54

MQ update

December 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 £100 ($160) per 1000 words and £50 ($80) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £20 ($32) 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.

Upgrading to WMQ V5.3 in an HA configuration

SCOPE

This article discusses WMQ (WebSphere MQ) running on Unix platforms in an HA (high availability) cluster environment, using one of the IBM HA SupportPacs listed in Figure 1. The article also describes the procedures required to upgrade from one version of WMQ to another and, specifically, provides guidance on how to upgrade to WMQ V5.3.

The major part of the procedure is the upgrade of the installed code and update of the queue manager objects.

BACKGROUND

During the upgrading of a cluster node to WMQ V5.3, the code will be updated during the install procedure, which can only be performed when there are no queue managers running on the node. To minimize downtime the queue managers can be kept running on other nodes in the HA cluster.

The queue manager objects are updated at runtime when the

Operating system	Cluster software	SupportPac	URL
AIX	HACMP	MC63	*http://www.ibm.com/software/integration/support/supportpacs/individual/mc63.html*
Solaris	Sun Cluster V2.*x*	MC69	*http://www.ibm.com/software/integration/support/supportpacs/individual/mc69.html*
	Veritas Cluster Server	MC6A	*http://www.ibm.com/software/integration/support/supportpacs/individual/mc6a.html*
HP-UX	ServiceGuard	MC6B	*http://www.ibm.com/software/integration/support/supportpacs/individual/mc6b.html*

Figure 1: HA SupportPacs

queue manager is next started. We'll return to these points in the following example.

There is no procedure for 'downgrading' an abandoned upgrade installation. The only way to revert to the earlier version is to use a back-up taken before the upgrade was attempted.

BASIC EXAMPLE

The procedure for performing an upgrade will be described for a single queue manager running in a two-node cluster. More advanced configurations will be discussed briefly in a later section.

The two nodes are called node A and node B. There is one queue manager, QM1. The queue manager was created and configured using the **hacrtmqm** and **halinkmqm** commands from the relevant HA SupportPac. The queue manager's log file and queue files are stored on shared disks, which are owned by the node that is running the queue manager. The other node is acting as a standby in case of failure. For the purpose of this example, suppose that node A is running the queue manager and node B is the standby node.

Before performing any of the following steps it is recommended that you first take a back-up.

Step one

Alter the cluster configuration so that the queue manager cannot be automatically failed over during the following procedure. Don't stop the queue manager, just alter the configuration of the cluster so that the group, logical host, or service that contains the queue manager will not be moved by the cluster framework. The means for achieving this depend on the clustering software you are using. As soon as the upgrade of the cluster is complete, automatic failover will be re-enabled.

Step two

Upgrade node B (the standby node). The queue manager is not running on this node so it is OK to upgrade the product code. The

objects in the log/queue files will not be needed at this stage because they are updated only when the queue manager is next started.

Step three

Perform a manually-initiated failover of the queue manager from node A to node B. When the queue manager is started on node B the queue manager objects are updated to the new version. Verify that the queue manager works correctly on node B.

Step four

Now upgrade WMQ on node A, which is acting as standby node. This will only update the product code; the queue manager is running on node B.

Step five

Perform a manually-initiated failover of the queue manager from node B to node A. When the queue manager is started on node A it will not need to update any queue manager objects to the new version because they were updated during step three. Verify that the queue manager works correctly on node A.

Step six

Finally, re-enable automatic failover of the queue manager.

MORE ADVANCED CONFIGURATIONS

Configurations with more nodes or more queue managers can be treated as extensions of this basic example.

The case of a queue manager running in an HA cluster with more than two nodes is similar to the above procedure with the difference that all standby nodes can be upgraded without a failover of the queue manager. Then, following one failover of the queue manager, the remaining node can be upgraded.

The case of an HA cluster that is running more than one queue

manager optionally, on different nodes in an active-active or mutual-takeover configuration, is also a straightforward extension of the above procedure. The additional step required is to begin by initiating failover(s) of one or more queue managers so that there is one or more node that is not running any queue managers. The idle node(s) can then be upgraded and the queue managers failed-over so as to vacate a different node. The nodes can be vacated and upgraded in turn until the whole HA cluster is running WMQ V5.3.

