ resources is at the Windows Service level. MQ provides the IBM MQSeries Service. This is like the master MQ switch. Initially, at system-boot time, the Windows operating system acts on Windows Services according to the properties associated with each Service. One such property is the auto | manual setting. When set to auto a system boot starts the IBM MQSeries Service; when set to manual the Service will not automatically start. Windows Services can be started (or stopped) using the Services Window controls or by using the **net** command.

Note that the Display Name is not the name you would use with the **net** command. If you type **net start IBM MQSeries** you'll get an error. You need the actual Service name as specified in the Windows Registry, typically MQSeriesServices. So to manually start the MQSeries Service you'll need to type **net start MQSeriesServices**.

How do you determine the precise Service name to use? Using the Registry Editor, look in the Windows Registry. Invoke the Registry Editor from a command prompt by typing in **regedit**. Once **regedit** is invoked look for the MQSeries entry in the following location:

```
HKEY_LOCALMACHINE => SYSTEM => CurrentControlSet => Services =>
MQSeriesServices
```

Another way to discover the actual Service name property (as opposed to the display name property) is by using the **sc.exe** command from a command prompt. This method can be tedious and a bit convoluted. Essentially, you would need to use the **sc query** command with the **ri=** option to scroll through the Windows Services until you find the MQSeries Service.

A key point to keep in mind is that when you manage MQ resources using the MQ Windows Service the span of control is at a global level but acting on each Snap-in group independently. As Figure 1 shows, the IBM MQSeries Service acts on both the QM1 and the QM2 Snap-in groups. So when you type **net stop MQSeriesServices** it's a global action that stops all of the resources associated with QM1 and with QM2.

Under the hood, *amqsvc.exe* and the *amqmperf.dll* are the executables associated with the IBM MQSeries Service. Looking at the Processes tab within the Windows Task Manager you'll see the *amqsvc.exe* process running when the IBM MQSeries Service is active.

You'll also observe in Figure 1 several other Windows Services listed; we'll get to those soon when we discuss the *srvany.exe* utility and the SCM.

MQ Snap-in Services

MQ provides MQSeries Explorer and MQSeries Services, two independent but related Windows programs based on MMC facilities. The Microsoft Management Console (MMC) represents a facility that enables the custom development of administration tools that operate in a Windows environment. Modules that are added to the basic MMC console are called Snap-ins. A complete MMC solution is represented as an *.msc* file.

Figure 2 shows the MQSeries Services window. Note that three queue managers are listed: QM1, QM2, and QM_H0T5Z, each of which represents an independent Snap-in group.

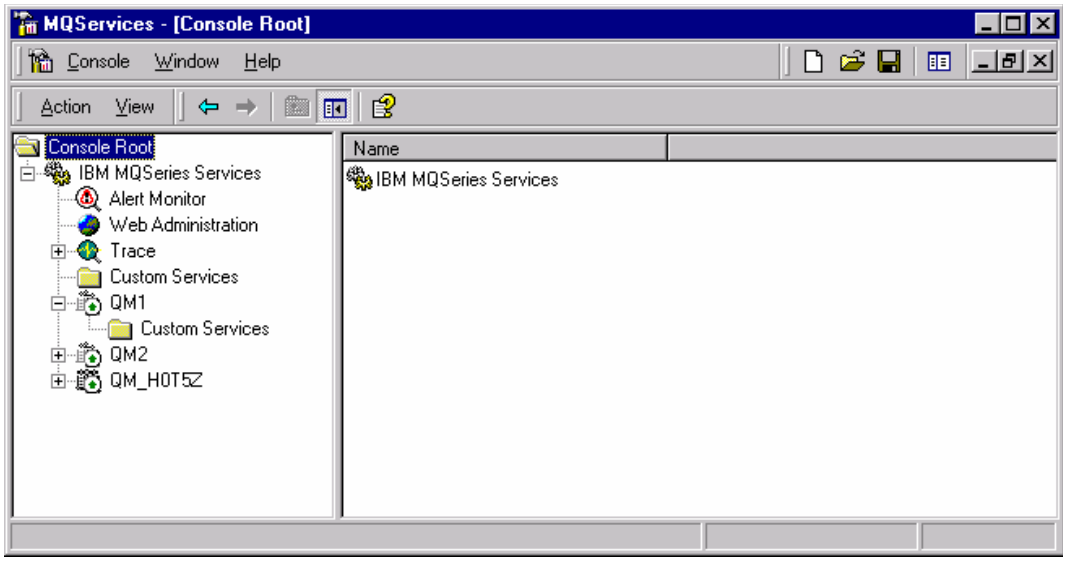

Figure 2: The WSMQ Services window

The item labelled MQ Snap-in Services shown in Figure 1 represents the set of services associated with each queue manager domain or Snap-in group.

You may have noticed a folder labelled Custom Services inside the MQServices MMC window. This is a new capability with MQ V5.2 that enables you to establish Custom Services that operate within the span of control of a Snap-in group. This facility is a replacement for the *scmmqm* utility that was used to install various MQ resources as a Windows Service.

amqmdain	**Capabilities**
Configuring services	
[auto | manual] QmgrName	Configures a previously defined queue manager within a Snap-in domain with either the automatic or manual property.
Creating services	
crtlsr QmgrName LsrParms	Creates a listener service within a Snap-in domain.
crttrm QmgrName InitQname	Creates a trigger monitor service within a Snap-in domain.
crtchi QmgrName ChInitQName	Creates a channel initiator service within a Snap-in domain.
Controlling services	
start QmgrName	Starts services within a Snap-in domain.
stop QmgrName	Stops services within a Snap-in domain.
Status	
status [QmgrName | all]	Displays the status of: • The IBM MQSeries Windows Service and *either:* • The MQ resources within the named Snap-in domain *or:* • The MQ resources within a domain for each Snap-in group.

Table 1: Resource management capabilities of ***amqmdain***

amqmdain

Essentially, **amqmdain** provides MQ resource management capabilities in four areas – configuration, creation, control, and display – as described in Table 1 below.

The table omits a couple of other capabilities that are not relevant to this discussion. The complete and official description of the **amqmdain** command can be found in the *MQSeries Release Guide for V5.2*, (document number GC34-5761-00). Descriptions for the other control facilities can be found in the *MQSeries System Administration* manual.

The key point to understand is that the **amqmdain** facility operates only on the MQ Snap-in domain level and not on any other level of granularity. Resources are controlled as a group. Thus, as shown in Figure 1, **amqmdain** operates on the QM1 Snap-in group domain of resources, the QM2 group, and so on.

One consequence of this is that you do not have the granularity of control over MQ resources that you might otherwise expect or desire. For example, the command **amqmdain start QM1** starts not only the QM resource but also the CS resource. In fact you cannot directly control the CS resource at all with **amqmdain** as you can with other facilities, such as the native MQ commands **strmqcsv** and **endmqcsv**. Further, the QM and CS resources share a box in Figure 1 because **amqmdain** always controls these resources jointly, regardless of their auto | man property setting.

In fact the **amqmdain** behaviour with respect to resources within a Snap-in group is quite puzzling. For example, suppose I have a Snap-in group with four resources defined: QM, CS, CI, and CL. If all resources are defined with the auto property and I execute the **amqmdain start** command, all four resources start automatically – as expected. On the other hand, if all resources are defined with the man property and I execute the **amqmdain start** command then the QM and CS resources start while the CI and CL resources remain inactive. In the case of the **amqmdain stop** command all resources are halted regardless of the auto |

man property. I view this puzzling behaviour as a limitation of **amqmdain**.

srvany utility

The NT Resource kit (and the Windows 2000 resource kit for that matter) supplies a utility called *srvany.exe* that enables you to run any Windows application as a Windows Service. This capability is functionally similar to the Custom Services capability described above under the MQ Snap-in Services section.

Figure 1 shows four such instances of services using *srvany.exe*:

- Listener Service (uses *runmqlsr.exe*).
- Trigger Monitor Service (uses *runmqtrm.exe*).
- Dead Letter Service (uses *runmqdlq.exe*).
- QPasa Monitor Service (uses *qamadm.exe*).

The first three Services use executables supplied with MQ, as indicated. The fourth Service – QPasa Monitor – is a third-party product from MQSoftware that uses the executable indicated.

Note that just as with any Windows Service you can control these services via the Services window or with the **net start | stop** command.

SCM Services

Yet another way to control MQ resources is with a program that exploits the SCM facilities. In Figure 1, the Monitor Service is just such an example.

The Microsoft Service Control Manager (SCM) API enables you to write a program that directly interfaces with the SCM. Such a program can install or remove itself as a Windows Service and exert programmatic control via the **sc.exe** command. This approach is also particularly useful if you want to write messages to the system event log.

MQ Resource	Native MQ Control Command	MQ Explorer Command	MQ Services Command	amqmdain	Windows Service	Perl Script
QM	strmqm | endmqm	Yes	Yes	Yes	2	mq_QM
Command server	strmqcsv | endmqcsv	No	Yes	No	2	3
Channel initiator	runmqchi | 1	No	Yes	No	2	3
Trigger monitor	runmqtrm | 1	No	Yes	No	2	mq_TM
Channel listener	runmqlsr | endmqlsr	No	Yes	No	2	mq_CL
Dead letter handler	runmqdlq | 1	No	No	No	2	mq_DLH
Channels	runmqchl | 1	Yes	Yes	No	2	3

Notes:

1 There is no **stop** or **end** command to complement the **run** command although the **endmqm** command will have the same effect.

