69

MQ update

March 2005

In this issue

A xephon PUBLICATION

MQ Update

Published by

Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA

Phone: 214-340-5690
Fax: 214-341-7081

Editor

Trevor Eddolls
E-mail: trevore@xephon.com

Publisher

Colin Smith
E-mail: info@xephon.com

Disclaimer

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

Subscriptions and back-issues

A year's subscription to *MQ Update*, comprising twelve monthly issues, costs $380.00 in the USA and Canada; £255.00 in the UK; £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 2000 issue, are available separately to subscribers for $33.75 (£22.50) each including postage.

Contributions

When Xephon is given copyright, articles published in *MQ Update* are paid for at the rate of $160 (£100 outside North America) per 1000 words and $80 (£50) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of $32 (£20) per 100 lines. To find out more about contributing an article, without any obligation, please 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.

Connecting queue managers using Internet Pass Thru

INTRODUCTION

Since Version 5.3, WebSphere MQ (WMQ) has been able to encrypt messages when transferring them from one queue manager to another. Messages that are held in queues are still not encrypted. This may be sufficient for WMQ systems in the intranet of a company, because it is much easier to sniff network transfers than to break into a WMQ system and have a look at the queue contents. But if a queue manager has to be connected to queue managers at other companies, the situation is much more complicated.

This article describes a solution to the problem of connecting the WMQ queue managers of two companies in a secure way, via the Internet. The solution uses the IBM SupportPac MS81 (WMQ Internet Pass Thru – IPT) to connect the systems.

THE SUPPORT PAC MS81

Reasons for using Internet Pass Thru

In general, connections between different companies have to pass several firewalls and demilitarized zones (DMZ). A DMZ is an area belonging to a company network, but it is separated from the intranet and Internet by firewalls. Connections through the DMZ should never pass both firewalls at once. Instead of direct connections, a system within the DMZ should act as a hub. This hub needs two network interfaces – one to the internal and one to the external firewall. Direct routes between the two firewalls should never exist.

One solution to setting up a WMQ hub within the DMZ would be a single queue manager on a DMZ system (see Figure 1). This queue manager may route messages from the corporate

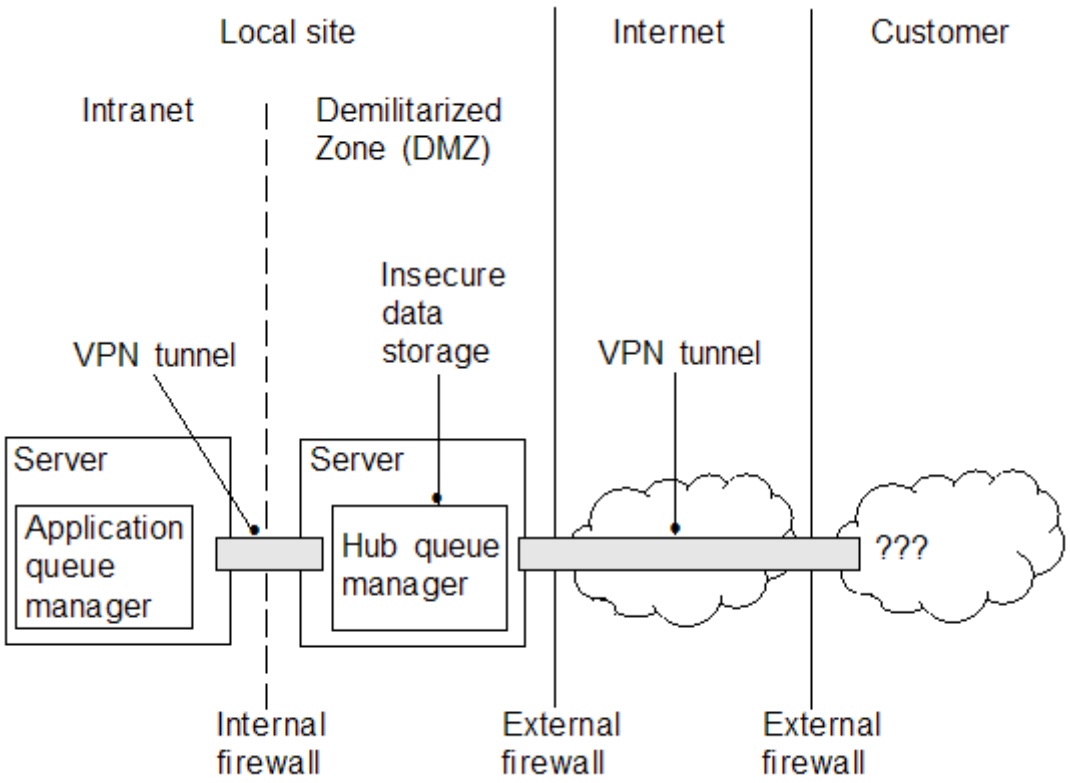

Figure 1: Securing the Internet connection with VPN

network to the customer through the Internet and back. During the transfer over the network, messages may be (and should be) encrypted. At least the connection through the Internet has to be encrypted – eg using a Virtual Private Network (VPN). Solutions using VPNs are possible for any version of WMQ. Nevertheless a hub queue manager needs transmission queues, to hold messages, until these can be sent to the destination queue manager. Messages in these transmission queues are not encrypted.

A DMZ is, by definition, an insecure network, whose job is to capture intruders and prevent them for entering the intranet. In such an environment it is not acceptable to store messages anywhere without encryption. Several companies provide solutions that are able to encrypt messages in WebSphere MQ queues, but mostly these are quite expensive. The use of SSL channel security in WMQ 5.3 encrypts messages during the transmission, but these are still readable when stored in

the queues. This may be OK for the sending and receiving queue managers in the intranet of your own company and on the customer site, but it is not acceptable at all for hub queue managers in a DMZ.

How Internet Pass Thru works

The IBM SupportPac MS81 (WMQ Internet Pass Thru) is a tool that acts like a proxy server for WebSphere MQ connections and does not store any messages locally. It is free of charge, and it could solve our problem. It pretends to be a WMQ queue manager but maps IP addresses and ports, so the participating queue managers do not need to know the real connection data of the partner systems. The IPT is also able to encrypt messages on the way to other IPTs – even queue managers with Version 5.2 and lower may use this selection. Additionally, IPT does not store any messages on the system as a hub queue manager would do in transmission queues.

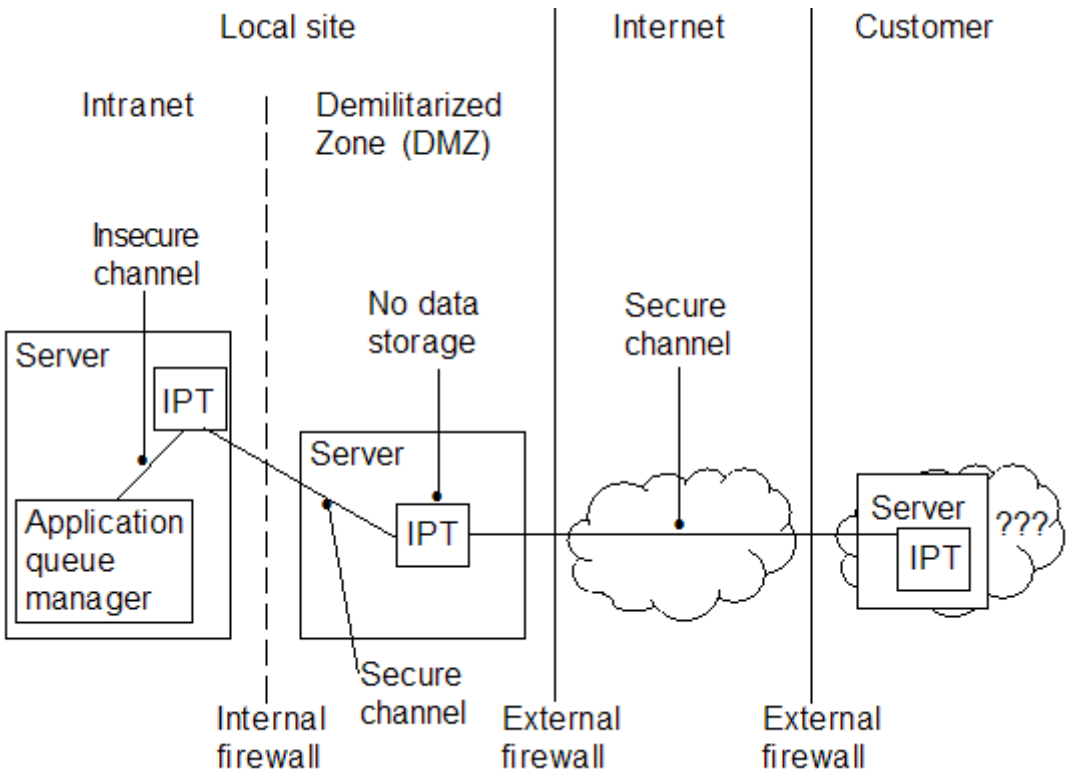

Figure 2: Securing the connection to the customer using IPTs

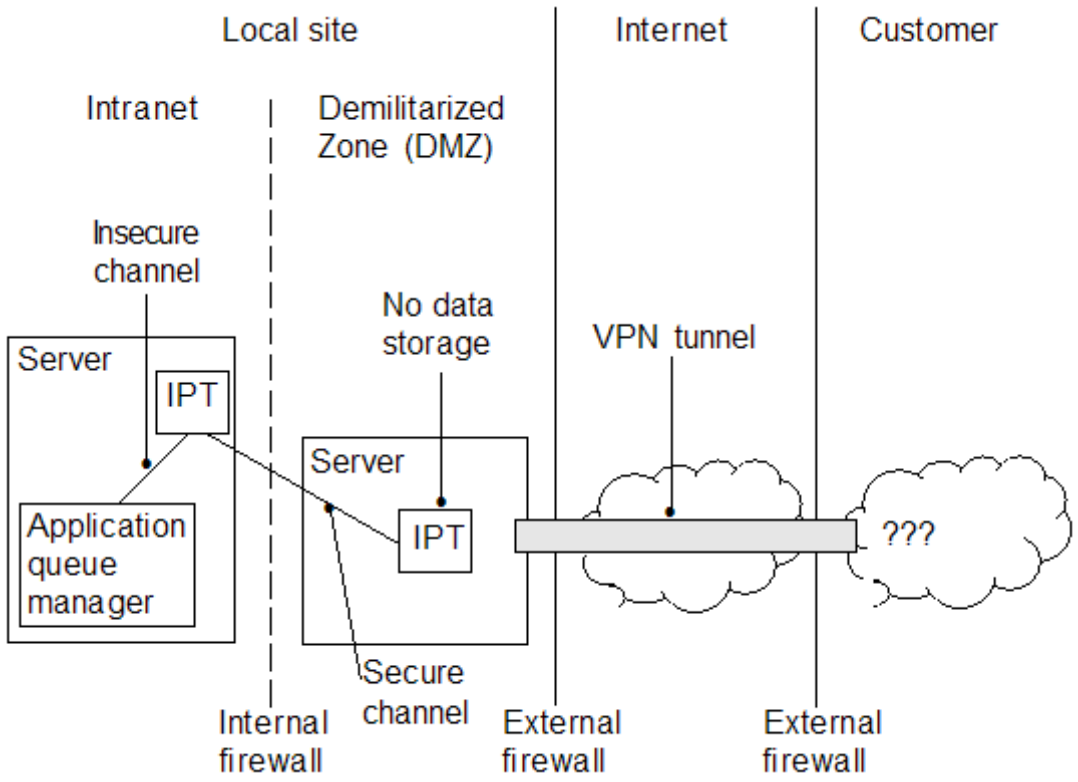

Figure 3: Securing the connection using IPTs and VPNs

The configuration shown in Figure 2 describes a WMQ queue manager – not necessarily a Version 5.3 system – which connects to a local IPT. This IPT encrypts and transfers the messages to another IPT on a DMZ system. The DMZ IPT itself is connected to a third IPT on the customer's side. We do not need to know whether there is another DMZ (although usually there will be) on the customer's side.