DOWNTIME

With any size of HA cluster there will necessarily be a brief period of downtime during the upgrade procedure whilst the queue manager is failed-over to the first node to be upgraded. This is a minimum requirement because the restart of the queue manager on a node with WMQ V5.3 is necessary in order to update the queue manager objects.

Although a small amount of additional downtime will be needed for each additional failover it is strongly recommended that the queue manager be failed-over to each node that is upgraded to ensure that it runs correctly on that node.

With a two-node cluster, during the upgrade procedure there is a small window of risk whilst automatic failover is disabled. A failure of the node running a queue manager during the upgrade of one of the other nodes could result in a period of unplanned downtime.

With an HA cluster containing more than two nodes this window of risk can be eliminated by restricting the HA configuration to the use of (at least) a pair of nodes for running the queue managers and upgrading the remaining node(s) to WMQ V5.3. Next, restrict the HA configuration to run the queue manager on a subset of the original nodes and one or more of the upgraded node(s). A failure during this period would cause a failover of the queue manager(s) to an upgraded node, which is OK. It is, therefore, safe to upgrade the idle non-upgraded node(s). Then, failover the queue manager(s) to any of the upgraded node(s) and restrict the HA configuration

to running the queue managers only on upgraded nodes whilst the remaining nodes are upgraded.

ADDITIONAL INFORMATION

When the queue manager was created and configured for HA operation the **halinkmqm** script will have been used on each node to move some control directories to a different filesystem and set up symbolic links from the original locations. Note that **halinkmqm** is invoked from within **hacrtmqm** so this refers to all nodes, not just the standby nodes. The list of directories can be seen by examining the **halinkmqm** script. WMQ V5.3 introduces an additional directory, qmpersist, within the @ipcc directory tree. This directory also needs to be moved and symbolically linked in the same way as the directories already set up by the **halinkmqm** script. You can do this by hand but if you intend to create and configure additional queue managers after the upgrade it would be worth editing the **halinkmqm** script to include the qmpersist directory.

Another point to watch is that, pre-CSD03, the WMQ V5.3 Object Authority Manager (OAM) will reject an authorization check during queue manager startup because of the symbolic link from /var/mqm/qmgrs/<qmname> to the HA filesystem directory for the queue manager. This can be worked around by removing the symbolic link. The behaviour of the OAM is fixed in WMQ V5.3 CSD03.

Graham Wallis
IBM Hursley (UK)

Using WMQ as a transport mechanism: case study

This article stems from an implementation at one of our 'life sciences' clients and is intended to help IT architects and technical specialists understand the use of WMQ as a transport mechanism.

In this particular case the implementation carried out was between WebSphere Commerce Suite (WCS) and an existing ERP (Enterprise Resource Planning) system, J D Edwards OneWorld (OneWorld).

SCENARIO OVERVIEW

The client is an organ donor organization located in southeastern USA. The organization supplies human organs, tissues, and bones to various hospitals and other organ banks across North America and Europe.

The company wanted to take advantage of the global connectivity offered by the Internet to give customers a new way to buy products and check order status while augmenting the manual sales processes with direct electronic sales to its business customers. The project was also aimed at improving the existing purchase processes, which required a lot of manual communication between customers' quality control departments and the company's own. WMQ connects the Web storefront to the ERP systems.

BUSINESS FUNCTIONS PROVIDED

The organ donor company wanted to provide online transactions for registered and qualified customers only. Considering the nature of the sales there are various criteria that are applied to customers. Major business functions are:

- Logon.
- Browse and search for the organ/tissue/bone.
- Checkout (place order).
- Check order status.
- Reorder items from previous orders.
- Priority orders.
- Change user profiles and synchronize information with OneWorld.

WMQ IMPLEMENTATION

The WMQ adapters implemented by WebSphere Commerce Suite (WCS) and JDE enable the business integration of the storefront and the back-end ERP system. The integration of both applications is through message exchange over the WMQ queueing infrastructure. All transported messages are in the form of Extensible Markup Language (XML) documents. The Integration Node component transforms the XML messages to and from the JDE and WCS formats. To implement the end-to-end WMQ message integration in this scenario we performed the following high-level tasks:

- WMQ was configured on the WCS server node and the ERP system node.
- The Commerce Suite WMQ Adapter was configured.
- The OneWorld Adapter for WMQ was configured.