2 The Windows Service affects all MQ resources globally.

3 A Perl script could be devised to control these resources.

Table 2: Facilities for exerting control over the MQ resources listed

SUMMARY

In this article we have reviewed a variety of available mechanisms that can be used to control MQ resources. A key issue in using any of these mechanisms is to understand the scope of control and granularity at which these mechanisms operate. The list below summarizes the methods for controlling MQ resources in descending order of narrowing span of control and greater granularity.

1 The IBM MQSeries Windows Service – the master MQ switch.

2 The MQ Snap-in domain.

3 A custom Windows service using either the *srvany.exe* utility or a program designed to exploit the SCM API.

4 A custom Perl script or a native MQ command.

Table 2 indicates the facilities for exerting control over the MQ resources listed.

One note of caution: it's possible to control a particular MQ resource through a number of different facilities. The current state of that resource is not always reflected uniformly throughout all facilities. So you might take an action on a resource using one facility and then get a false indication of its state when examining that resource through another facility.

So as a general MQ administration principle, establish an MQ management architecture that specifies what MQ resources you need to control and the degree to which you need to control each of these resources independently, and then map those requirements to the available facilities. In some cases it may be suitable to manage all of your MQ resources through the master MQ switch – the IBM MQSeries Windows Service. In other cases you may choose to use Perl scripts to exert control over MQ resources in a highly granular fashion. Or perhaps it is sufficient to manage MQ resources at a Snap-in group level.

Tom Krpata, Founder, President,
Stellar Software Corporation (USA)

WMQ Integrator Broker: a performance evaluation

INTRODUCTION

In October 2002 IBM announced WebSphere Business Integration for Financial Networks (WBI for FN). (This was originally announced as WebSphere Financial Network Integrator,

or WebSphere FNI.) Part of this announcement is the WBI for FN Base product and the Extension for SWIFTNet (WBI for FN ESN or just ESN) on z/OS. WBI for FN ESN offers access to the new SWIFT Secure IP Network (SIPN). The SWIFT FIN protocol, which forms part of ESN, allows applications to interchange financial messages between different banks via SWIFT. An existing application that is supported with the ESN FIN part is IBM MERVA (Message Entry and Routing with Interfaces to Various Applications).

Both WBI for FN and the FN ESN use WSMQ Integrator Broker and run on z/OS. The functionality is delivered as a set of message flows and a set of user-defined nodes (plug-ins). This article describes the performance evaluation carried out on the WBI for FN ESN FIN component on z/OS and the results of the evaluation.

The performance evaluation was carried out to provide information to prospective WBI for FN customers. Another reason was that WBI for FN is the first product built by our development organization to use WSMQ Integrator Broker as the underlying middleware. We wanted to understand how much throughput we could achieve in such an environment, where the bottlenecks might appear, and where we had room to improve either WBI for FN or WSMQ Integrator Broker.

STARTING POINT

WSMQ Integrator Broker is a relatively new product on z/OS and before the evaluation was begun we had no firm idea of what we could expect. There is a performance report available for WSMQ Integrator on z/OS but it covers only the basic nodes delivered by WSMQ Integrator Broker. It does not encompass a real application scenario, such as that needed for the SWIFT FIN processing. It was apparent from this performance report, however, that our product would use a lot of CPU resources.

To get an idea of what we could expect we used a capacity planning tool (CPT), which is provided as SupportPac IP03 for WSMQ Integrator Broker. As input for this tool we checked our

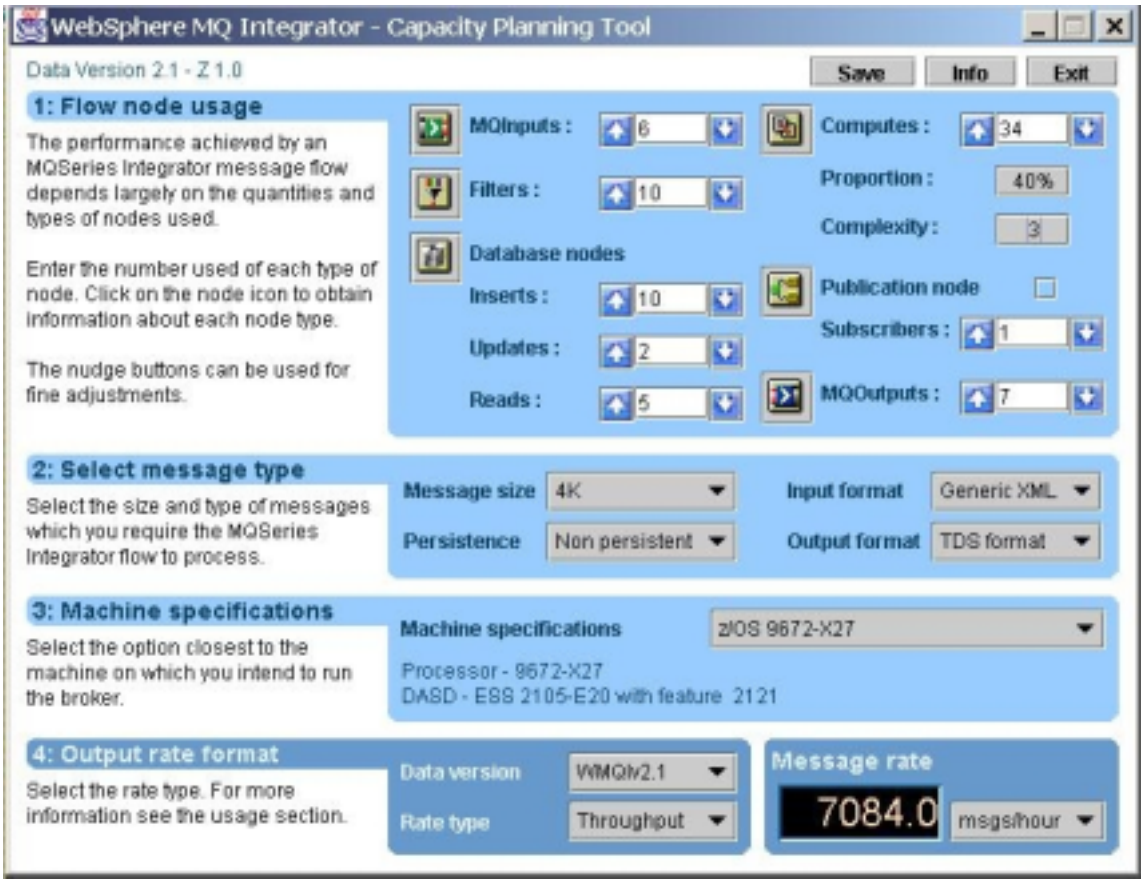

Figure 1: Results from the Capacity Planning Tool

processing and counted the nodes that are invoked for normal processing. WBI for FN has its own plug-in nodes. We translated them to be either compute or database nodes, depending on the kind of operation they perform. Other nodes, eg try-catch nodes provided with WSMQ Integrator Broker, are omitted because there is no means of inserting information about the number of such nodes into the CPT. Even without such nodes, many others are involved in the necessary processing for WBI for FN ESN FIN, which is an indication of the complexity of the processing required.

The CPT allows you to enter only an average figure for the complexity and proportion of the compute nodes. We made some assumptions when entering values for these fields. When sending SWIFT FIN messages and receiving an acknowledgment

from the SWIFT network the CPT stated that it should be possible to process 7,084 FIN messages per hour. This figure included only the WBI for FN part of sending FIN messages.

For the processing in MERVA and access to the SWIFT SIPN we expected some degradation. The results given by the CPT are shown in Figure 1. Based on the number of input nodes it can be seen that in the processing of one FIN message there are multiple WSMQ messages involved. In this case there are nine WSMQ messages for each FIN message processed.

We did a similar calculation for the scenario where the ESN receives messages from the SIPN and acknowledges them. This processing is a little more complex and the result was that WBI for FN ESN is expected to process 4,966 FIN messages per hour.