If the customer is not able, or does not want, to set up his own IPT, it is also possible to use a VPN connection through the Internet to the customer. The DMZ IPT will then connect to the customer's queue manager using an unencrypted channel through the VPN (see Figure 3). VPN connections between companies often exist already for other applications like FTP, whereas VPN connections to the corporate network are unusual. So the situation described in Figure 3 may be the most used.

Installation of the SupportPac

Download the SupportPac MS81 from the IBM home page and follow the instructions in the documentation to install the software. WebSphere MQ Internet Pass Thru comes as a platform-specific installable package, such as lpp for AIX or pkg for Sun Solaris.

Configure Internet Pass Thru

It is quite easy to configure the WebSphere MQ Internet Pass Thru. Define the listener port for incoming requests and the destination address and port for outgoing connections in a configuration file, as described in the samples below. Identify the listener port to the sending queue manager and start the IPT with this configuration. The destination parameter must fit the attributes of the channel listener on the receiving queue manager. Create channels on both queue managers as usual, but use the address and port of the IPT instead of the channel listener on the receiving side.

SAMPLES: CONNECTING TWO QUEUE MANAGERS VIA WMQ-IPT

The sample configuration consists of two queue managers, which have to be connected via one or more IPTs. The first sample uses one IPT as a gateway between two queue managers. In the second sample both queue managers are connected to separate IPTs. The IPTs themselves are connected together using SSL.

Configure two test queue managers

I created two queue managers on a Windows system. The queue managers are named QM1 and QM2 and listen to the ports 1421 and 1422 respectively. WebSphere MQ Internet Pass Thru is installed on a Sun Solaris system. Each queue manager has a sender channel to the IPT listener port and a transmission queue, a receiver channel, a remote queue for requests, and a local queue for answers. The configuration files are listed below.

QM1

```
DEFINE QLOCAL (RCVR.FROM.QM2) +
       REPLACE

DEFINE QREMOTE (SND.TO.QM2) +
       DEFPSIST(YES) +
       RNAME(RCVR.FROM.QM1) +
       RQMNAME(QM2) +
       XMITQ(QM2) +
       REPLACE

DEFINE QLOCAL (QM2) +
       USAGE(XMITQ) +
       TRIGGER +
       TRIGTYPE(FIRST) +
       TRIGDATA(QM1.QM2) +
       INITQ(SYSTEM.CHANNEL.INITQ) +
       REPLACE

DEFINE CHANNEL (QM1.QM2) CHLTYPE(SDR) +
       TRPTYPE(TCP) +
       CONNAME('10.10.104.42(1492)') +
       XMITQ(QM2) +
       REPLACE

DEFINE CHANNEL (QM2.QM1) CHLTYPE(RCVR) +
       TRPTYPE(TCP) +
       REPLACE
```

QM2

```
DEFINE QLOCAL (RCVR.FROM.QM1) +
       REPLACE

DEFINE QREMOTE (SND.TO.QM1) +
       DEFPSIST(YES) +
       RNAME(RCVR.FROM.QM2) +
       RQMNAME(QM1) +
       XMITQ(QM1) +
       REPLACE

DEFINE QLOCAL (QM1) +
       USAGE(XMITQ) +
       TRIGGER +
       TRIGTYPE(FIRST) +
       TRIGDATA(QM2.QM1) +
       INITQ(SYSTEM.CHANNEL.INITQ) +
       REPLACE
```

```
DEFINE CHANNEL (QM2.QM1) CHLTYPE(SDR) +
       TRPTYPE(TCP) +
       CONNAME('10.10.104.42(1491)') +
       XMITQ(QM1) +
       REPLACE

DEFINE CHANNEL (QM1.QM2) CHLTYPE(RCVR) +
       TRPTYPE(TCP) +
       REPLACE
```

One Internet Pass Thru used as a hub

In the first sample scenario (see Figure 4) the IPT acts as a proxy between the two queue managers. There are two route definitions in the IPT configuration file, to route messages from QM1 to QM2 and back. In the global section access is allowed for queue managers (QMgrAccess=true) and client access is prohibited (ClientAccess=true). See the listing of the configuration files below for more details.

```
[global]
CommandPort=1881
```

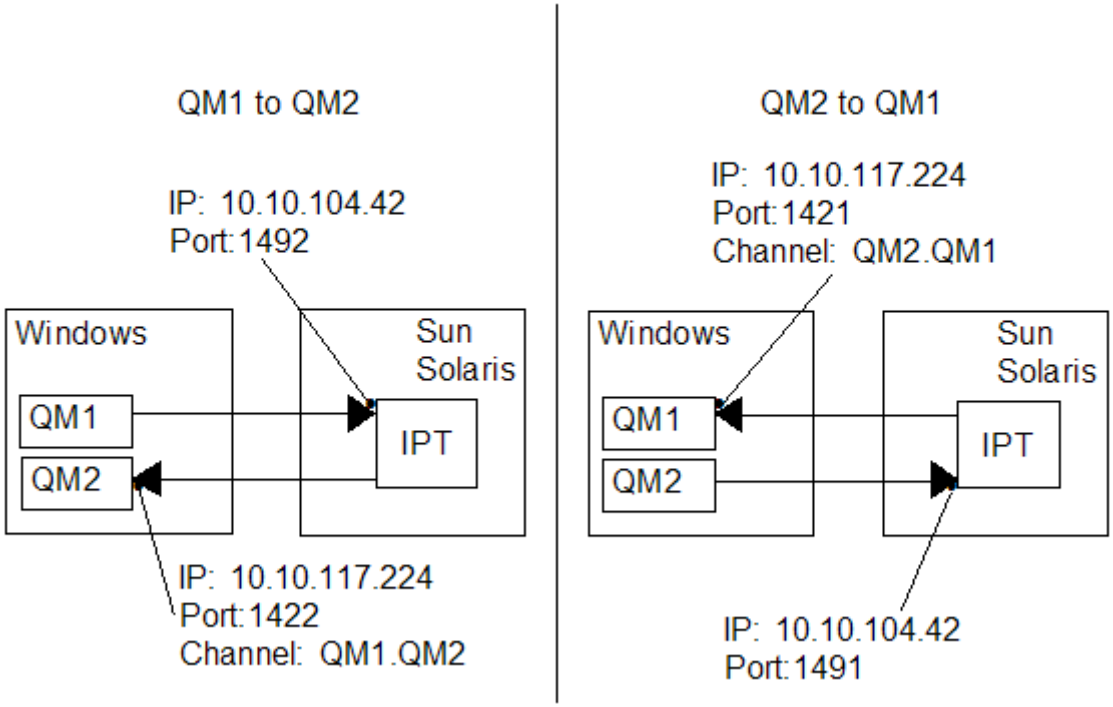

Figure 4: Sample with one IPT

```
RemoteShutDown=true
MinConnectionThreads=5
MaxConnectionThreads=100
IdleTimeout=20
ClientAccess=false
QMgrAccess=true
HTTP=false
HTTPChunking=false
Trace=5
ConnectionLog=true
MaxLogFileSize=50
AccessPW=MQIPT

#
# First route definition:
#

[route]
Name=Connection to QM1
Active=true
ListenerPort=1491
Destination=10.10.117.224
DestinationPort=1421

#
# Second route definition:
#

[route]
Name=Connection to QM2
Active=true
ListenerPort=1492
Destination=10.10.117.224
DestinationPort=1422
```

In the sample described above, the IPT acts simply as a router or proxy. The channels are not encrypted, but the queue managers need to know the real addresses and ports of their partner system.

WebSphere MQ Internet Pass Thru requires the Version 1.4 Java run-time environment. WebSphere MQ itself comes with Java Version 1.3, so you usually have to install the higher version. On my test system, Java 1.4 was installed in */usr/j2sdk1.4.2_05*. A small script called **runipt** includes this directory in the search path and starts WebSphere MQ Internet Pass Thru. The configuration file is located in the directory /

home/mqm/ipt. IPT expects its configuration always in a file named *mqipt.conf*.

runipt

The following script starts one IPT instance, which routes messages between the WebSphere MQ queue managers QM1 and QM2:

```
#!/bin/ksh

PATH=/usr/bin:/etc:/usr/sbin:/usr/ucb:/home/mqm/bin:/usr/bin/X11:/sbin:/
opt/local/bin:/usr/local/bin:.:/usr/j2sdk1.4.2_05/bin
export PATH

nohup /opt/mqipt/bin/mqipt /home/mqm/ipt &
```