Products used

The following products were used to implement WMQ in this project:

- WMQ for AS/400 V5.1.
- WebSphere Commerce Suite V4.1 for AS/400. The Commerce Suite WMQ Adapter is included in the WCS V4.1 product.
- OneWorld Adapter for WMQ.

WMQ configuration summary

The following tasks were performed on COMMSVR and ERPSVR to set up WMQ configuration objects in this scenario:

- WMQ for AS/400 V5.1 was installed.
- We created a default queue manager.
- We created a transmission queue and local and remote queues.
- We created communication channels.

- We verified end-to-end message delivery in both directions.

To create the WMQ configuration we used the **STRMQMMQSC** command, which uses as input a source file member containing the WMQ commands (MQSC) to be processed.

Commerce Suite WMQ Adapter configuration summary

The Commerce Suite WMQ Adapter is a component of WCS V4.1, which is installed with the main product suite. It enables integration with a back-end application using WMQ as the transport mechanism to exchange messages. The Commerce Suite WMQ Adapter works with a set of predefined messages that it sends and receives using WMQ.

To enable the Commerce Suite WMQ Adapter in our scenario you need to perform the following steps:

- Install and configure WMQ. The adapter requires the following WMQ objects:
  - queue manager (COMMSVR)
  - inbound queue (JDE.TO.WCS)
  - outbound queue (WCS.TO. JDE)
  - error queue (ERROR.Q).
- Start the WMQ server before you start the WCS instance. Grant the WCS instance user profile authority to the queue manager and queues.
- Use the Grant MQM Object Authority (**GRTMQMAUT**) command.
- Enable the Commerce Suite WMQ Adapter by running the WMQ Adapter (**CFGNETCMQ**) command 'Start the QMQM subsystem' STRSBS SBSD (QMQM/QMQM).
  - before starting the Integration Node programs start the queue manager, the MQM channels, and the MQM listener

- instruct WMQ Adapter (**CFGNETCMQ**)
- type choices, press Enter
- instance name: LifeNet Character value
- port: 18000
- host name: www.lifenet.org
- inbound queue: JDE.TO.WCS
- outbound queue: WCS.TO.JDE
- error queue: ERROR.Q
- queue manager: COMM.SERVER
- user-ID: gtownuser
- the **CFGNETCMQ** command creates the file /qibm/userdata/commercesuite/instance/lifenet/config/ncei_ece.ini
- ncei_ece.ini is the Commerce Suite WMQ Adapter configuration file. The following lines in the file pertain to the WMQ setup in your scenario:

```
NC_MQ_INBOUND JDE.TO.WCS
NC_MQ_OUTBOUND WCS.TO.JDE
NC_MQ_INERROR ERROR.Q
NC_MQ_QMANAGER COMM.SERVER
NC_MQ_BRIDGE_USERID gtownuser
```

- Enable outbound messages that use the XML format.

Configure the EXT_ORD_PROC process task to use the ECEExtXMLOrdProc overridable function. The user-ID referred to in the **CFGNETCMQ** command is a WCS user and must be specified in the Site Administrator group within the Access Group Assignment form of the Commerce Suite Administrator tool.

OneWorld Adapter for WMQ configuration summary

For information on how to install and configure the OneWorld Adapter for WMQ refer to the following JDE OneWorld documentation.

- *Asynchronous Messaging Adapter Programmer's Guide.*
- *OneWorld Adapter for WMQ Installation Guide (AS/400 Systems)*.
- *OneWorld Adapter for WMQ Configuration Guide (AS/400 Systems)*.
- *MQSI Download Instructions for AS/400.*

The OneWorld Adapter for WMQ accepts input and produces output by reading and writing to WMQ queues. After WMQ has been installed you must create specific queues for the OneWorld Adapter on which WMQ can operate. You must specify the names of these queues in the jde.ini file on the OneWorld server so that the OneWorld Adapter for WMQ can find them. When you install the WMQ Adapter you are asked to create several message queues. The following queues must be defined in the OneWorld jde.ini file:

- Inbound (INBOUND.Q).
- Outbound (OUTBOUND.Q).
- Default response (ERROR.Q).