The prediction holds true for a defined hardware processor – a 9672-X27. The CPT provides no indication about CPU utilization when processing these messages nor how the result will be influenced when other programs are running in parallel. In the case of WBI for FN ESN FIN processing at least two programs are running in parallel. The first program is MERVA. We used

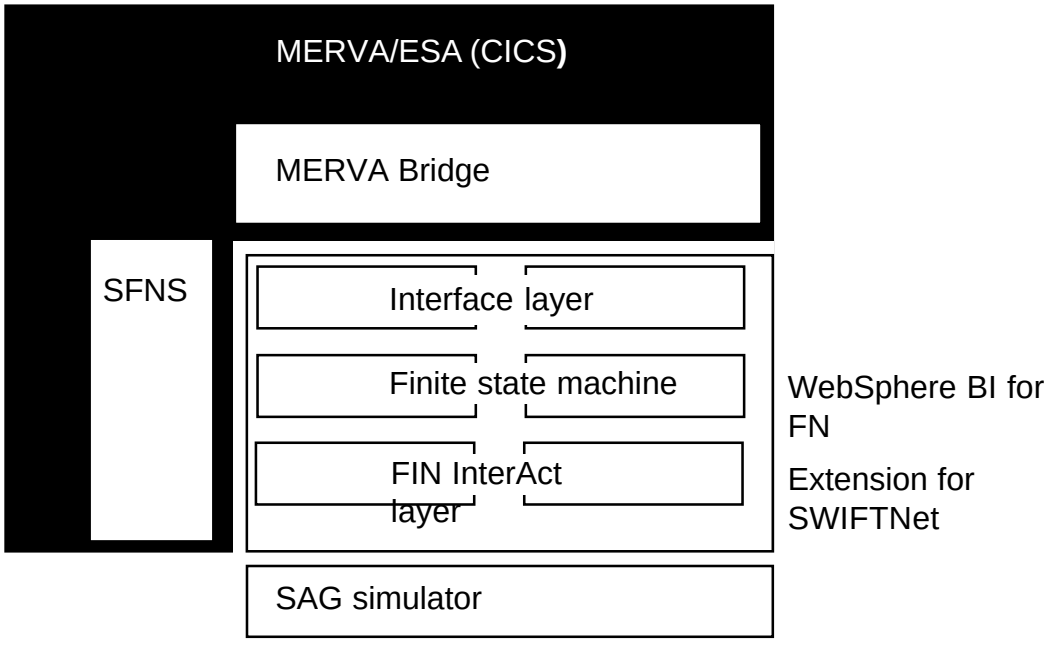

Figure 2: Test environment

MERVA/ESA running in a CICS region to provide the ESN with FIN messages and to perform some processing for FIN message validation and FIN checksum calculations.

The real access to the SIPN is performed by a component provided by SWIFT, the SWIFT Alliance Gateway (SAG). This component does not run on the same machine but as a second program it could influence the result because it accesses messages in the queues remotely and requires the WSMQ client attachment to run. So there was still a lot of uncertainty but at least we had a number to start with.

TEST ENVIRONMENT

For the performance evaluation system we used dedicated zSeries hardware; for the test we used a G7 (IBM zSeries 900 Model 216). On this system we allocated a logical partition (LPAR) with four processors and 4 GB of storage. This is a much faster machine than the one used as a reference machine for the results of the CPT. The CPT documentation told us to use the LSPR rate (IBM Large System Performance Reference, see *http://www.ibm.com/servers/eserver/zseries/lspr/*) of the hardware to scale the results from the CPT to the expected rate on the test machine. The G7 with four processors has an LSPR rate for a mixed workload of 4.29, whilst the reference machine of the CPT has a rate of 1.28. Therefore we expected to be able to process 7,084 FIN messages/hour * 4.29/1.28 = 23,742 FIN messages/hour.

The environment for the performance evaluation is illustrated in Figure 2. All processing was done on the test machine. MERVA/ESA was running in a single CICS region. This is the same MERVA that is accessed from within WBI for FN for security processing.

FIN messages are issued in MERVA and copied using a MERVA component, called MERVA Bridge, to a WSMQ queue. This message travels through the Interface Layer (IL), which accepts the message, the Finite State Machine, which implements the

FIN protocol as defined by SWIFT, and the FIN InterAct Layer to the network accessing program, the SAG. InterAct is the new protocol for the SIPN and is defined by SWIFT. Each FIN message is acknowledged and the acknowledgment travels the same way in the opposite direction until the bridge correlates the acknowledgment with the original FIN message in MERVA.

SWIFT does not provide a network where performance measurements can be carried out. The production network can't be used because this is used to process financial messages. A test network is provided for functional tests but this is not suitable for performance testing.

For these reasons we developed a simulator for the SAG and the SWIFT network. This simulator will, from a performance perspective, behave differently from the real SAG with SWIFT network but we decided that we wanted to analyse our message flows and plug-in nodes and not the SWIFT network, so this was OK for us. The simulator itself is running on a distributed platform like the SAG; in our case it's running on an AIX machine.

WBI for FN ESN message flows and nodes are deployed to one WSMQ Integrator Broker broker. This broker was at CSD3 level. WBI for FN Base was at GA level and the ESN started with an early development level and was upgraded during the evaluation phase with a development level shortly preceding the GA version.

TEST APPROACH

For the performance evaluation the main point of interest for us was the message throughput that ESN FIN processing could achieve, so we defined a configuration with specific settings. We chose to start the test with one instance for each message flow. WBI for FN ESN also has some tuning settings.

In this configuration we used a set of 15,000 FIN messages. The FIN messages were selected so that they were representative of the average length and complexity of messages found at most customer sites. When starting the test all messages were copied

as a batch into a MERVA queue. The MERVA Bridge then took the messages and put them into a WSMQ queue for processing by WBI for FN ESN message flows.

The test ended when all the messages were processed and acknowledged from the simulated SWIFT network. This is when all messages are back in a MERVA queue. To get the message throughput we measured the time from the initial copy to the end. This, in relation to the number of messages issued, results in the message throughput rate. We also measured the average utilization of the z/OS system.

INITIAL RESULTS

The first tests showed a throughput of 12,000 FIN messages per hour. At this time our message flows still had WSMQ Integrator Broker trace nodes instead of WBI for FN trace nodes. To determine where we spent the time we ran a small test with a few messages. During this test we turned on the WSMQ Integrator Broker user trace, with a level of 'normal'. This has enabled us to gain a deeper insight into which message flows take the most time and where in the message flow most time is required.

MESSAGE FLOW ANALYSIS

The first factor we looked at was the time required for each step in the message processing procedure. This is important because the processing for one FIN message is carried out in a sequence of message flows. The slowest of all these flows determines the processing speed for all messages. The message flows that have the longest elapsed processing time are the ones that we concentrated on first. Using this method the Interface Layer message flows have been identified as causing the initial bottleneck.

When looking at the CPU time for such a test run it can be seen that 60% of the processing time is spent in WSMQ Integrator Broker code, the runtime library, and WBI for FN. The rest is in CICS, MERVA, DB2, and other system components. I/O was not found to be a constraint. That's why we concentrated our

analysis on the message processing in the message flows rather than in DB2, for example.

Analysing user-defined nodes

When looking into the message processing nodes that consumed the most time we identified three different nodes, as detailed below.

- *Audit*. WBI for FN provides a node that can be used to audit processed messages. So it writes the message body and part of the WBI for FN information in a WBI for FN folder in the MQRFH2 into a database table. This node is composed only of nodes provided by WMQ Integrator Broker.
- *Configuration data provider node*. The task of the configuration provider node (CPN) is to provide dynamic configuration information into the message flow. This is done in a plug-in node provided by WebSphere BI for FN Base. This node reads the configuration information from a database table, caches the information, and inserts it into the message being processed as required.
- *Trace*. WMQ Integrator Broker already provides a trace node. Tracing is mandatory in order to provide service for products, so the WBI for FN Base also provides a node with the equivalent tracing function. The main difference is that tracing can be switched on and off dynamically.

Audit

The audit function is mandatory for processing financial messages. It is used four times during the processing of one SWIFT FIN message. Message auditing, provided by WBI for FN Base as a subflow, consists of a sequence of nodes provided by WSMQ Integrator Broker.

To investigate why these nodes consume a relatively large amount of time we used a WSMQ Integrator Broker debug trace. From such a trace it's possible to obtain information on which nodes consume what quantity of elapsed time. The first thing we

looked at was the time needed for the SQL INSERT call to insert data into the database table. We found that preparing the call takes up to 100 times more elapsed time than the actual insert. The DB2 insert takes less than 1 millisecond (ms) while the overall audit processing took up to 80 ms.

When looking at the processing time we discovered that most of the elapsed time is spent serializing parts of the MQRFH2. This is so expensive because the message tree must be completely navigated. Navigating messages using ESQL has been found to be very CPU-intensive.

When analysing this kind of problem a 'normal' WSMQ Integrator Broker trace is helpful. When looking for a better solution we found that, with CSD 3, WSMQ Integrator Broker provides a new functionality that serializes parts of a message in one ESQL call. With this change we were able to reduce the elapsed time for this node by 80%.

Configuration data provider node

Dynamic information is required in each ESN message flow. It allows WebSphere BI for FN ESN to provide generic message flows that are enriched with customer-specific information. Each message flow requires different information. This configuration information is stored in a database.

The configuration information is inserted into the message by the configuration data provider node. Since DB access is expensive in general, the node retrieves the information from the database table and caches it so that database access is not required too often. When looking at a user trace we have observed that this node takes up to 70 ms. The WSMQ Integrator Broker trace doesn't provide enough information to enable us to determine where this time is spent.

To analyse whether the time is spent in the WBI for FN code or whether WSMQ Integrator Broker requires too much time we made a comparison. We measured the time in our CPN that is needed to insert the data elements and in addition we built a

message flow that substituted the CPN with a compute node. In this compute node we hard-coded the statements to insert the same data elements in ESQL. A comparison of the results showed that inserting the elements using the WBI for FN CPN consumed far more elapsed time than when using the compute node. This shows that further investigation is required in this area.