Using two IPTs with a secured connection between

In the second sample I set up an SSL connection between two

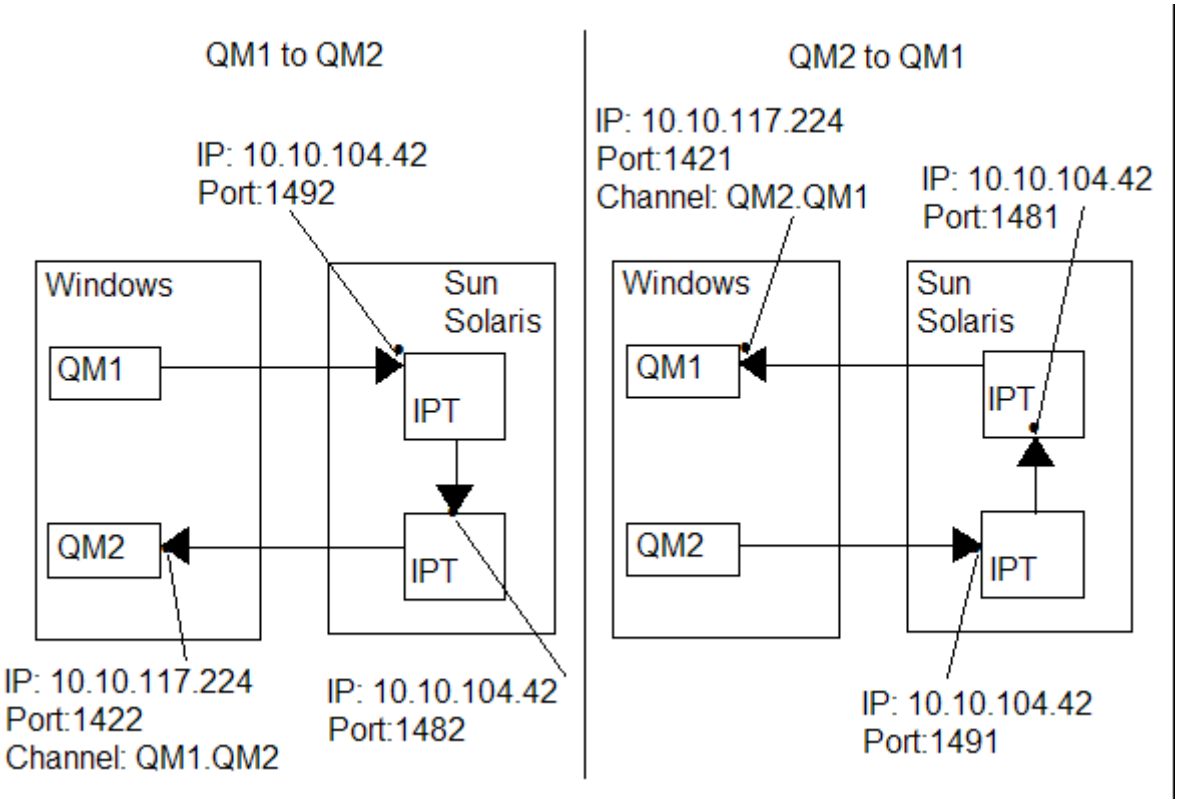

Figure 5: Sample with two IPTs

IPTs (this is illustrated in Figure 5). Now each queue manager is connected to a separate IPT. The queue managers do not see any difference, because both IPTs run on the same Sun Solaris system. I just created two directories, each with an IPT configuration file. The IPTs are now connected to a queue manager on one side and the other IPT on the other side. The SSL configuration allows any connection, but it is possible to restrict the access by using attributes like SSLServerDN_O … etc. See the IPT documentation for all available attributes. I used the SSL test certificates shipped with WebSphere MQ Internet Pass Thru, but I also tested it with my own private certificate. It is also possible to use different certificates for both IPTs when they are signed by the same Certification Authorization (CA) body. Keep in mind that the command port of the second IPT has been changed in order to run two instances of WebSphere MQ Internet Pass Thru on the same Sun Solaris system.

Corporate IPT

```
[global]
CommandPort=1881
RemoteShutDown=true
MinConnectionThreads=5
MaxConnectionThreads=100
IdleTimeout=20
ClientAccess=false
QMgrAccess=true
HTTP=false
HTTPChunking=false
Trace=5
ConnectionLog=true
MaxLogFileSize=50
AccessPW=MQIPT

#
# First route definition:
#

[route]
Name=Connection to QM1
Active=true
ListenerPort=1481
Destination=10.10.117.224
DestinationPort=1421
```

```
SSLClient=false
SSLServer=true
SSLServerKeyRing=/home/mqm/sslSample.pfx
SSLServerKeyRingPW=/home/mqm/sslSample.pwd
SSLServerCipherSuites=SSL_RSA_WITH_RC4_128_SHA
SSLServerDN_O=*

#
# Second route definition:
#

[route]
Name=Connection to DMZ IPT
Active=true
ListenerPort=1492
Destination=10.10.104.42
DestinationPort=1482
SSLClient=true
SSLClientKeyRing=/home/mqm/sslSample.pfx
SSLClientKeyRingPW=/home/mqm/sslSample.pwd
SSLClientCipherSuites=SSL_RSA_WITH_RC4_128_SHA
SSLClientDN_O=*
SSLServer=false
```

DMZ-IPT

```
[global]
# Command port changed to run two IPTs
CommandPort=1882
RemoteShutDown=true
MinConnectionThreads=5
MaxConnectionThreads=100
IdleTimeout=20
ClientAccess=false
QMgrAccess=true
HTTP=false
HTTPChunking=false
Trace=5
ConnectionLog=true
MaxLogFileSize=50
AccessPW=MQIPT

#
# First route definition:
#

[route]
Name=Connection to corporate IPT
Active=true
ListenerPort=1491
```

```
Destination=10.10.104.42
DestinationPort=1481
SSLClient=true
SSLClientKeyRing=/home/mqm/sslSample.pfx
SSLClientKeyRingPW=/home/mqm/sslSample.pwd
SSLClientCipherSuites=SSL_RSA_WITH_RC4_128_SHA
SSLClientDN_O=*
SSLServer=false

#
# Second route definition:
#

[route]
Name=Connection to QM2
Active=true
ListenerPort=1482
Destination=10.10.117.224
DestinationPort=1422
SSLClient=false
SSLServer=true
SSLServerKeyRing=/home/mqm/sslSample.pfx
SSLServerKeyRingPW=/home/mqm/sslSample.pwd
SSLServerCipherSuites=SSL_RSA_WITH_RC4_128_SHA
SSLServerDN_O=*
```

Run the Internet Pass Thru

Both IPTs are started with another small script called run2ipt.

run2ipt

```
#!/bin/ksh

PATH=/usr/bin:/etc:/usr/sbin:/usr/ucb:/home/f992953/bin:/usr/bin/X11:/
sbin:/opt/local/bin:/usr/local/bin:.:/usr/j2sdk1.4.2_05/bin
export PATH

nohup /opt/mqipt/bin/mqipt /home/mqm/ipt1 &
nohup /opt/mqipt/bin/mqipt /home/mqm/ipt2 &
```

More than two IPTs

To set up more than two IPTs is quite easy. Additional IPTs would have connections only to other IPTs, not to queue managers. The configuration is similar to the sample above for the IPT-IPT connection.

Files

The following files and directories are used in the samples above:

```
/home/mqm/ipt
/home/mqm/ipt/errors
/home/mqm/ipt/logs
/home/mqm/ipt/mqipt.conf

/home/mqm/ipt1
/home/mqm/ipt1/errors
/home/mqm/ipt1/logs
/home/mqm/ipt1/mqipt.conf

/home/mqm/ipt2
/home/mqm/ipt2/errors
/home/mqm/ipt2/logs
/home/mqm/ipt2/mqipt.conf

/home/mqm/run2ipt
/home/mqm/runipt

/home/mqm/sslSample.pfx
/home/mqm/sslSample.pwd
```

Hubert Kleinmanns
Senior Consultant
N-Tuition Business Solutions (Germany)

Directing SYSPRINT output to an HFS file

I have found that WebSphere for z/OS customers who are used to a Unix or NT environment are reluctant to use the facilities of SDSF, or IOF, to view the SYSPRINT output from their Application Server regions. They prefer to use vi in a TELNET session to view the STDOUT and STDERR streams. These streams have been redirected to SYSPRINT, and it is possible to redirect them to files in the HFS for viewing using such an editor.

The JCL below is for an Application Server region that uses

this facility. A new SET statement has been added to point to the LOGPATH directory, and the SYSPRINT DD statement has been changed to point to a file in the LOGPATH/servername directory, in this case named:

```
thx.log.d&LYYMMDD..t&LHHMMSS
```

Note that the extra period (full stop) between the date and time variables is not an error, but rather a requirement of the JCL syntax, and is necessary to terminate the first variable, where &LYYMMDD will be replaced with the local date in YYMMDD format and &LHHMMSS will be replaced by the local time in HHMMSS format. Using the local date and time ensures a unique file for each instance of the Application Server region that is started. The PATHMODE subparameter sets the file mode to 775, the PATHOPTS subparameter OWRONLY opens the file for WRITE access, and the subparameter OCREAT indicates that if the file does not already exist, it will be created.

The JCL for the Application Region is:

```
//JXB7884S   PROC IWMSSNM='JXB7884S',PARMS='-ORBsrvname '
//  SET CBCONFIG='/WebSphere/TSDCONF'
//  SET LOGPATH='/WebSpherelogs'
//  SET RELPATH='controlinfo/envfile'
//WSDAS1S    EXEC PGM=BBOSR,REGION=0M,TIME=NOLIMIT,
// PARM='/ &PARMS &IWMSSNM'
//BBOENV     DD PATH='&CBCONFIG/&RELPATH/&SYSPLEX/&IWMSSNM/current.env'
//CEEDUMP    DD SYSOUT=*
//SYSOUT     DD SYSOUT=*
//SYSPRINT   DD PATHMODE=(SIRWXU,SIRWXG,SIROTH),
//           PATHOPTS=(OWRONLY,OCREAT),
//           PATH='&LOGPATH/JXB7884S/thx.log.d&LYYMMDD..t&LHHMMSS'
```

In the above example, if the file specified by the SYSPRINT DD statement is created on 03 June 2004 at 3:30:40pm, the PATH parameter will resolve to */WebSphere/logs/JXB7884S/thx.log.d030604.t153040*, which will be a unique file for this execution of this server instance.

John Bradley
Systems Programmer
Meerkat Computer Services (UK)

CICS TS 3.1 and WebSphere Studio Enterprise Developer

While IBM was celebrating the 35th birthday of its popular transaction monitor, CICS, it also continued to enhance its functionality. On 30 November 2004, IBM announced a new release of CICS Transaction Server, Version 3.1, with a general availability date of 25 March 2005.

IBM also announced, as a statement of direction, an intention to add CICS TS 3.1 support to WebSphere Studio Enterprise Developer (WSED) during 2005. This is meant to allow developers with skills in COBOL, PL/I, Java, and Web services to reuse, build, and deploy components that integrate into an enterprise-wide SOA (Service-Oriented Architecture). WSED is to provide the visual development environment supporting Web services, SOAP for CICS, and aggregation of CICS resources.

IBM considers WSED to be a strategic development environment, now with added CICS TS V3 support. An optimized CICS data exchange capability and the ability to use a single development tool (such as WSED) provides enhanced application transformation capabilities and increases developer productivity.

New application development tools will extend WSED to allow the composition of CICS application assets to form business service functions that can be used as Web services. This will enable an external business process engine, such as WebSphere Business Integration Server Foundation, to externally orchestrate business service functions implemented in CICS. This means that sites can extend the use of their CICS applications in a service-oriented manner, integrating their CICS investments into other parts of the business.

IBM will provide a batch program for use by automated software build procedures, such as JCL, which will input the

XML Schema Definition (XSD) or language structure declaration to generate client Web Services Description Language (WSDL) and converters for the CICS Web services implementation.