The queue names in the jde.ini file must correspond to the queue names on the server. The following parameters in the MQSI section of the jde.ini file pertain to the WMQ setup in our scenario:

```
[MQSI]
QMGRName=JDE _QMGR
QInboundName=INBOUND.Q
QOutboundName=OUTBOUND.Q
QErrorName=ERROR.Q
TimeoutWaitInterval=15
MaxBufferLength=10240
CreateHeader=NO
AppGroup=NNJDE
JDEOrderStatusCode=JDESOOUT
JDECustomerCode=JDEAB
```

QErrorName queue is the default queue for any response messages. If no specific queue is defined for a response message or if the destination queue for a response message cannot be

located the message is sent to the queue defined by QErrorName in jde.ini. In addition to the three queues required by the OneWorld Adapter for WMQ we created the following:

- POSUCCESS.Q – which receives messages with return code information from the OrderEntry message.

  Be sure to specify CreateHeader=NO. Specify CreateHeader=YES when the Adapter is used with WMQ Integrator, which is not the case in this scenario.

- SUCCESS.Q – which receives messages that are responses to the price quote request from the getrtprice overridable function.

SUCCESS.Q, ERROR.Q, and any other queues are named and created by the users. They are not defined in the jde.ini. file. The user can specify these queue names as the destination queue for response messages in the request message. If the OneWorld Adapter for WMQ physically finds these queues the Adapter sends the response message to the user-specified queues, otherwise the Adapter sends the response message to the default queue (QErrorName).

Recommendations

During the configuration and operation of WMQ we found it useful to remember the points listed below.

- The names of the queues and channels in WMQ are case-sensitive.
- Use the 'Start MQM WMQ Commands' (**STRMQMMQSC**) command and associated source file with WMQ commands to set up and document the initial WMQ configuration.
- Use the 'Work with Queue Managers' (**WRKMQM**) command as your primary interface for operating WMQ objects.
- For a complete list of all the WMQ CL commands use the CMDMQMmenu. You can invoke this menu by entering the following command from any command line:

```
GO MENU(CMDMQM).
```

- The WMQ subsystem, queue manager, and channels must be started before you start the WCS instance. Be sure to restart the WCS instance whenever you end and start the queue manager.
- The WMQ listener job must be started for the Integration Node program to work.
- The instance user profile must be authorized to the WMQ queues. Queue managerUser profiles for users that require WMQ administrator rights must be part of the QMQMADM group profile.
- Use the WMQ-supplied sample programs to put test messages to a queue and get messages from a queue.

REQUIRED PRODUCTS

The following products are required to implement the Integration Node in this scenario:

- WebSphere Application Server V3.02 Advanced Edition.
- LotusXSL, which can be downloaded from *http://www.alphaworks.ibm.com/tech/LotusXSL*. I recommend you obtain the latest version. We used V1.0.1.
- AS/400 Developer Kit for Java (5769-JV1): Base and option 3 (Java Developer Kit 1.2).
- WMQ for AS/400 V5.1.
- MA87 WMQ Client for Java on the AS/400 platform. You can download this package from *http://www.ibm.com/software/ts/mqseries/txppacs/txpm2.html#cat3*.
- QShell Interpreter.

IMPLEMENTING THE INTEGRATION NODE

WCS and OneWorld define a set of standard transactions for common business functions, which are:

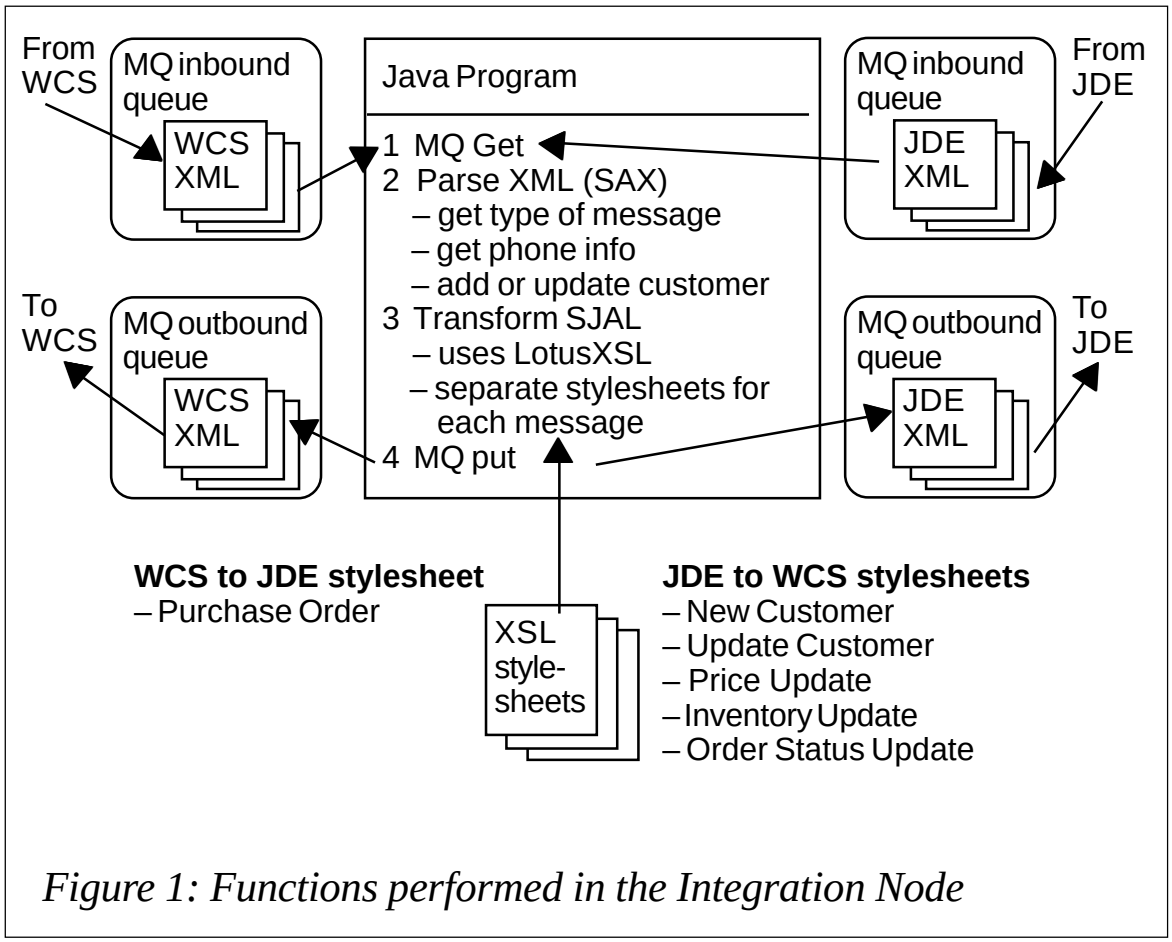

Figure 1: Functions performed in the Integration Node

- Customer New and Update.
- Product Price Update.
- Product Quantity Update.
- Order Status.
- Order Create.

The first four messages are outbound messages generated by OneWorld. Order Create is the only outbound WCS message. Both WCS and OneWorld are capable of generating XML-formatted messages for these business functions.

The MQ Adapters implemented by both products monitor an inbound queue for request and reply messages, pass the message to the application to perform the requested service, and place the results in an outbound queue. WCS and OneWorld have their own

XML message formats. The Integration Node reads the message from a queue, processes the message to transform it, and writes the message to an outbound queue.

The Integration Node programs perform the following principal tasks:

- Read messages from inbound queues and write messages to outbound queues after being transformed. To perform this task use the WMQ for AS/400 and WMQ Client for Java on AS/400.
- Parse the inbound message to determine the business transaction and extract specific parameters that appear in different formats in the WCS and OneWorld XML messages. To perform this task use the Simple API for XML (SAX) parser, which is included in the LotusXSL package. SAX is a standard interface for event-based XML parsing and was developed collaboratively by members of the XML-DEV mailing list, which is currently hosted by the Organization for the Advancement of Structured Information Standards (OASIS).
- Transform the OneWorld XML format to the corresponding WCS XML format and *vice versa*. To perform this function develop XSL stylesheets and use the LotusXSL processor provided in the LotusXSL package.