Trace

WSMQ Integrator Broker provides a trace node that can be used during the development phase of a message flow to find functional problems. The disadvantage of using this node is that it costs much time when passed and it serializes processing when writing the trace information to a file. In addition, writing the trace information every time consumes a lot of disk space. This is why the WSMQ Integrator Broker trace nodes should not be left in messages flows when getting a message flow into production.

For a product, the trace function is mandatory in order to find problems at customer sites. This is why WBI for FN Base provides an internal node with similar functionality to that of the WSMQ Integrator Broker trace node. The main difference is that the actual tracing can be switched on or off dynamically.

For the performance evaluation we ran WSMQ Integrator Broker user traces to get the elapsed time of nodes in our message flows. We did this measurement for two WBI for FN ESN drivers, the first one with WSMQ Integrator Broker trace nodes and a second, which replaced the WSMQ Integrator Broker trace nodes with WebSphere BI for FN trace nodes. For the second driver we measured the elapsed time for the trace node when the trace node was activated and when it was deactivated. We tried to output a whole message tree (the root element) as a trace statement.

Comparing the results of the WSMQ Integrator trace node with the results for the WBI for FN trace node when tracing is enabled we observed that the WebSphere BI for FN trace node was nearly 20% slower than the WSMQ Integrator trace node. This

shows that the WSMQ Integrator Broker trace node is integrated better than user-defined nodes.

Tracing, using the WBI for FN trace node when tracing is disabled, requires less than 1 ms. This time is independent of what part of the message is traced. For the test message that is roughly 20 times faster than with tracing enabled. The test message was not that large or complex. From this reason it can be assumed that the relationship hardly depends on the actual message or part of the message that is traced.

In general, when tracing is necessary, the extra time required in the WBI for FN trace node is acceptable. It is outweighed by the fact that tracing in normal processing is disabled but can be enabled if necessary.

For problem analysis WBI for FN message flows usually needed multiple trace points. So replacing WSMQ integrator Broke trace nodes with WBI for FN trace node speeds up the processing significantly for normal message processing. The increase for the overall throughput was more than 80%.

FINAL RESULTS

Further investigations were prevented by the close of the test period. Within the performance evaluation period a fix for the audit subflow was received that uses the new WSMQ Integrator Broker functionality, which was introduced with WSMQ Integrator Broker CSD 3. As described, the WSMQ Integrator Broker trace nodes were replaced with WBI for FN trace node. This increased the initial throughput from 12,000 FIN messages per hour in steps via 25,000 FIN messages to the final number of 45,000 FIN messages per hour.

This illustrates that investigations into the message flows and into the elapsed time for message processing nodes can identify bottlenecks. Analysing where the time is spent can be a very productive means of improving performance.

In the final results the average CPU utilization was still 67%. This

showed that, on average, one of the four processors was not utilized. Running the message flows that were identified as the bottleneck, the Interface Layer in multiple execution groups, it was possible to increase the throughput, thereby having a higher CPU utilization.

To get an understanding of the scaling behaviour we ran the SWIFT FIN scenario on the same hardware with three, two, and one processor(s) respectively. The measurements showed that the product is CPU-bound and scales with an internal throughput rate that is nearly linear with the CPU capabilities. Based on this we developed a different scaling method from the one proposed in the Capacity Planning Tool. The ESN scaling recommends the use of the CBW2 LSPR rate to scale the throughput across zSeries CPU families and the number of processors to scale the throughput within one CPU family. The CBW2 workload is recommended instead of the mixed workload LSPR rate because it scales in an almost linear fashion with the number of processors.

OUTLOOK

For the configuration data provider node we have to do further analysis. The processing time for this node needs to be reduced. This is essential because at the end of the evaluation period this node required a high percentage of the elapsed time of the ESN message flows. Once this is resolved additional evaluation periods will be scheduled.

SUMMARY

Overall the performance evaluation test proved very valuable in understanding the performance behaviour of WSMQ Integrator Broker and WBI for FN. This exercise has shown that the prediction for the complex SWIFT FIN processing, based on the WSMQ Integrator Capacity Planning Tool, was very conservative. The actual throughput of 45,000 FIN messages per hour was roughly two times better than the rate calculated using the tool. The main reason could be that our application runs more parts in parallel than is assumed by the tool.

The test has also shown that WSMQ Integrator Broker-delivered nodes are normally better integrated and therefore faster than user-defined nodes. Nevertheless, any processing – especially ESQL – is very CPU-intensive. To reduce this we used new functionality from WSMQ Integrator Broker CSD3 and were able to reduce the elapsed time for one of the WBI for FN nodes by 80%. For some other processing WBI for FN provides nodes with similar functionality to WSMQ Integrator Broker. This is slower that the WSMQ Integrator Broker node but compensates through additional functionality.

Finally, the evaluation test has identified the problem areas in the WBI for FN ESN message flows and plug-in nodes. This gives us a baseline from which to improve message throughput in the future.

Michael Groetzner and Christian Herrmann, IBM (Germany)

Setting up a client to server SSL connection

This article explains the procedure required to test an MQ client to MQ server SSL connection using WSMQ V5.3 CSD01. Support for SSL connections is a new feature of V5.3 and both the client and server must be at V5.3.

In this example the client was on a Windows NT SP6 platform and the server was on a Windows 2000 platform. A different certification authority was used for the client and server to avoid confusion as to which certificates are required in the client and queue manager stores.

If both certificates are from the same authority it is easy to think –mistakenly – that the client and server need the intermediate and roots from their own certificates; this is not the case.

PROCEDURE

Obtaining a certificate

The first step is to obtain a temporary certificate from a certification authority for queue manager use. In this example the certificate was obtained from DST digitalsignaturetrust at *www.digsigtrust.com*. The Web link provides instructions for obtaining and downloading the certificate. The certificate was named SSL QRS1QMGR. The queue manager name in this test was TEST1.

Next, use Internet Explorer (IE) to determine the certification path by opening IE and selecting Tools/Internet options. Click on the Content tab and select the Certificates… button. Double-click on the SSL QRS1QMGR certificate and select the Certification path tab. The certification path is displayed as follows:

DST RootCA X1 (this is the ROOT).

DST Root CA X3 (this is an intermediate or CA).

DEMO CA A6 (this is an intermediate or CA).

SSL QRS1QMGR (this is your queue manager certificate).

This certification path will be used later to determine which certificates need to be exported and added to the MQ client's certificate store. The MQ client needs access to all the Root and CA certificates in order to verify the MQ server's certificate.

Add the SSL QRS1QMGR certificate to queue manager TEST1's SSL certificate store by doing the following:

- Using the MQSeries Explorer, right-click on the TEST1 queue manager, select Properties, and choose the SSL tab.
- Select the Manage SSL certificates button and add the certificate to the queue manager store using the Add button. Check that the certificate is added by looking at the list in the window labelled Certificates in store for 'TEST1'.

Notice that in the window above, labelled 'Certificate assigned to this queue manager', the following text appears: 'No certificate has been assigned. SSL connections requiring authentication of this queue manager will fail.' Follow the instructions below to assign the certificate.

Assigning the certificate

Assign your certificate to the queue manager. Using the MQSeries Explorer, right-click on the TEST1 queue manager, select Properties, and choose the SSL tab. Select the Manage SSL certificates button. Click the Assign button, select the queue manager's certificate (eg SSL QRS1QMGR), and select the Assign button that appears.

Notice that in the window above, labelled 'Certificate assigned to this queue manager', the following text appears: 'Assigned certificate: SSL QRS1QMGR Certification Authority: DEMO CA A6'. Note that there is a minor bug in the MQ software (client and server), which causes it not to recognize valid certificates on the first day they are issued. This can be circumvented by waiting a day or advancing the date on the computer.

Obtain a temporary certificate from a certification authority for client use. In this example the certificate was obtained from Global Sign at *www.globalsign.com*. The Web link provides instructions for obtaining and downloading the certificate. The certificate is named using the e-mail address supplied (*eg yourname@your.com*).

Use IE to determine the certification path for this certificate by opening IE and selecting Tools/Internet options. Click on the Content tab and select the Certificates… button (see Figure 1). Double-click on the *yourname@your.com* certificate and select the Certification path tab. The certification path is displayed as follows (see Figure 2):

GlobalSign Root CA (this is the ROOT).

GlobalSign Primary Class 1 CA (this is an intermediate or CA).