Elena Nanos
IBM Certified Solution Expert in CICS Web Enablement and MQSeries
Zurich NA (USA)

Start/stop message flow from another message flow

Message flows process messages as they arrive on the input queue. This is how message flows are built to work. However, some clients would like the message flow to process the messages only after a certain event has occurred. This event might be the input queue reaching a certain depth, a trigger message arriving, or a certain time occurring. For requirements like these, we have to alter the behaviour of message flows and find a way to start/stop the message flow when the event occurs.

The most common approach, as far as stopping the message flow is concerned, is to act on the input queue, making it get-inhibited (or get-enabled to start it). This means that the message flow is as good as stopped because it cannot get any messages from the input queue. Refer to *MQ Update,* issue 28, October 2001 for details of issuing a PCF command from a message flow to change the attribute of the input queue.

The problem with this approach is that we are not really stopping or starting the message flow, we are just manipulating the input queue's attribute to mimic the message flow start/stop. Because the input queue's attribute was changed to get-inhibited, and the message flow is still running and trying

constantly to read messages from it, we will have quite a number of complaining mqrc2016 log messages issued. These messages may (or may not) be undesirable, depending on how the monitoring tool is set up.

To avoid this pitfall, we need to really stop and start the message flow, not mimic the action, by actually issuing the start/stop commands. We know this can be done because we can start/stop message flows from the toolkit. We just have to figure out where to issue the command and what the command looks like!

In the manual *Messagebroker Application Programming* we read that the **mqsicreatebroker** command is sent to the SYSTEM.BROKER.ADMIN.QUEUE – so this is where we will issue the command. The next thing we need to figure out is what the command looks like. This of course can be done by the trick of changing the attribute of the SYSTEM.BROKER.ADMIN.QUEUE to get-inhibited, such that the broker will not be able to get the command message from the queue. On the toolkit, open the Broker Administration perspective, right-click on the message flow, and issue a stop command on the Domain view. We can then use the RfhUtil to read the message, which looks like:

```
<Broker label="WBRKBKR" uuid="589ebc8a-ff00-0000-0080-d315c1b06539"
version="1">
      <ExecutionGroup uuid="599ebc8a-ff00-0000-0080-d315c1b06539">
            <Stop>
                  <MessageFlow uuid="1708ecb3-0001-0000-0080-
bf6cb11a4390"/>
            </Stop>
      </ExecutionGroup>
</Broker>
```

Similarly, we can use the same method to get the start command syntax:

```
<Broker label="WBRKBKR" uuid="589ebc8a-ff00-0000-0080-d315c1b06539"
version="1">
      <ExecutionGroup uuid="599ebc8a-ff00-0000-0080-d315c1b06539">
            <Start>
                  <MessageFlow uuid="1708ecb3-0001-0000-0080-
bf6cb11a4390"/>
```

```
            </Start>
        </ExecutionGroup>
</Broker>
```

As shown above, the command is very simple and straightforward. The outermost tag is <Broker>, with <ExecutionGroup> as child, and then the command either <Start> or <Stop>, followed by the <MessageFlow>.

The only challenge left is to figure out what are the uuids for a given Broker name, ExecutionGroup name, and the MessageFlow name. This can be done in two ways: either by issuing queries to the Broker to find out all the uuids, or by simply querying the CMDB database directly.

To query the uuids from the Broker is not difficult, but it is an iterative process. First we have to find out the uuid of the Broker, and with the information returned from the reply message we can then query the uuids of the ExecutionGroup, and then, in turn, that of the MessageFlow.

To start the process, issue a command as follows:

```
<Broker uuid="?"/>
```

The command is to be issued as a request message, and it specifies the replyToQueue to retrieve the reply. Be careful when doing this with the RfhUtil. The RfhUtil is a wonderful tool, but there is a little bug in it (the latest version may have fixed this bug!). Even after you specify the MsgType as ‘1 Request’ on the MQMD panel, it still issues the message as a ‘8 Datagram’ – see Figure 1.

The result of issuing the command as a datagram instead of request is that the reply message will fail as indicated by the <OverallCompletionCode result="failure"> tag. The simplest way may be by using a simple message flow to put the command.

A successful reply message will be as follows:

```
<Broker uuid="589ebc8a-ff00-0000-0080-d315c1b06539" label="WBRKBKR"
version="1">
```

```
      <OverallCompletionCode result="failure">
              <LogEntry catalog="BIPv500" number="2045">
                     <Insert type="string" text="WBRKBKR"/>
                     <Insert type="string" text="589ebc8a-ff00-0000-
0080-d315c1b06539"/>
                     <Insert type="string" text="WBRKQM"/>
                     <Insert type="string" text="?"/>
              </LogEntry>
              <LogEntry catalog="BIPv500" number="2087">
                     <Insert type="string" text="WBRKBKR"/>
                     <Insert type="string" text="589ebc8a-ff00-0000-
0080-d315c1b06539"/>
              </LogEntry>
       </OverallCompletionCode>
</Broker>
```

Note that even though the <OverallCompletionCode result="failure">, this is OK for the Broker uuid query since the

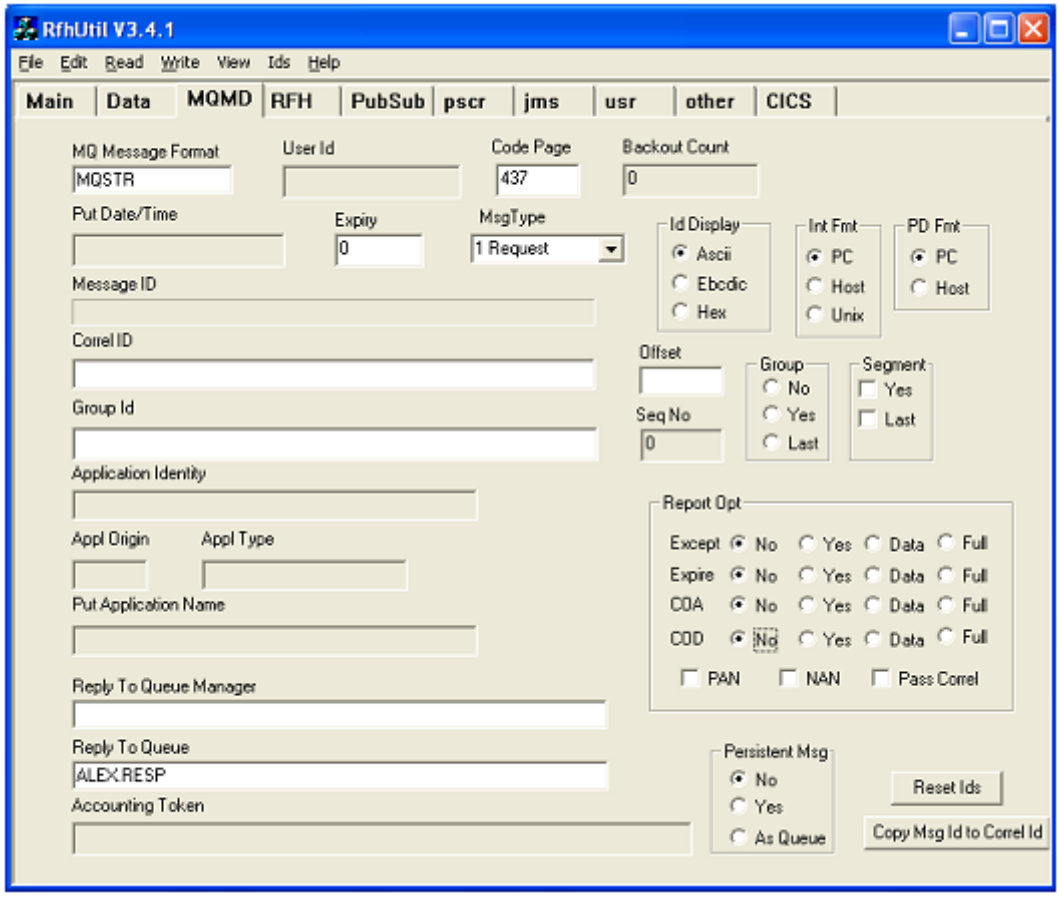

Figure 1: The MQMD panel

Broker uuid is returned.

With the Broker uuid found, we can then query the uuids for the ExecutionGroup, as follows:

```
<Broker uuid="589ebc8a-ff00-0000-0080-d315c1b06539" label="WBRKBKR"
version="1">
      <Report recursive="no">
            <AllExecutionGroups/>
      </Report>
</Broker>
```

The reply message will look like this:

```
<Broker uuid="589ebc8a-ff00-0000-0080-d315c1b06539" label="WBRKBKR"
version="1">
      <OverallCompletionCode result="success">
            <LogEntry catalog="BIPv500" number="2056">
                  <Insert type="string" text="WBRKBKR"/>
                  <Insert type="string" text="589ebc8a-ff00-0000-0080-
d315c1b06539"/>
            </LogEntry>
      </OverallCompletionCode>
      <ExecutionGroupCompletionCode uuid="599ebc8a-ff00-0000-0080-
d315c1b06539" result="success">
            <LogEntry catalog="BIPv500" number="4040">
                  <Insert type="string" text="default"/>
                  <Insert type="string" text="599ebc8a-ff00-0000-0080-
d315c1b06539"/>
            </LogEntry>
      </ExecutionGroupCompletionCode>
      <ReportResponse>
            <ExecutionGroup uuid="599ebc8a-ff00-0000-0080-d315c1b06539"
userTraceLevel="debugTrace" traceLevel="none"
userTraceFilter="debugTrace" traceFilter="none" label="default"
unnamedUserTraceLevel="none" unnamedTraceLevel="none" consoleMode="off"/
>
      </ReportResponse>
</Broker>
```

I have only the default execution group defined in my Broker. (As a sidetrack, something that you may or may not have noticed: once you create an ExecutionGroup and deploy message flows to it, even though the message flow within it may not be running or may have been removed, each DataFlowEngine can still take up to 57KB of memory. This uses up a lot of memory on my workstation, and that's why I keep only one ExecutionGroup.) With the uuid of the

ExecutionGroup (in my case it's the default), we then further query the uuids for the MessageFlows that run in that ExecutionGroup:

```
<Broker uuid="589ebc8a-ff00-0000-0080-d315c1b06539" label="WBRKBKR"
version="1">
      <ExecutionGroup  uuid="599ebc8a-ff00-0000-0080-d315c1b06539">
            <Report recursive="no">
                  <AllMessageFlows/>
            </Report>
      </ExecutionGroup>
</Broker>
```