Figure 1 shows the main functions performed in the Integration Node in the project.

JAVA CLASSES AND WMQ QUEUES

A Java application drives the Integration Node. You can download the MA87 WMQ Client for Java on AS/400 package from *http://www-4.ibm.com/software/ts/mqseries/txppacs/txpm2.html*. For information on WMQ classes for Java refer to the manual *WMQ – Using Java, SC34-5456*.

The Java classes we developed for the client that made up the Integration Node application are:

- *IntegrationNode.java*. The main method is called by the CL program IntNode in library eBit3. It is responsible for setting up

the initial environment and starting the threads that monitor queues for incoming messages.

- *JDE2NCIntegrationNodeHandler.java*. This class is used by IntegrationNode. It is responsible for the following functions:
  - monitoring the FROM.ERP.SERVER queue for OneWorld XML messages
  - passing the message to JdeMessage for transformation
  - writing the transformed message (in WCS XML format) to the JDE.TO.WCS queue.
- *NC2JDEIntegrationNodeHandler.java*. This class is used by IntegrationNode. It is responsible for the following functions:
  - monitoring the queue WCS.TO.JDE for outbound WCS XML messages
  - passing the message to WCSMessage for transformation
  - writing the transformed message (in OneWorld XML format) to the TO.ERP.SERVER queue.
- *JdeMessage.Java*. This class is used by JDE2NCIntegrationNodeHandler. It is responsible for parsing the OneWorld XML messages and transforming them to WCS XML format. The processor uses one of the following stylesheets for the transformation:
  - UpdateCustomer.xsl
  - CreateCustomer.xsl
  - OrderStatus.xsl
  - ProductPrice.xsl
  - ProductInventory.xsl.
- *WCSMessage.java*. This class is used by NC2JDEIntegrationNodeHandler. It is responsible for transforming the WCS XML messages to OneWorld XML format. The processor uses the Report_PurchaseOrder.xsl stylesheet for the transformation.