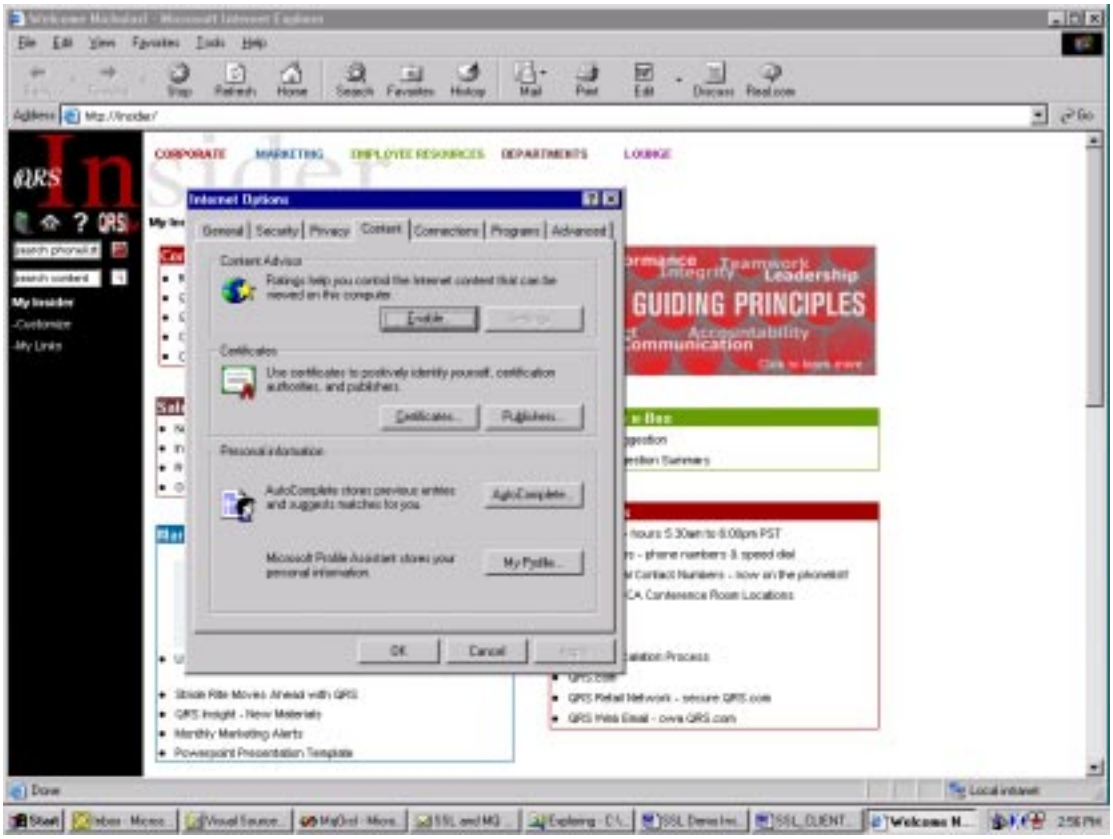

Figure 1: Determining the certification path

GlobalSign Class 1 CA (this is an intermediate or CA).

yourname@your.com (this is your client certicate).

This certification path will be used later to determine which certificates need to be exported and added to the MQ server's certificate store. The MQ server needs access to all the Root and CA certificates in order to verify the MQ client's certificate.

Ensure the MQSSLKEYR environment variable is set to the location where you want to locate the MQ client certificate store (*c:\mqm\key* in this example – see Figure 3). You should not include the extension *.sto* in the environment variable setting. The *key.sto* file is created from the default store the first time **amqmcert** is executed.

Add your client's personal certificate to the MQ client certificate store. Identify the handle number of the certificate in the personal

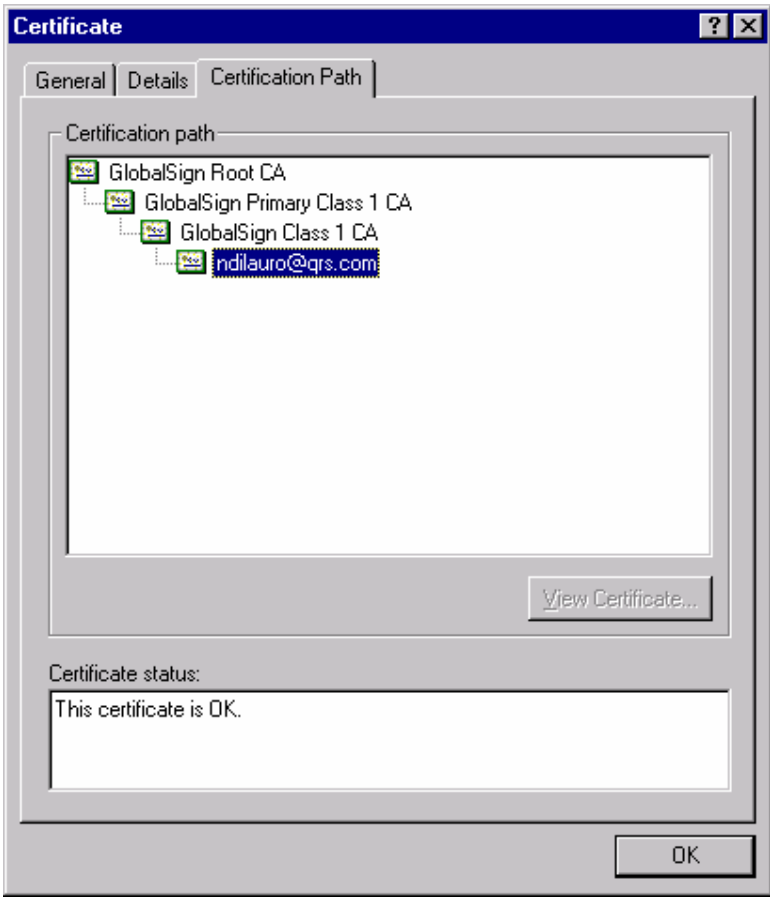

Figure 2: The certification path

certificate store (called MY) by executing the command **Amqmcert -k MY -l**. (The file *MY.TXT* in the Appendix lists the output of this command.)

Check that the client certificate *yourname@your.com* is listed and note the handle number (14001 in this example).

Now add the client certificate to the MQ client certificate store by executing the command **amqmcert -k MY -a 14001,** which in effect moves a copy of the certificate from the store of personal certificates (MY) to the MQ client's store in the file *key.sto*.

Check that the client certificate has been added by executing the command **amqmcert -l**. (The file *LIST.TXT* in the Appendix lists the output of this command.)

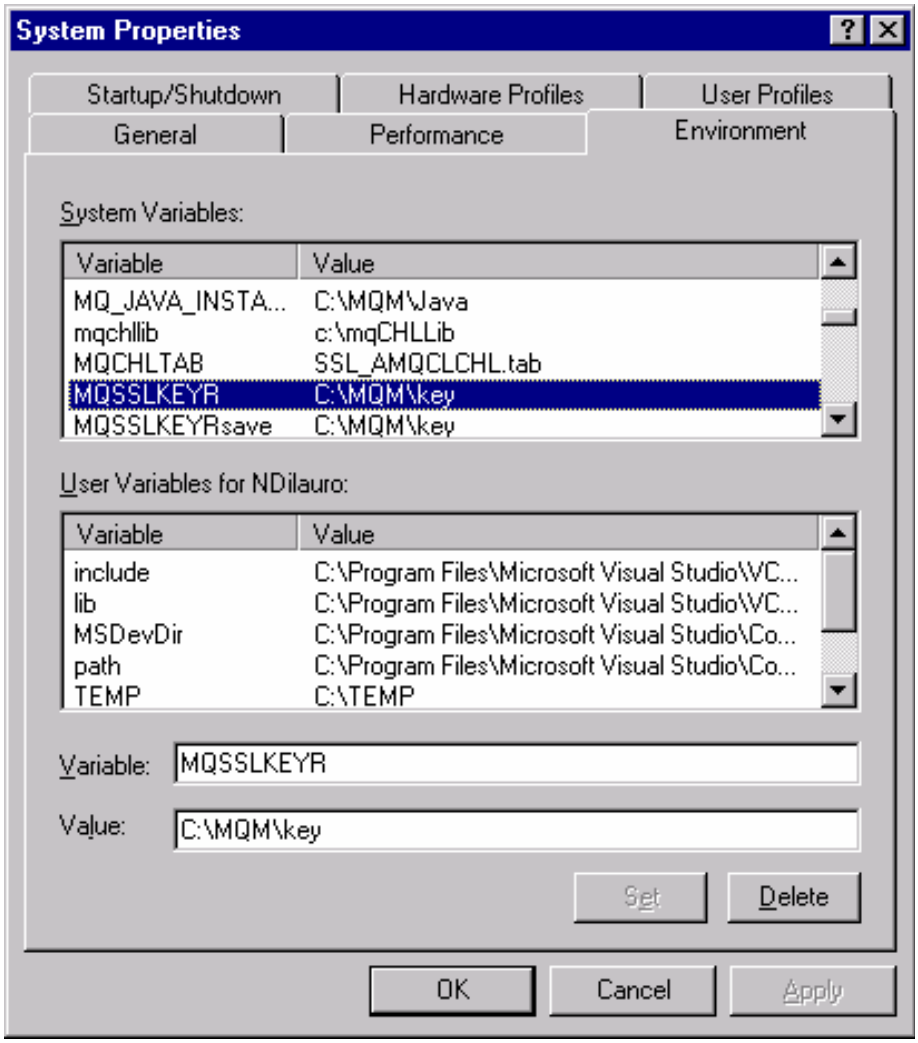

Figure 3: Locating the MQ client certificate store

Check that the certificate is listed and note the handle number. Notice that the handle number in the MQ client store is different from the handle number in the MY store. Make a note of the handle for the Client certificate (02014 in this example). Also, notice that at the beginning of the listing there is text to indicate that no certificate has been assigned to the MQ client.