The reply message will look like this:

```
<Broker uuid="589ebc8a-ff00-0000-0080-d315c1b06539" label="WBRKBKR"
version="1">
      <OverallCompletionCode result="success">
            <LogEntry catalog="BIPv500" number="2056">
                  <Insert type="string" text="WBRKBKR"/>
                  <Insert type="string" text="589ebc8a-ff00-0000-0080-
d315c1b06539"/>
            </LogEntry>
      </OverallCompletionCode>
      <ExecutionGroupCompletionCode uuid="599ebc8a-ff00-0000-0080-
d315c1b06539" result="success">
            <LogEntry catalog="BIPv500" number="4040">
                  <Insert type="string" text="default"/>
                  <Insert type="string" text="599ebc8a-ff00-0000-0080-
d315c1b06539"/>
            </LogEntry>
      </ExecutionGroupCompletionCode>
      <ExecutionGroup uuid="599ebc8a-ff00-0000-0080-d315c1b06539">
            <ReportResponse>
                  <MessageFlow uuid="0d59c48a-ff00-0000-0080-
bf6cb11a4390" userTraceLevel="none" traceLevel="none"
userTraceFilter="debugTrace" traceFilter="none" label="Test_failure"
additionalInstances="0" commitCount="1" commitInterval="0"
coordinatedTransaction="no" StatsArchivalOn="inactive"
StatsArchiveThreadDataLevel="none" StatsArchiveNodeDataLevel="none"
StatsSnapPublicationOn="inactive" StatsSnapThreadDataLevel="none"
StatsSnapNodeDataLevel="basic" StatsArchiveOutputFormat="usertrace"
StatsSnapOutputFormat="xml" StatsArchiveReset="no"
StatsArchiveAccountingOrigin="none" StatsSnapAccountingOrigin="none"/>
                  <MessageFlow uuid="6c6fcdb3-0001-0000-0080-
bf6cb11a4390" userTraceLevel="none" traceLevel="none"
userTraceFilter="debugTrace" traceFilter="none" label="Test_failure"
additionalInstances="0" commitCount="1" commitInterval="0"
coordinatedTransaction="no" StatsArchivalOn="inactive"
```

```
StatsArchiveThreadDataLevel="none" StatsArchiveNodeDataLevel="none"
StatsSnapPublicationOn="inactive" StatsSnapThreadDataLevel="none"
StatsSnapNodeDataLevel="none" StatsArchiveOutputFormat="usertrace"
StatsSnapOutputFormat="usertrace" StatsArchiveReset="no"
StatsArchiveAccountingOrigin="none" StatsSnapAccountingOrigin="none"/>
                  <MessageFlow uuid="Configuration"
userTraceLevel="none" traceLevel="none" userTraceFilter="debugTrace"
traceFilter="none" label="ConfigurationMessageFlow"
additionalInstances="0" commitCount="1" commitInterval="5"
coordinatedTransaction="no" StatsArchivalOn="inactive"
StatsArchiveThreadDataLevel="none" StatsArchiveNodeDataLevel="none"
StatsSnapPublicationOn="inactive" StatsSnapThreadDataLevel="none"
StatsSnapNodeDataLevel="none" StatsArchiveOutputFormat="usertrace"
StatsSnapOutputFormat="usertrace" StatsArchiveReset="no"
StatsArchiveAccountingOrigin="none" StatsSnapAccountingOrigin="none"/>
                  <MessageFlow uuid="PubSubControl"
userTraceLevel="none" traceLevel="none" userTraceFilter="debugTrace"
traceFilter="none" label="PubSubControlMsgFlow" additionalInstances="0"
commitCount="1" commitInterval="5" coordinatedTransaction="no"
StatsArchivalOn="inactive" StatsArchiveThreadDataLevel="none"
StatsArchiveNodeDataLevel="none" StatsSnapPublicationOn="inactive"
StatsSnapThreadDataLevel="none" StatsSnapNodeDataLevel="none"
StatsArchiveOutputFormat="usertrace" StatsSnapOutputFormat="usertrace"
StatsArchiveReset="" StatsArchiveAccountingOrigin="none"
StatsSnapAccountingOrigin="none"/>
            </ReportResponse>
      </ExecutionGroup>
</Broker>
```

As you can see, using message flow to issue the command and interpret the reply messages can be quite complicated. The other approach is much simpler. We first query the CMDB database for the uuid for the Broker, and then that of the ExecutionGroup, and the MessageFlow.

The uuid of Broker can be found in the table CBROKER; the uuid of ExecutionGroup can be found in the table CEG; and the uuid of the MessageFlow can be found in table CMSGFLOW.

Armed with this information, we can begin coding the message flow. The input to this message flow will consist of the name of the Broker, the ExecutionGroup, and the MessageFlow, and the action (whether we want to start or stop it). The simplest form will be in XML format as shown:

```
<MFCmd>
      <BrokerName>WBRKBKR</BrokerName>
      <EGName>default</EGName>
      <MFName>StopMsgFlow</MFName>
      <Action>Stop</Action>
</MFCmd>
```

or:

```
<MFCmd>
      <BrokerName>WBRKBKR</BrokerName>
      <EGName>default</EGName>
      <MFName>StopMsgFlow</MFName>
      <Action>Start</Action>
</MFCmd>
```

The message flow itself is very simple and is shown in Figure 2.

The compute node Read_Config_MFCmd will have the CMDB specified as Data Source; if you are running the Broker in the same box as the ConfigMgr, the ODBC has already been defined – see Figure 3.

The ESQL is also very simple; first we set up the message as a request and specify where the ReplyToQ the Broker will put the reply:

```
CALL CopyMessageHeaders();
            -- CALL CopyEntireMessage();

            -- make sure MQMD is set correctly
            -- MsgType = request
            -- ReplyToQ specified
            SET OutputRoot.MQMD.MsgType = 1;
            SET OutputRoot.MQMD.ReplyToQ = 'ALEX.RESP';
```

The next task is to get the uuids from the CMDB database tables:

```
Declare BKUUID, EGUUID, MFUUID char;
            -- database read to get the UUIDs from CMDB

            SET Environment.Variables.Result[] = (SELECT B.CUUID FROM
Database.alexau.CBROKER AS B where B.CNAME =
InputBody.MFCmd.BrokerName);
            SET BKUUID = Environment.Variables.Result[1].CUUID;

            SET Environment.Variables.Result[] = (SELECT E.CUUID FROM
```

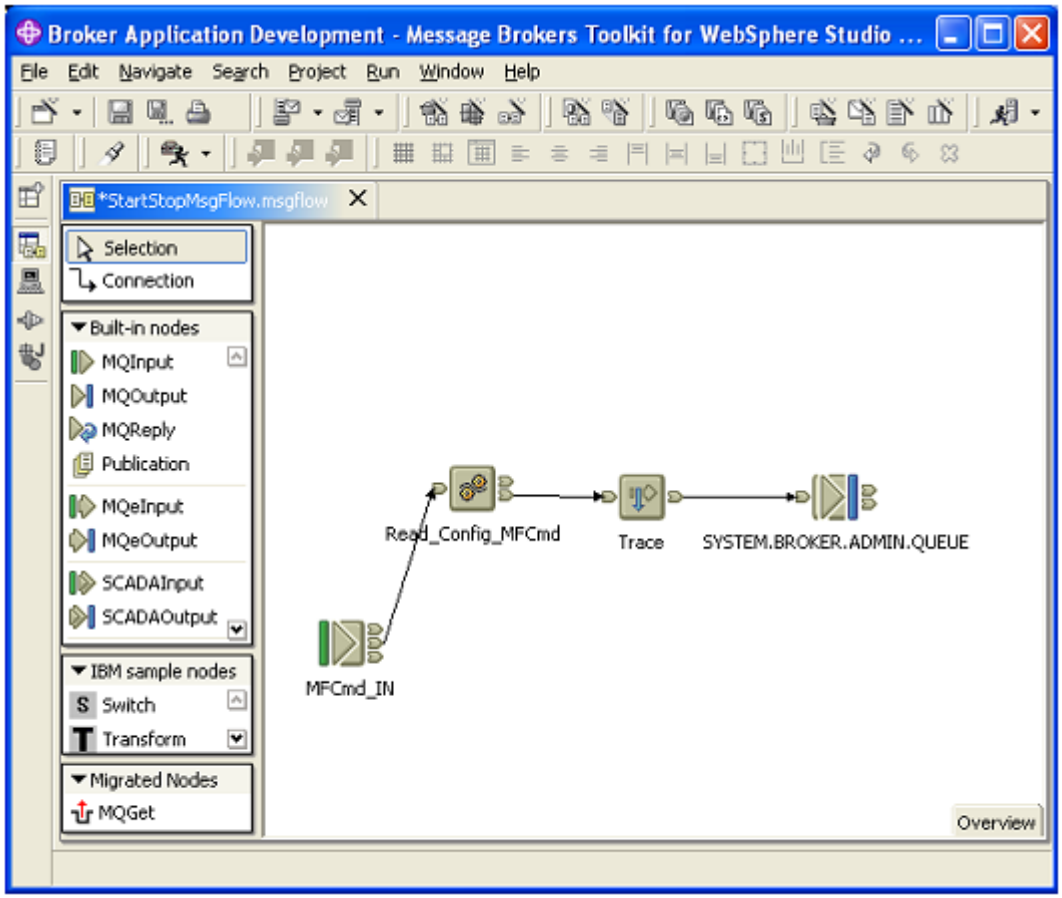

Figure 2: The message flow

```
Database.alexau.CEG AS E where E.CBROKERCUUID = BKUUID and E.CNAME =
InputBody.MFCmd.EGName);
            SET EGUUID = Environment.Variables.Result[1].CUUID;

            SET Environment.Variables.Result[] = (SELECT M.CUUID FROM
Database.alexau.CMSGFLOW AS M where M.CBROKERCUUID = BKUUID and
M.CEGCUUID = EGUUID and M.CNAME = InputBody.MFCmd.MFName);
            SET MFUUID = Environment.Variables.Result[1].CUUID;
```

The final thing is to create the command:

```
-- make the command now
            SET OutputRoot.XML.Broker.(XML.attr)label =
InputBody.MFCmd.BrokerName;
            SET OutputRoot.XML.Broker.(XML.attr)uuid = BKUUID;
            SET OutputRoot.XML.Broker.(XML.attr)version = '1';
            SET OutputRoot.XML.Broker.ExecutionGroup.(XML.attr)uuid =
```