Assign your personal client certificate (*yourname@your.com* in this example, handle 02014) to the MQ client by executing the command **amqmcert -d 02014**.

You can repeat the **amqcert -l** command to verify the certificate is assigned.

Validating the certificates

Now it is necessary to ensure that the MQ server certificate store has all the ROOT and intermediate (CA) certificates from the client's certificate chain in order to validate the client's certificate. Likewise, it is necessary for the client to have all the ROOT and CA certificates that are in the server's certificate chain in order to validate the server's certificate. These certificate chains have been discussed above. These steps may not be necessary if the certificates are among those pre-loaded for the MQ client and MQ server but for the demo certificates used in this test this was not the case.

To add the client root and intermediate certificates to the queue manager's store, first export all of the client's ROOT and intermediate certificates to separate files, using the Internet Explorer on the client, by opening IE and selecting Tools/Internet options. Click on the Content tab and select the Certificates… button. Select the Intermediate certification authorities tab and find the two intermediate certificates:

- GlobalSign Primary Class 1 CA.
- GlobalSign Class 1 CA.

Export each of these to separate files by highlighting the name and selecting Export and Next. Three options are listed. The default 'DER encoded binary X509 (*.CER*)' was chosen for this test. Click Next and choose a file name (*c:\mqm\GlobPrimClass1.cer* and *c:\mqm\GlobClass1*). Now select the Trusted root certification authorities tab to list the ROOT certificates. Select the GlobalSign Root CA certificate and export it to a file (*c:\mqm\GlobRoot.cer*), as was done for the two intermediate certificates.

Transfer these three files (*GlobRoot.cer*, *GlobPrimClass1.cer*, and *GlobClass1*) to the MQ server and import them into the TEST1 queue store by opening MQExplorer, right-clicking on TEST1 queue manager, and selecting Properties and the SSL tab. Select the Manage SSL certificates… button and select Add.

Now select Add from a file radio button, specify the file, and select the Add button that appears. Do this for each of the three files (*GlobRoot.cer*, *GlobPrimClass1.cer*, and *GlobClass1*). If one of these certificates is already in the store an error message will be displayed. Check that all three certificates are listed in the window for the queue manager's certificate store.

To add the queue manager's intermediate and root certificates to the MQ client's store, first export all of the queue manager's ROOT and intermediate certificates to separate files using the Internet Explorer on the server by opening IE and selecting tools/ Internet options. Click on the Content tab and select the Certificates… button. Select the Intermediate certification authorities tab and find the two intermediate certificates:

- DST Root CA X3.
- DEMO CA A6.

Export each of these to separate files by highlighting the name, selecting Export and Next. Three options are listed. The default 'DER encoded binary X509 (*.CER*)' was chosen for this test. Click Next and choose a file name (*c:\mqm\DSTX3.cer* and *c:\mqm\DEMOA6.cer*). Now select the Trusted root certification authorities tab to list the ROOT certificates. Select the DST RootCA X1 certificate and export it to a file (*c:\mqm\DSTROOT.cer*), as was done for the two intermediate certificates.

Transfer the three files from the MQ server (*DSTX3.CER*, *DEMOCA6.CER*, *DSTROOT.CER*) to the client platform and import them into the root and intermediate stores using Internet Explorer. For each of the three files open IE and select Tools/ Internet options. Click on the Content tab and select the Certificates… button. Select the Import button and then Next, specify the file name and select Next again, choose the default option 'Automatically select the store based on the type of certificate' or specify the store (intermediate or trusted root), and click on the Finish button.

Execute the command **amqmcert -k ROOT -l** to list the handle

number (14042 in this example) for the DST RootCA X1 certificate. (The file *ROOT.TXT* in the Appendix lists the output of this command.)

Add the queue manager's root certificate to the MQ client certificate store by executing the command **amqmcert -k ROOT -a 14042**.

Execute the command **amqmcert -k CA -l** to list the handle numbers (14002 and 14003 in this example) for the two intermediate certificates (DST Root CA and X3DEMO CA A6). (The file *CA.TXT* in the Appendix lists the output of this command.)

Add the queue manager's two intermediate certificates to the MQ client certificate store by executing the following commands **amqmcert -k CA -a 14002** and **amqmcert -k CA -a 14003**.

Define a svrconn channel on the MQ server TEST1 (TEST.CONN in this example). Under the SSL tab select an option 'RC4_MD5_US'.

Create a corresponding CLNTCONN channel on the server and select the same SSL option for this channel ('RC4_MD5_US' in this example). Export the chl table to the client machine and activate it by ensuring the correct MQCHLLIB and MQCHLTAB environment variables are set or that the channel is in the default directory with the default name.

Create a test queue on the server (TESTQ in this example).

On the client machine execute the test **amqsputc** and **amqsgetc** programs and put and get some messages to verify the connection:

- amqsputc TESTQ TEST1.
- amqsgetc TESTQ TEST1.

Nick Dilauro, MQSeries Administrator
QRS Corporation (USA)

Improving performance on SSL channels running on WSMQ for AIX

WHAT IS THE PROBLEM?

At V5.3 WSMQ provides support for Secure Socket Layer (SSL) security on its channels. SSL authentication involves the use of very CPU-intensive algorithms, which increase channel start-up times. The algorithms used to encrypt the user data also slow the channels to a significant extent. A common way to counter the performance impact of running SSL is to make use of hardware that is specifically designed to perform cryptographic functions very quickly. In this article I discuss the options for such hardware that are currently supported on WSMQ for the AIX platform.

HOW MIGHT CRYPTOGRAPHIC HARDWARE HELP?

It is worth looking first at the ways in which such hardware may be expected to improve performance. SSL communication can be divided into two major phases:

- *The initial authentication and key exchange* (also known as the SSL handshake). The algorithms used here involve large keys and much computation. They are very CPU-intensive. An SSL handshake is run every time a WSMQ SSL channel starts. WSMQ channels always use the algorithms supplied by the Rivest, Shamir, Adleman (RSA) company for authentication and key exchange. The key size used can vary, depending on the contents of the certificates and the CipherSpec specified on the channel definitions.
- *The transfer of user data*. On a WSMQ SSL channel messages are always signed digitally by applying a hashing algorithm to the user data and then encrypting the hashed value and appending it to the data. Generally the data itself is also encrypted. The hashing and encryption algorithms used are determined by the CipherSpec.

The signing and encryption/decryption of an individual item of data is not as intensive as the handshaking processing. As a general rule more secure algorithms affect channel performance more than less secure algorithms. However, it is also true that certain SSL algorithms are more efficient with regard to perfomance than others.

The relative importance of these two factors clearly depends on the use to which a channel is put. The handshake overhead is particularly important on channels that start and stop frequently; the data transfer overhead is particularly important for long-running channels.

Cryptographic hardware cards provide assistance with the CPU-intensive algorithms involved in authentication and key exchange. WSMQ makes use of this capability in the cards that it supports.

Some cryptographic hardware cards also have the potential to provide assistance with some of the algorithms used to encrypt data for data transfer. Many cards cannot be used to provide encryption assistance for data transfer and those cards that do provide this facility limit it to a restricted range of encryption algorithms.

WSMQ doesn't currently exploit this feature even if it is available on a particular type of card. So in the WSMQ Unix environment cryptographic hardware is particularly useful to improve performance for short-lived channels.

CRYPTOGRAPHIC CARDS SUPPORTED BY WSMQ (ON AIX)

At the time of writing, WSMQ only provides support for two cryptographic cards on the AIX platform:

- The IBM Cryptographic Coprocessor for IBM e-server pSeries: feature code 4963.
- The IBM Cryptographic Accelerator for IBM e-server pSeries: feature code 4960.

Both cards provide acceleration for authentication and key exchange on a WSMQ channel.

Note that both cards must be ordered separately from WSMQ. Note also that, at the time of writing, support is restricted to V4.3.3 of the AIX platform for the 4963 card and V5.1 of the AIX platform for the 4960 card. Please check at time of deployment as more versions of the AIX platform may be supported at that time.

PKCS #11

PKCS #11 is an abbreviation of Public Key Cryptography Standard 11. It originates from RSA but is an open industry standard. It defines an application programming interface (API) provided by a cryptographic product so it may be controlled by an application program.A specification of the interface is available by following the *PKCS #11: Cryptographic Token Interface Standard, version 2.0* link from the Web site *http://developer.netscape.com/docs/manuals/security/pkcs/index.htm*.

WSMQ uses this interface to control both the 4963 and the 4960 cryptographic cards.

While the details of the API between WSMQ and the cryptographic cards are clearly not of direct interest to the reader they are still relevant. For instance, when using a PKCS #11 card the personal certificate in use and generally also the CA certificates are stored by the cryptographic product and not directly by WSMQ. How the WSMQ user works with PKCS #11 cards is discussed below.

As mentioned, one particular feature of the PKCS #11 interface is that it allows the application (in this case WSMQ) to pass certificates through the PKCS #11 API so that they are stored within the PKCS #11 environment. With some PKCS #11 cards the certificates are physically stored on the card itself. This applies in the case of the 4963; it gives extra security against internal tampering with the certificates. With other PKCS #11 cards the certificates passed across the API are physically stored on disk. This applies on the 4960; the focus of the 4960 is speeding up the cryptography (using parallel processing) rather than providing extra certificate security.

When the cryptographic cards under discussion are received by the user they must be installed and configured to provide the PKCS #11 interface that WSMQ will use. For both the 4963 and the 4960 the basic installation of the hardware and its device driver is straightforward and is not discussed here. This should be successfully completed before the PKCS #11 configuration work described below.

The 4960 is rather simpler to set up than the 4963 because the card does not need to be initialized to hold certificates. The following instructions apply to both cards.

- The operating system must have basic PKCS #11 support. You must ensure that you have the current version of *bos.pkcs11* installed and that you rebooted after it was installed.
- Ensure that the operating system is at the current maintenance level.
- The user should access the manual entitled *IBM 4758 PCI Cryptographic Coprocessor PKCS #11 Support Program Installation Manual for IBM 4758 Models 002 and 023,* which can be found at *ftp://www6.software.ibm.com/software/cryptocards/PKCS11_INSTALL_241_Manual.pdf*.

  The 4758 runs in a Microsoft Windows environment and is very similar to the 4963. The 4758 manual includes instructions explicitly intended for setting up the 4963. Some of these apply to the 4960 also.
- If you have a 4963 follow instructions for the AIX platform in chapters three (*Installing the support program*), four (*Loading software into the Coprocesor*), and five (*Token Initialization*) of the manual referenced above. Note that the alphabetic characters in the token must be entirely in lower case if it is to be used by WSMQ.
- If you have a 4960 the setup necessary is described in the manual referenced above, in the following subsections of Chapter five, entitled *Token Initialization*:

– *Initialization of PKCS #11*

– *Setting of the User PIN*.

Unfortunately, on the AIX V5.1 platform, depending on the system software installed, you may run into the following misleading message when you select 'Initialize a token' in SMIT: 'There are no items of this type'. If you get this message you should run the commands from the command line as follows:

- Set the token label (the alphabetic characters in the token must be entirely in lower case if it is to be used by WSMQ).

  ```
  /usr/lib/pkcs11/methods/pkcsconf -I -c –
  ```

- Enter the SO PIN: 87654321.
- Enter a unique token label: testtoken.
- Set the security officer PIN. It is sensible for the security officer to change the security officer PIN directly after initializing the token, otherwise an unauthorized user could access the device and, for instance, reinitialize it or delete certificates and keys. Make sure you remember the new security officer PIN.

  ```
  /usr/lib/pkcs11/methods/pkcsconf -P -c –
  ```

- Enter the SO PIN: 87654321.
- Enter the new SO PIN: newsopin.
- Re-enter the new SO PIN: newsopin.
- The security officer sets the user PIN

  ```
  /usr/lib/pkcs11/methods/pkcsconf -u -c –
  ```

- Enter the SO PIN: newsopin.
- Enter the new user PIN: 12345678.
- Re-enter the new user PIN: 12345678.

These commands will have the desired effect but a spurious 'Memory fault' error message will be output. (Remember that these commands were only executed because we were getting

an error from SMIT in the first place.) The following commands are useful to confirm that the required changes have in fact taken place:

- **/usr/lib/pkcs11/methods/pkcsconf -?**
  - what parameters does pkcsconf have?.
- **/usr/lib/pkcs11/methods/pkcsconf -s**
  - PKCS #11 slot information.
- **/usr/lib/pkcs11/methods/pkcsconf -i**
  - general PKCS #11 information.
- **/usr/lib/pkcs11/methods/pkcsconf -t -c 0**
  - PKCS #11 token information.

You have now completed the set-up of PKCS #11 on the AIX platform for the cryptographic hardware. You will now need to set up WSMQ to use it.

CONFIGURING CERTIFICATES AND KEYS ON THE PKCS #11 SYSTEM

The WSMQ iKeyMan GUI tool is used to set up keys and certificates for WSMQ processing. This is activated by typing **gsk6ikm**. WSMQ use of this tool is documented in detail in the IBM manual *WebSphere MQ Security (SC34-6079)*. The section that describes how to get the certificates and keys onto your cryptographic cards is called *Configuring for cryptographic hardware*, which is a subsection of *Working with the Secure Sockets Layer (SSL) on Unix systems*. I will not repeat the information here, however, the following notes will help you with the specifics required for configuring the 4960 and 4963.

- The File Name field should contain *PKCS11_API.so*.
- The Location field should contain *usr/lib/pkcs11*.
- For normal use of the 4960 and 4963 you will not require a secondary CMS key database to hold the signer certificates

(ie there is room for them in the PKCS #11 environment).

- Note also that you can create self-signed certificates in the PKCS #11 environment.

CONFIGURING WSMQ RUNTIME FOR CRYPTOGRAPHIC HARDWARE

On a queue manager running on AIX the SSLCryptoHardware queue manager attribute contains the parameters that determine whether cryptographic hardware is in use and if so what kind it is and, if necessary, its parameters.

The form of the SSLCryptoHardware value for PKCS #11 hardware is defined in the IBM manual *WebSphere MQ Programmable Command Formats and Administration Interface (SC34-6060)* as:

```
GSK_PKCS11=<the PKCS #11 driver path and filename>;<the PKCS #11
token label>;<the PKCS #11 token password>;
```

The token password is the same as the user PIN.

So with the PKCS #11 software installed in the standard place on AIX and, assuming the manual pkcsconf configuration steps listed above, the SSLCryptoHardware parameter would be:

```
GSK_PKCS11=/usr/lib/pkcs11/PKCS11_API.so;testtoken;12345678;
```

(Note the mandatory terminating semi-colon.)

This parameter can also be input using the SSLCRYP MQSC parameter and the information can be entered from the WSMQ Windows Explorer.

On a WSMQ client running on the AIX platform the GSK_PKCS11 parameter can be specified using the MQSSLCRYP environment variable. The information can also be supplied on an MQCONNX call.

You are now configured to make use of your cryptographic hardware to reduce the start-up time on your SSL channels.

Mike Horan, WSMQ Base Development (distributed platforms)
IBM Hursley (UK)

dmpmqaut parser

INTRODUCTION

In my previous article on generic profiles (*MQ Update* Issue 41, November 2002) I explained how the **dmpmqaut** command introduced in WebSphere MQ (WSMQ) V5.3 can be used to recover all authority profiles for a particular queue manager. This involves dumping all of the authority profiles to a text file and then, when recovery is required, converting the profiles stored in the file into **setmqaut** statements. This article presents a program that will perform this conversion automatically, taking as input the file created by **dmpmqaut** and producing as output a file containing all **setmqaut** statements required to create exactly all authority profiles that existed at the point when the **dmpmqaut** command was issued. The program requires WSMQ V5.3 CSD 3.

USAGE

Once all of the authority profiles which will later require recovery have been defined, run the WSMQ **dmpmqaut** command, specifying only the queue manager name, and pipe the output to a text file. This will produce a file containing all authority profiles defined on the queue manager in verbose format. The file should be safely stored until recovery is needed.

To recover the authority profiles first compile the C program which accompanies this article. This will produce an executable that accepts the following parameters:

```
ParseDmp <input file> <output file> <queue manager name>
```

Next, run the parser, specifying as input the file stored earlier, containing all the authority profiles. Choose a suitable output file name and specify also the name of the queue manager being recovered. The parser will generate an output file containing all of the **setmqaut** statements required to recreate the authority

profiles. Finally, after recreating all of the queue manager objects, ensure that the queue manager is running and pipe the output file created by the parser to the current command shell.

Example

The example given here is for the Windows platform but the parser should run on any suitable platform – simply replace the pipe commands in this example with those appropriate to the operating system.

To store the authority profiles on a queue manager named qmgr1 in a file named *dmpmqaut.output*:

```
dmpmqaut -m qmgr1 2> dmpmqaut.output
```

To recreate the authority profiles:

```
ParseDmp dmpmqaut.output setmqaut.txt qmgr1
cmd.exe < setmqaut.txt
```

PARSER INTERNALS

The parser works by scanning the output from the **dmpmqaut** command twice. The first iteration is used to recreate all of the straightforward **setmqaut** statements pertaining to objects. Each authority record is examined and the various field values are used to construct a **setmqaut** statement that will exactly recreate the authority record.