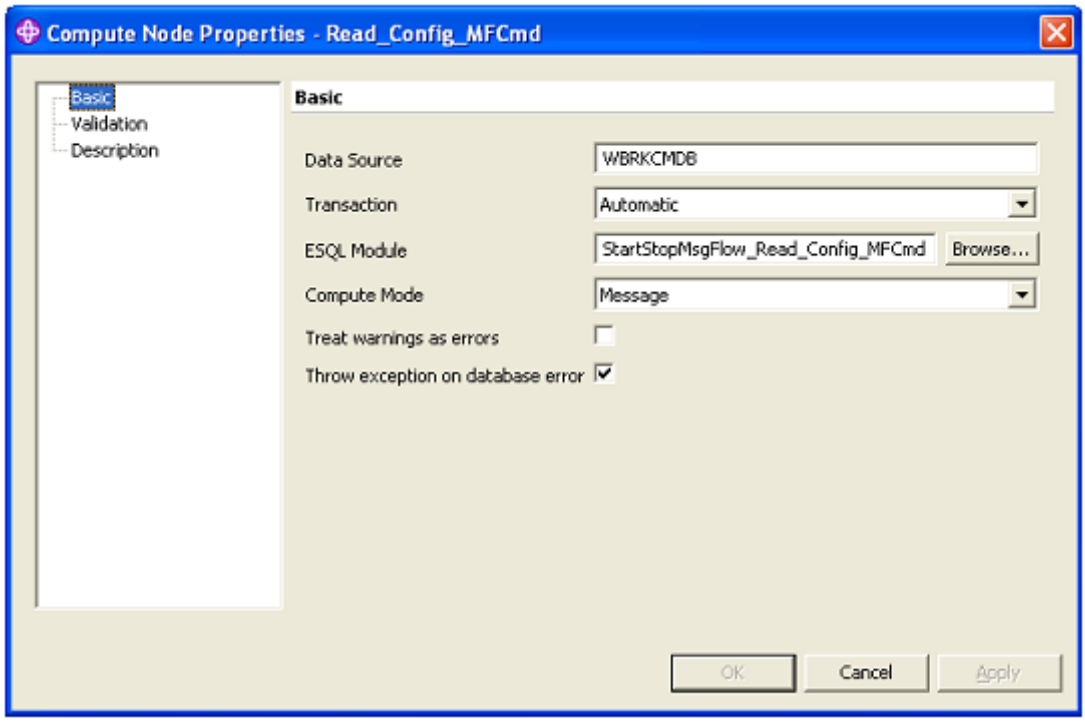

Figure 3: Compute Node Properties

```
EGUUID;

            case InputBody.MFCmd.Action
            when 'Stop' then
                  SET
OutputRoot.XML.Broker.ExecutionGroup.Stop.MessageFlow.(XML.attr)uuid  =
MFUUID;
            when 'Start' then
                  SET
OutputRoot.XML.Broker.ExecutionGroup.Start.MessageFlow.(XML.attr)uuid  =
MFUUID;
            else
                  SET
OutputRoot.XML.Broker.ExecutionGroup.Status.MessageFlow.(XML.attr)uuid  =
MFUUID;
            end case;
```

The command message will then be output to the SYSTEM.BROKER.ADMIN.QUEUE. We can read the reply message from the specified ReplyToQ and interpret the result. A successful stop command will have the following

response from the Broker:

```
<Broker uuid="589ebc8a-ff00-0000-0080-d315c1b06539" label="WBRKBKR"
version="1">
      <OverallCompletionCode result="success">
            <LogEntry catalog="BIPv500" number="2056">
                  <Insert type="string" text="WBRKBKR"/>
                  <Insert type="string" text="589ebc8a-ff00-0000-0080-
d315c1b06539"/>
            </LogEntry>
      </OverallCompletionCode>
      <ExecutionGroupCompletionCode uuid="599ebc8a-ff00-0000-0080-
d315c1b06539" result="success">
            <LogEntry catalog="BIPv500" number="4040">
                  <Insert type="string" text="default"/>
                  <Insert type="string" text="599ebc8a-ff00-0000-0080-
d315c1b06539"/>
            </LogEntry>
      </ExecutionGroupCompletionCode>
      <ExecutionGroup uuid="599ebc8a-ff00-0000-0080-d315c1b06539">
            <StopResponse>
                  <MessageFlow uuid="1708ecb3-0001-0000-0080-
bf6cb11a4390"/>
            </StopResponse>
      </ExecutionGroup>
</Broker>
```

Similarly for the start command:

```
<Broker uuid="589ebc8a-ff00-0000-0080-d315c1b06539" label="WBRKBKR"
version="1">
      <OverallCompletionCode result="success">
            <LogEntry catalog="BIPv500" number="2056">
                  <Insert type="string" text="WBRKBKR"/>
                  <Insert type="string" text="589ebc8a-ff00-0000-0080-
d315c1b06539"/>
            </LogEntry>
      </OverallCompletionCode>
      <ExecutionGroupCompletionCode uuid="599ebc8a-ff00-0000-0080-
d315c1b06539" result="success">
            <LogEntry catalog="BIPv500" number="4040">
                  <Insert type="string" text="default"/>
                  <Insert type="string" text="599ebc8a-ff00-0000-0080-
d315c1b06539"/>
            </LogEntry>
      </ExecutionGroupCompletionCode>
      <ExecutionGroup uuid="599ebc8a-ff00-0000-0080-d315c1b06539">
            <StartResponse>
                  <MessageFlow uuid="1708ecb3-0001-0000-0080-
bf6cb11a4390"/>
```

```
            </StartResponse>
      </ExecutionGroup>
</Broker>
```

As you may have noticed in the case statement of the ESQL, we can also display the status of the message flow by issuing the following command:

```
 <MFCmd>
      <BrokerName>WBRKBKR</BrokerName>
      <EGName>default</EGName>
      <MFName>StopMsgFlow</MFName>
      <Action>Status</Action>
</MFCmd>
```

The reply message will be as follows:

```
<Broker uuid="589ebc8a-ff00-0000-0080-d315c1b06539" label="WBRKBKR"
version="1">
      <OverallCompletionCode result="success">
              <LogEntry catalog="BIPv500" number="2056">
                     <Insert type="string" text="WBRKBKR"/>
                     <Insert type="string" text="589ebc8a-ff00-0000-
0080-d315c1b06539"/>
             </LogEntry>
       </OverallCompletionCode>
       <ExecutionGroupCompletionCode uuid="599ebc8a-ff00-0000-0080-
d315c1b06539" result="success">
              <LogEntry catalog="BIPv500" number="4040">
                     <Insert type="string" text="default"/>
                     <Insert type="string" text="599ebc8a-ff00-0000-
0080-d315c1b06539"/>
              </LogEntry>
        </ExecutionGroupCompletionCode>
        <ExecutionGroup uuid="599ebc8a-ff00-0000-0080-d315c1b06539">
              <StatusResponse>
                 <MessageFlow uuid="0d59c48a-ff00-0000-0080-
bf6cb11a4390" status="Started"/>
            </StatusResponse>
      </ExecutionGroup>
</Broker>
```

Voila! It is this simple.

Of course, we can enhance the message flow to read the reply message from the ReplyToQ and evaluate the response as successful or not, this can be done by extending the message flow as shown in Figure 4.

The compute node copy_MsgID_CorrelID copies the MsgId to the CorrelId such that we can use the MQGet node to read the reply message with the same CorrelId (remember we issue our command as a request. The Broker will copy the MsgId to the CorrelId of the MQMD and generate a new MsgId for the reply message). We will also simulate some delay to give the Broker enough time to execute the command and issue a reply message:

```
-- CALL CopyMessageHeaders();
            CALL CopyEntireMessage();
            SET OutputRoot.MQMD.CorrelId = OutputRoot.MQMD.MsgId;
            -- do some delay here for the response message to appear in
the RESP queue
            DECLARE I,J INTEGER;
            SET I = 1;
            SET J = 5000;
            WHILE I < J DO
                  SET I = I + 1;
            END WHILE;
```

To use the MQGet PlugIn node in WBIMB, refer to *MQ Update* January 2005, issue 67, *How to migrate Plug-In node from WMQI to WBIMB*. On the MQGet Node, we check the

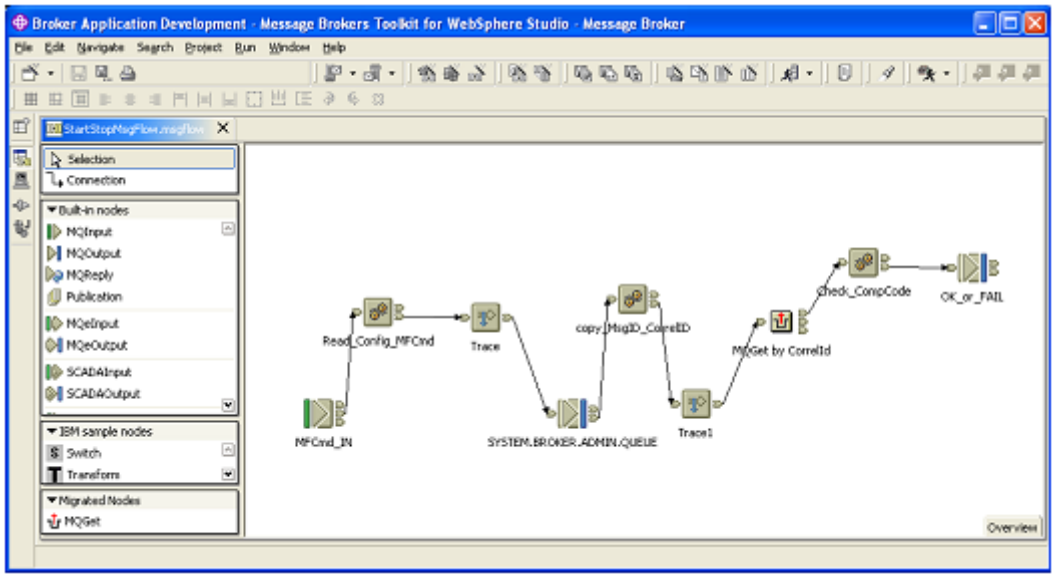

Figure 4: Extended message flow

selectByCorrelId box, as shown in Figure 5.