One special case concerns an authority's records for queue manager objects; such records require **setmqaut** statements with no object type parameter. Also, during the first iteration an object name for each of the different object types is stored. These are required during the second iteration, which is used to recreate all of the **setmqaut** statements pertaining to authority class records. The syntax of the **setmqaut** command requires that each authority for a class of objects is set against an object of the appropriate type. Therefore, when reconstructing each **setmqaut** statement for a class record, the name of an object of the corresponding class that was stored during the first iteration is specified as the object name parameter.

```
/* ParseDmp.c                      #include <stdio.h>
#include <string.h>
#define BUFFER_SIZE       1Ø24
#define STRING_SIZE        128
int main( int argc, char *argv[] )
{
  FILE *InputStream, *OutputStream;
  char *pChar;
  char *pSearch;
  char Buffer[BUFFER_SIZE];
  char Command[BUFFER_SIZE];
  char EntityName[STRING_SIZE];
  char CurrentObjectName[STRING_SIZE];
  char ObjectName[5][STRING_SIZE];
  char *pAuthBuffer;
  char *pAuthCommand;
  int  Index;
  int  Iteration;
  int  ObjectClass;
  int  Error;
  int  ObjectStored[5];
  int  i;
  char *ObjectType[] = { "queue ",
           "process ",
           "namelist ",
           "authinfo ",
           "qmgr " };
  /* check arguments                     */
  if( argc != 4 )
  {
    printf( "Usage:\n" );
 printf( "ParseDmp <input file> <output file> <queue manager name>\n" );
    goto exit;
  }
  /* open input file in text mode               */
  InputStream = fopen( argv[1], "r" );
  if( InputStream == NULL )
  {
    printf( "Error, unable to open input file\n" );
    goto exit;
  }
  else
    printf( "Input file %s opened\n", argv[1] );
  /* open output file                    */
  OutputStream = fopen( argv[2], "w" );
  if( OutputStream == NULL )
  {
    printf( "Error, unable to open output file\n" );
```

```
    goto exit;
  }
  else
    printf( "Output file %s opened\n", argv[2] );
  /* initialize stored object array              */
  memset( ObjectStored, 0, sizeof(ObjectStored) );
  /* parse the input file twice:                 */
  /* - on the first iteration, all object and queue manager records are
*/
  /*   processed                   */
  /* - on the second iteration, all class records are processed     */
  for( Iteration = 0; Iteration <= 1; Iteration++ )
  {
    while( !feof( InputStream ) )
    {
      /* construct setmqaut statement            */
      sprintf( Command, "setmqaut -m %s ", argv[3] );
      Error = 0;
      for( Index = 0; Index < 5; Index++ )
      {
        pChar = fgets( Buffer, BUFFER_SIZE, InputStream);
        if( pChar != NULL)
        {
   /* remove return character                    */
   pSearch = strchr( Buffer, '\r' );
   if( pSearch )
     *pSearch = 0;
   /* remove newline character                   */
   pSearch = strchr( Buffer, '\n' );
   if( pSearch )
     *pSearch = 0;
   switch( Index )
   {
     case 0:
       /* process profile name, noting the following special cases:  */
       /* - if the profile name is @SELF, the authority applies to   */
       /*   the queue manager             */
       /* - if the profile name is @CLASS, the authority applies to  */
       /*   the object class              */
       if( !strcmp( &Buffer[13], "@CLASS" ) )
         ObjectClass = 1;
       else
         ObjectClass = 0;
       if( !Iteration && !ObjectClass )
       {
         /* the -n parameter is not required if the authority applies*/
         /* to the queue manager                */
         if( strcmp( &Buffer[13], "SELF" ) )
         {
           strcat( Command, "-n " );
```

```
      strcat( Command, &Buffer[13] );
      strcat( Command, " " );
      strcpy( CurrentObjectName, &Buffer[13] );
    }
  }
  break;
case 1:
  /* process object type, convert object type to index number   */
  for( i = 0; i < 4; i++ )
  {
    if( !strcmp( &Buffer[13], ObjectType[i] ) )
      break;
  }
  /* no object name processing required if the authority applies*/
  /* to the queue manager                    */
  if ( i != 4 )
  {
    if( Iteration && ObjectClass )
    {
      if( ObjectStored[i] )
      {
 /* this profile applies to the object class, so specify        */
 /* an object name of this object type      */
 strcat( Command, "-n " );
 strcat( Command, &ObjectName[i][0] );
 strcat( Command, " " );
      }
      else
      {
 /* if there are no objects of this type then this              */
 /* profile cannot be recreated             */
 printf( "Error, unable to recreate %sclass profile\n",
         ObjectType[i] );
 Error = 1;
      }
    }
    else
    {
      if( !ObjectClass && !ObjectStored[i])
      {
 /* store an object name of this type for use when              */
 /* processing the object classes           */
 strcpy( &ObjectName[i][0], CurrentObjectName );
 ObjectStored[i] = 1;
      }
    }
  }
  /* append the object type parameter       */
  strcat( Command, "-t " );
  strcat( Command, &Buffer[13] );
```

```
    break;
  case 2:
    /* process entity name                    */
    strcpy( EntityName, &Buffer[13] );
    break;
  case 3:
    /* process entity type                    */
    switch( Buffer[13] )
    {
      case 'p':
        /* entity type principal              */
        strcat( Command, "-p " );
        strcat( Command, EntityName );
        break;
      case 'g':
        /* entity type group                  */
        strcat( Command, "-g " );
        if( strchr( EntityName, '@' ) )
 memset( strchr( EntityName, '@' ), Ø, 1 );
        strcat( Command, EntityName );
        break;
      default:
        /* entity type unknown, this profile cannot be recreated  */
        if( Iteration == ObjectClass )
 printf( "Error, unknown entity type\n" );
        Error = 1;
        break;
    }
    break;
  case 4:
    /* process authority                      */
    strcat( Command, " " );
    pAuthBuffer  = Buffer + 13;
    pAuthCommand = Command + strlen(Command);
    while( pAuthBuffer < (Buffer + strlen( Buffer )) )
    {
      *pAuthCommand++ = '+';
      while( (*pAuthBuffer != Ø) && (*pAuthBuffer != ' ') )
        *pAuthCommand++ = *pAuthBuffer++;
      *pAuthCommand++ = *pAuthBuffer++;
    }
    /* add line feed at end of command        */
    strcat( Command, "\n");
    break;
  default:
    break;
}
     }
   }
   /* write command to output file            */
```

```
      if( (Iteration == ObjectClass) && !Error )
        fputs( Command, OutputStream );
      /* skip line of delimiters in input file                */
      pChar = fgets( Buffer, BUFFER_SIZE, InputStream );
    }
    /* reposition file pointer to beginning of input file     */
    if( fseek( InputStream, 0, SEEK_SET ) )
    {
      printf( "fseek() failed\n" );
      goto close_files;
    }
  }
  printf( "Parsing complete\n" );
  /* close files                    */
close_files:
  fclose( InputStream );
  fclose( OutputStream );
exit:
  return 0;
}
```

David Postlethwaite, MQSeries Development
IBM Hursley (UK)

In addition to *MQ Update*, the Xephon family of *Update* publications now includes *CICS Update*, *MVS Update*, *VSAM Update*, *DB2 Update*, *AIX Update*, *and RACF Update*. Although the articles published are of a very high standard, the vast majority are not written by professional writers, and we rely heavily on our readers themselves taking the time and trouble to share their experiences with others. Many have discovered that writing an article is not the daunting task that it might appear to be at first glance.

If you have ever experienced any difficulties with MQ, or made an interesting discovery, you could receive a cash payment, a free subscription to any of our *Updates*, or a credit against any of Xephon's wide range of products and services, simply by telling us all about it. For a copy of our *Notes for Contributors*, which explains the terms and conditions under which we publish articles, please point your browser at www.xephon.com/nfc.

Stellar Software has just announced the availability of TGEN for MQSeries.

According to Stellar, TGEN is a traffic generator and performance measurement solution that enables MQ developers and administrators to load test their MQSeries network before it goes into production to ensure that it will meet proposed application traffic requirements.

TGEN Version 2.5 adds custom message payload, XML report format, and licence key protection to its capabilities.

The company claims that TGEN streamlines the process of enterprise integration by simulating application traffic profiles and reporting on performance. The product is said to provide a simple, user-configurable application traffic generator and flexible input/output options.

TGEN V2.5 is available now. The active TGEN component is supported on Windows (2000, XP, NT, and 95/98), IBM AIX, Sun Solaris, Hewlett Packard HP-UX, and Stratus VOS. TGEN can interact with all other MQSeries platforms including OS/390 and AS/400.

For more information contact:
Stellar Software, 172 Mill Street, Holliston, MA 01746, USA.
Tel: +1 508 429 4473.
Fax: +1 508 429 4556.
Web: http://www.stellar-corp.com

* * *

MQSoftware says it has expanded the capabilities of its Q Pasa! middleware management application significantly with new features for Version 3. Apparently, in addition to tighter integration with IBM Tivoli, new features in Q Pasa! V3 include support for WebSphere Application Server 5.0, giving users the ability to monitor the WebSphere MQ family, databases, and WebSphere Application Server with one management tool.

The company claims that a new communications encryption function provides the ability to encrypt the communications link carrying the message traffic between Q Pasa! Configuration Manager and clients and agents to protect against unwanted viewers.

The addition of remote log and file viewing for WebSphere Application Server and WebSphere MQ is claimed to help administrators research and analyse WebSphere-related problems more quickly.

For more information contact:
MQSoftware,1660 South Highway 100, Suite 400, Minneapolis, Minnesota 55416, USA.
Tel: +1 952 345 8720.
Fax: +1 952 345 8721.
Web: http://www.mqsoftware.com

MQSoftware, Surrey Technology Centre, 40 Occam Road, Surrey Research Park, Guildford, Surrey, GU2 7YG, UK.
Tel: +44 1483 295400.
Fax: +44 1483 573704.

* * *