The compute node Check_CompCode will check the status of the command reply and route the reply message to either the success or failure queue:

```
CALL CopyMessageHeaders();
            -- CALL CopyEntireMessage();
            CREATE LASTCHILD OF Environment.Variables.InRecord DOMAIN
'XML' PARSE
(InputRoot.BLOB.BLOB,InputRoot.MQMD.Encoding,InputRoot.MQMD.CodedCharSetId);
            If
Environment.Variables.InRecord.XML.Broker.OverallCompletionCode.(XML.attr)result
= 'success' then
                  SET
OutputLocalEnvironment.Destination.MQ.DestinationData.queueName  =
'ALEX.MFCMD.OK';
            else
                  SET
OutputLocalEnvironment.Destination.MQ.DestinationData.queueName  =
'ALEX.MFCMD.FAIL';
```

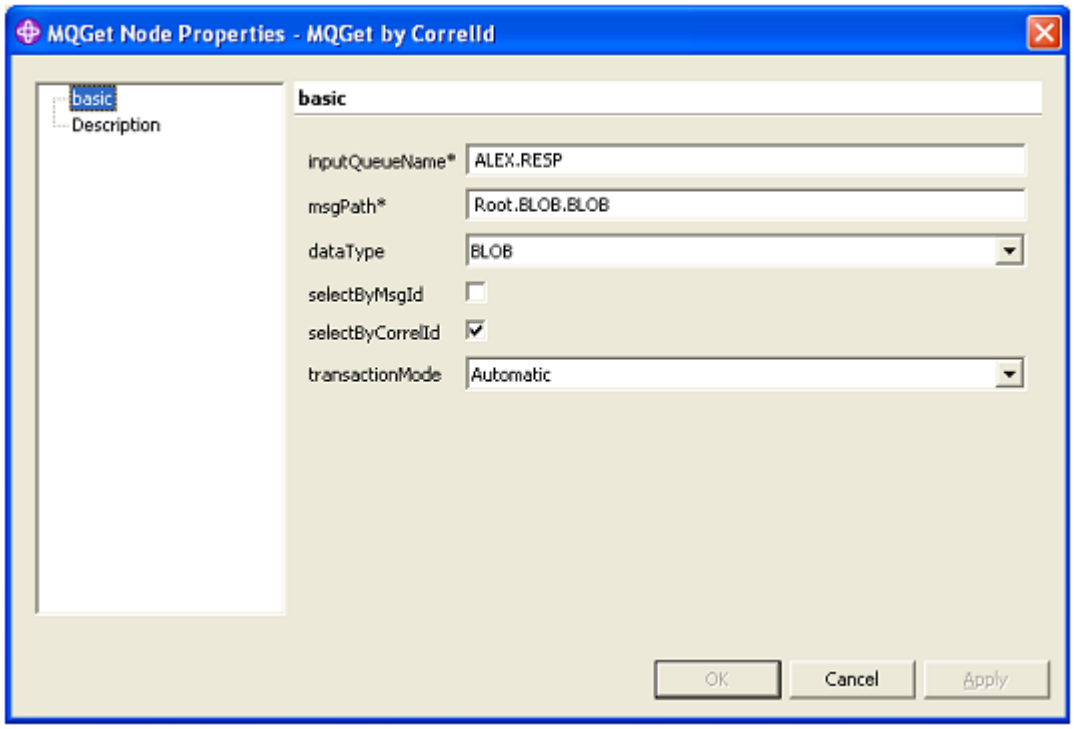

Figure 5: Checking the selectByCorrelId box

```
end if;
-- switch the message body to that read from RESP queue
SET OutputRoot.XML = null;
SET OutputRoot.BLOB.BLOB = InputRoot.BLOB.BLOB;
```

The project provided serves only as an example; you can modify it according to your design requirements.

Alex Au
IT Architect
IBM Global Services (USA)

Why not share your expertise and earn money at the same time? *MQ Update* is looking for program code, JavaScript, REXX EXECs, etc, that experienced users of WebSphere MQ have written to make their life, or the lives of their users, easier. We are also looking for explanatory articles, and hints and tips, from experienced users. We would also like suggestions on how to improve MQ performance.

We will publish your article (after vetting by our expert panel) and send you a cheque, as payment, and two copies of the issue containing the article once it has been published. Articles can be of any length and should be e-mailed to the editor, Trevor Eddolls, at trevore@xephon.com.

A free copy of our *Notes for Contributors*, which includes information about payment rates, is available from our Web site at www.xephon.com/nfc.

WebSphere Studio Asset Analyzer

WebSphere Studio Asset Analyzer is in essence an asset management product. The WebSphere Studio Asset Analyzer consists of a number of components. Its main engine is a DB2 database that contains over 100 tables. It is also a set of parsers that scan source code, on-line resources, and Web components, along with a set of programs that take the scanned information and load it into the database tables. While performing the load, various relationships are formed through the various components. You can then view all of the collected information using a Web browser.

To use WebSphere Studio Asset Analyzer, first of all identify the production resources at your site. You can then let WebSphere Studio Asset Analyzer scan the resources that you want to know more about. MVS resources and non-MVS (Web-based) resources can be scanned. MVS resources consist of source code, JCL, IMS, and CICS region information. These resources can exist in partitioned datasets or in source code change management systems. Scanners for MVS resources execute on MVS itself. Non-MVS resources consist of J2EE applications (including WAR and EAR files), Java source and byte code, XML, HTML, and others. These resources can reside on the appropriate native file system or in Rational ClearCase. The distributed scanners, termed crawlers, for non-MVS resources run on Microsoft Windows machines.

After WebSphere Studio Asset Analyzer has stored the information about these resources in the database, the information can be shared across an enterprise by all of your application development teams. As a WebSphere application, WebSphere Studio Asset Analyzer uses JavaServer Pages (JSP), servlets, and HTML to display information in a Web browser for you. This interface keeps the details of the database queries hidden from view, allowing you to concentrate

on the information you are looking for and freeing you from the task of writing detailed queries to obtain it. When you view the information in the database, the pages displayed in a Web browser reflect the logical organization of the various applications in your company. Through a series of links built on the relationships between the components that WebSphere Studio Asset Analyzer discovered during the scan, you can drill down from the highest level of your application to a single data element at the very lowest level of the application. Visual representations show how your programs, data files, batch jobs, and transactions are related. WebSphere Studio Asset Analyzer should help your application development teams to:

- Understand components and their relationships.
- Analyse the impact of a proposed change.
- Scope and develop project plans.
- Gather connector information for MVS programs.
- Extract business logic from existing code.

The product is aimed at members of the following groups querying the database to obtain information that can help them do a better job: project managers, programmer analysts, application developers, and quality assurance testers.

They can use WebSphere Studio Asset Analyzer in any phase of the application development process. For companies that are looking at expansion of existing applications to a Web-based audience, either internally or externally, WebSphere Studio Asset Analyzer provides the ability to fully explore the inter-relationships between components in an application so that application development project leaders or group managers can prepare project plans and make the appropriate assignment of resources.

Application programmers can then use the information that was gathered, initially by their team leaders, to manage their workload. They can complete their assignments more quickly because of the easy way in which WebSphere Studio Asset

Analyzer enables them to drill down to understand the details of their application programs.

As you would expect, WebSphere Studio Asset Analyzer requires a number of other program products to support it. WebSphere Studio Asset Analyzer has four core functions that can be accessed using a Web browser. They are:

- Inventory collection.
- Application exploration with change impact analysis.
- Connector building.
- Help.

Inventory collection forms the foundation of WebSphere Studio Asset Analyzer. You identify all of the production application resources at your site. You can then ask WebSphere Studio Asset Analyzer to process your site's application code and load it. During inventory collection, WebSphere Studio Asset Analyzer's parsers scan the resources that you specify.

The information is then stored in a DB2 database. WebSphere Studio Asset Analyzer collects inventory information for both mainframe and distributed applications. Inventory is collected for OS/390 or z/OS components by examining source code from partitioned datasets (either PDS or PDSE), IBM Source Control Library Manager (SCLM), or ChangeMan. The types of source code that can be scanned are COBOL, PL/I, and Assembler, including copybooks and macros, JCL, procs, and control cards, CICS and IMS on-line regions, transactions, and subsystems. The mainframe scanners execute as batch jobs, which are submitted from either a Web browser or an ISPF session.

Inventory for Web-based components is collected from filesystems, including the hierarchical file system of Unix System Services, any accessible WebDAV server, or Rational ClearCase. The distributed scanners are crawlers that run on Windows workstations.

As components are scanned into the database, WebSphere Studio Asset Analyzer establishes the logical connections that exist among them.

For example, copybooks are related to the programs that use them. In addition, programs are related to the batch jobs or on-line transactions that execute them. Ultimately, application data files are linked to both.

By using the Explore function to view application components and their relationships after they have been loaded into the database, you can start at a very high level to obtain an overview of the general characteristics of your application. You can also drill down into the details of specific components. This can go down as far as the data element (that is a WORKING-STORAGE field) level of an application.

You can search for components by looking for a specific one or by selecting one from a list of components. If you need to, you can search for components by their name, application name, project name, or site. You can then use WebSphere Studio Asset Analyzer's Explore function to perform a change impact analysis. You can determine what components are affected based on changes to field declarations, a section of program source code, an entry point signature, etc. You can create projects within WebSphere Studio Asset Analyzer to save the results of your exploration.

By having the ability to explore how the components of your application fit together, you can gain a better understanding of how you can maintain or improve your application. The ease with which you can generate a list of components that are affected by a change can reduce your project determination time and resulting maintenance or new development costs.

You can also use WebSphere Studio Asset Analyzer to obtain and build connector information for CICS and IMS programs. It is easy to gather input and output data structures of transactions. For each transaction, WebSphere Studio Asset Analyzer generates a summary report, a COBOL copybook

containing the input/output data structures, and then allows you to format this information into a form that you can import into a connector-building tool, such as WebSphere Studio Application Developer. Connectors provide the interface through which data moves between a Web interface and your business logic on MVS. Your business logic can be in applications such as COBOL, PL/I, or Assembler. By using connectors, you can enable authorized users on the Web to obtain their account data from a database on the host. Frequently, connectors drive existing transactions and provide input and output data to them. To build those connectors, you need an easy way to find useful transactions, and to quickly identify the input and output data structures of those transactions. WebSphere Studio Asset Analyzer helps you quickly gather this information and puts it in a form that you can import into a connector-building tool such as WebSphere Studio Application Developer Integration Edition. For each transaction, WebSphere Studio Asset Analyzer generates a summary report and a COBOL copybook containing the transaction's input/output data structures.

WebSphere Studio Asset Analyzer also provides you with extensive on-line help information. There is a product overview, a reference, tutorials, set-up information, PDF manuals, and a variety of general information such as a glossary.

There are several unique words and phrases associated with WebSphere Studio Asset Analyzer that you need to be familiar with. The four I will discuss briefly are:

- Site.
- Application.
- Concatenation set.
- Asset or artefact, as it is sometimes termed.

A site is the focal point or name by which an organization wants to be known to accommodate that organization's production application source code.

During inventory collection, WebSphere Studio Asset Analyzer associates the site with each of the components that are scanned into the database. It then organizes the components based on a hierarchy, which is built as a tree structure below the declared site.

WebSphere Studio Asset Analyzer on-line help defines an application as, ‘a user-defined grouping of components. This can have CICS transactions, IMS transactions, and members as part of an application.’

An application is another level that WebSphere Studio Asset Analyzer builds as part of its hierarchy. An application name will exist below the site name in the logical hierarchy. The application entry represents any grouping of source, JCL, CICS, and other components that you choose. It is possible to build application names in WebSphere Studio Asset Analyzer that include cross-application groupings as needed by your site’s requirements.

WebSphere Studio Asset Analyzer gives you the ability to define components and associate them with these corporate entities either when you load the database or afterwards. WebSphere Studio Asset Analyzer even allows you to create hierarchies of applications.

A concatenation set is an ordered list of libraries to be searched to resolve references to included source. An example would be JCL procs, Assembler macros, or COBOL copybooks. During inventory collection, WebSphere Studio Asset Analyzer attempts to resolve the location of each of the included components found in the main programs. A concatenation set is the equivalent of the datasets used in a SYSLIB DD in a batch compile or specified in the ORDER parameter of a JCLLIB statement. In a similar manner, WebSphere Studio Asset Analyzer uses a top-down search through the libraries defined in the concatenation set to determine the appropriate member to use.

WebSphere Studio Asset Analyzer's online help defines an asset as, 'a programming asset, such as a file, an object, documentation, or code'.

They include, but are not limited to:

- Programs.
- Run units – batch jobs.
- CICS transactions.
- Datasets.
- Data stores.
- DDnames.

WebSphere Studio Asset Analyzer even considers applications, concatenation sets, and projects to be assets.

When you take an inventory of distributed assets, tools like scanners and analysers identify, analyse, and classify components such as Java and HTML files and then store information about these components in a database. WebSphere Studio Asset Analyzer uses scanners to find distributed assets on enterprise servers where code components are maintained:

- Filesystems including local and remote Windows filesystems, the hierarchical filesystem of Unix System Services, and mapped AIX filesystems.
- Configuration management systems including Rational ClearCase, PVCS, CVS, and CMVC.
- WebSphere Application Server 4.0 Advanced Edition installations.

A scanner will start searching at predefined locations on an enterprise server and scan recursively from these locations. These initial start points are termed 'scanning roots'. The scan process can be started by user command or as part of a schedule set by the administrator. WebSphere Studio Asset

Analyzer provides over 40 analysers that understand the structure and contents of distributed assets, including Java sources, Java bytecode, C++, JavaServer Pages (JSP) files, HTML, XML, and text files. These domain-specific analysers examine the structure of physical components, files for example, and logical components such as Java classes to determine various pieces of information. Some examples are:

- Resource-specific semantic attributes.
- Textual information.
- Keywords and relations among the components.
- The Java analyser extracts semantic features like class name, imports, package, and methods from Java components.

These are then stored in attributes such as the package attribute, so that you can query about all classes that are part of a specific Java package. There will also be stored the free-text attribute, which has the textual information delimited by spaces that you might want to search for. These words are indexed so that you can quickly retrieve resources containing them.

After the structure and contents of assets have been analysed, the assets are organized into categories such as containers, Java assets, or Web assets. Users can explore the categories to discover what is available in the enterprise server and to find reuse candidates and code examples that are relevant to their work.

For WebSphere Studio Asset Analyzer you need certain software on the host, certain software on each client, and, if you use the distributed assets support in WebSphere Studio Asset Analyzer, certain software on a Windows server.

To install and use WebSphere Studio Asset Analyzer, you must have the following software installed on your host with WebSphere Studio Asset Analyzer. You need z/OS Version 1

Release 1 or later, or OS/390 Version 2 Release 10 or later. In either case, you must install the optional feature Security Server or an equivalent security product before people use WebSphere Studio Asset Analyzer.

You also need DB2 for OS/390 Version 7 or later, with its optional feature Net.Data for OS/390 Version 2.2 with PUT0112 or higher applied. If you use Version 7.1 of DB2 and Net.Data 7.1, you need to apply PTF UQ62108. You must also install and configure the JDBC application support by using the RRS attachment facility (RRSAF). Or you need DB2 for z/OS Version 8 or later.

You also need one of the following, but remember that this software is optional if your install configuration includes WebSphere Application Server for Windows on your Windows server:

1 WebSphere Application Server for OS/390 Version V4.0.0, including Java Development Kit V1R3 with PTF UQ62958.

2 DB2 access using JDBC WebSphere Application Server for z/OS V5.0.0 enabled.

If you plan to use the IMS support in WebSphere Studio Asset Analyzer, the following software must be installed on the host system. You need IMS/ESA Database Manager Version 5 or later, and IMS/ESA Transaction Manager Version 5 or later. You also need one of the following:

- IMS Library Integrity Utilities for z/OS V1 or later.
- IMS Library Management Utilities Version 1 or later.

If you intend to use the CICS support in WebSphere Studio Asset Analyzer, CICS Transaction Server Version 1 Release 3 or later with PTF UQ48250 must be installed on the host system.

If you are going to use the distributed assets support in WebSphere Studio Asset Analyzer then you need a number of pieces of software installed on your Windows server as well.

You must have one of the following versions of Windows:

- Microsoft Windows XP Professional with Service Pack 1 or later.
- Windows 2000 Professional.
- Windows NT Version 4 with SP4 or later.

You also need:

- The IBM DB2 Universal Database for Windows Runtime Client Version V7 or V8 as required by WebSphere Application Server. The level of this client should be the same as that of your host system. IBM recommends that you use the V8 client with the V8 database. You need DB2 Connect Personal Edition.
- The Java Runtime Environment (JRE) V1.3.1 or later.

If you want to run WebSphere Studio Asset Analyzer with a configuration that does not require WebSphere Application Server on MVS, you must have the following software installed as well – WebSphere Application Server Advanced Edition V4.0.3 or later, or WebSphere Application Server Enterprise V5.0, fixpack 3 or later, or IBM HTTP Server.

You must have Microsoft Internet Explorer 5.5 or later installed on your workstation to access WebSphere Studio Asset Analyzer. ActiveX must be enabled to support scalable vector graphics.

Below is a list of useful publications, and their publication numbers, from IBM that relate to the WebSphere Studio Asset Analyzer :

- *Brochure* – GC18-9126-00
- *Program Directory* – GI10-8546-00
- *Getting Started* – GC18-9290-01
- *Quick Tour* – GC18-9291-01
- *Taking an Inventory* – SC18-9292-01

- *Messages and Codes* – SC31-8959-00
- *Size and Implement Code Changes* – SC18-9298-00
- *Create Connector Information for a COBOL CICS Application* – SC18-9295-01
- *Create Connector Information for a COBOL IMS Application* – SC18-9296-01
- *Searching Distributed Assets for Reuse* – SC18-9297-01
- *Configuration and Migration Guide* – SC18-9108-02
- *License Information* – GC18-7913-00.

John Bradley
Systems Programmer
Meerkat Computer Services (UK)

Optimizing a WebSphere MQ Workflow environment

IBM's WebSphere MQ Workflow product (currently at Version 3.5) is designed to support long-running business process workflows as they interact with systems and people. IBM claims that it allows users to bring systems and people into a managed process integration environment for EAI, B2Bi, BAM, and BPM solutions, built to a service-oriented architecture based on standards. It runs on z/OS, AIX, HP-UX, Sun Solaris, and Windows platforms.

Something that is associated with business integration methodologies, process automation, and business-level monitoring in real-time is going to turn into a cumbersome beast – and it's quite likely that there are going to be performance issues. This article looks at just a few areas that can be addressed to, hopefully, improve performance.

Performance issues usually come down to two simple areas – did you design properly in the first place, and are you using it now in the way it was designed?

The initial design is very important. Moving away from the initial test system to a live system really involves spreading things out. If files and logs are on the same physical disk, there will be contention on the disk and performance will suffer. If the logs fill up, Workflow itself can be abended. In fact, it's best to install Workflow on a different node (or filesystem) from the one WebSphere is on.

Where a large number of clients connect to the default queue manager it is better to create a separate queue manager to act as a host to the clients. This queue manager then passes the requests to the Workflow queue manager.

Should you use blocks or subprocesses? A subprocess is a piece of executable code designed to be called from other processes. In general, blocks are better in performance terms than subprocesses.

What database are you using? If it's DB2, then it needs to be optimized. That means running runstats and rebinds regularly. If you are using Oracle, perform DBMS schema analysis on a regular basis and ensure that there is enough space for table spaces.

Once everything is up and running, it is important to monitor what is happening to maintain optimum performance.

There is a performance capacity planning measurement called the Basic Workflow Unit (BWU). IBM provides a SupportPac to help users understand how expensive their source is. This allows users to make changes in order to reduce this value – and then test their change.

Associated with this is the fact that the BWU calculation allows users to calculate their Flow Definition Language (FDL) costs.

If performance is below what you anticipated, MQSeries tracing can be turned on.

As well as tracing, you can use logging to identify problems. Your choices are circular logging or linear logging. With circular logging, the old logs are re-used after a period of time. Linear logging just keeps writing the log files. This means that space calculations must be performed to ensure that the log space is large enough.

Using WMQ monitoring means that you are informed when important queues (eg EXEXMLINPUTQ) are full or channels are in a state other than active. It's also possible to monitor the MQ processes, the listeners, and queue managers. This helps with error handling.

Databases can also be monitored. This will identify when they are filling up. Database processes, and database instances, can also be monitored.

WAS (WebSphere Application Server) can be monitored too – again to identify errors and recover from them.

Auditing can also be an issue. With FULL auditing, it can be better to write the information to a database rather than MQ.

There are, of course, other areas to consider, but these provide a good place to start.

Independent Consultant (UK)

MQ news

Systinet has announced the release of Systinet Server for IBM WebSphere MQ, the latest addition to the company's standards-based Web services enablement platform for Java, C/C++, and Message-Oriented Middleware (MOM) applications.

The Systinet Server will interoperate WebSphere MQ applications with Web services without the need to install any proprietary middleware hardware or software.

Systinet Server for IBM WebSphere MQ adds an abstraction layer that preserves all of the functionality of the MQ application while providing unique support for important Web service standards, including WS-ReliableMessaging, WS-Eventing, WS-Addressing, and WS-Security. Customers also have the option to incorporate SOA governance and business services life-cycle management with the latest release of the Systinet Registry.

The latest version also features a Content-Based Routing (CBR) Web service that creates routing scenarios based on message content.

For further information contact:
URL: www.systinet.com/news/in_the_news/article&id_ele=70.

* * *

Entrust has announced Version 7.0 of TruePass for WebSphere. The Entrust 7.0 system, a component of the Entrust Secure Identity Management Solution, provides authentication, digital signatures, and encryption for WebSphere Application Server 5.0 and WebSphere Portal Server 5.0.

The Entrust Secure Identity Management Solution provides a tightly-integrated security solution for identity and access management.

The Entrust TruePass software is a zero-footprint solution for strongly authenticating users and protecting transactions with digital signatures and encryption.

For further information contact:
URL: www.entrust.com/identity_management/index.htm.

* * *

IBM has renamed DB2 Information Integrator products. In future they will be known as IBM WebSphere Information Integrator.

There are no product changes: ie no technology change, nor changes in product prerequisites, nor any changes in product packaging, licensing, or pricing.

DB2 Information Integrator Content Edition becomes WebSphere Information Integrator Content Edition; DB2 Information Integrator Classic Federation becomes WebSphere Information Integrator Classic Federation, and so on.

For further information contact your local IBM representative.

* * *