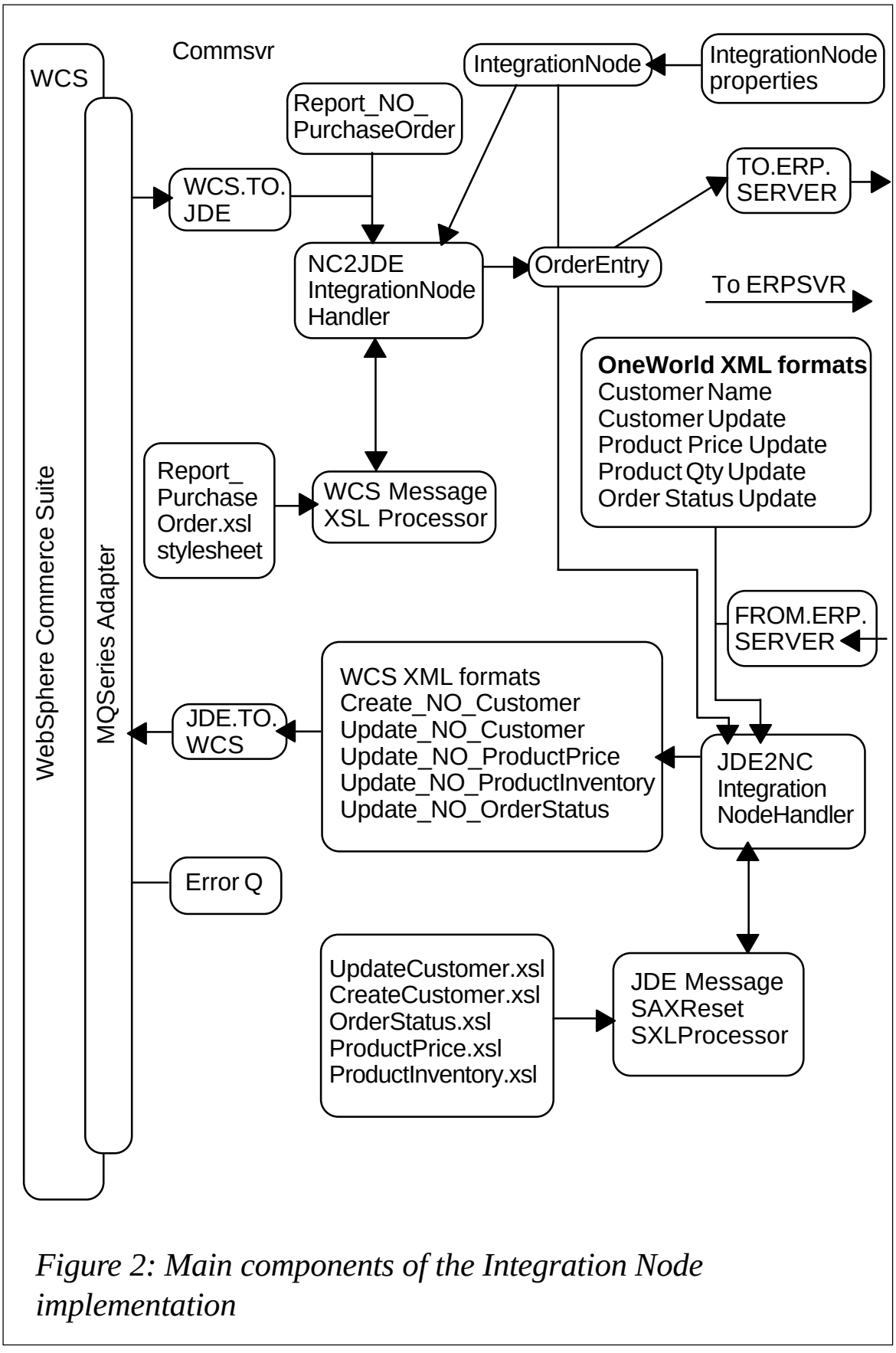

Figure 2: Main components of the Integration Node implementation

The IntegrationNode.property file stores WMQ configuration information, XSL stylesheet information, and other data specific to the environment where the Integration Node runs. Listed below is the IntegrationNode.property file used in this project.

```
MaxThreads=1
MaxMessageSize=1000000
QueueManager=COMM.SERVER
QueueChannel=SYSTEM.DEF.SVRCONN
QueueErrorQ=ERROR.Q
# JDE to WCS properties
QueueJDEOutboundQ=FROM.ERP.SERVER
QueueWCSInboundQ=JDE.TO.WCS
XslFileUpdateCustomer=./UpdateCustomer.xsl
XslFileCreateCustomer=./CreateCustomer.xsl
XslFileOrderStatus=./OrderStatus.xsl
XslFileProductPrice=./Product_Price.xsl
XslFileProductInventory=./ProductInventory.xsl
# WCS to JDE properties
QueueWCSOutboundQ=WCS.TO.JDE
QueueJDEInboundQ=TO.ERP.SERVER
XslFileReportPurchaseOrder=./Report_PurchaseOrder.xsl
```

Figure 2 shows the main components of the Integration Node implementation in our scenario.

The following sections provide more information on the Java classes used by the Integration Node implementation.

INTEGRATIONNODE CLASS

The main() method initiates the processing and performs the following tasks:

- Retrieves the following configuration information from the properties file IntegrationNode.property:
  - maximum number of threads
  - WMQ queue manager name
  - WMQ inbound queue name
  - WMQ outbound queue name
  - WMQ error queue name.

- Sets up the WMQ environment.
- Starts the thread that handles XML OneWorld to XML WCS message transformation (JDE2NCIntegrationNodeHandler).
- Starts the thread that handles XML WCS to XML OneWorld message transformation (NC2JDEIntegrationNodeHandler).
- Starts an infinite loop that wakes up the main thread every 60 seconds to ensure that there are active child threads.

To call the main() method in IntegrationNode we used the CL program eBit3/IntNode. We called the program from a Command Entry display or 'Submit it to Batch'. The program is listed below.

```
CD '/QIBM/USERDATA/COMMERCESUITE/INSTANCE/lifenet/INTEGRATIONNODE'
JAVA +
CLASS(com.ibm.as400.ebit3.integrationnode.IntegrationNode) +
CLASSPATH('.:./lotusxsl.jar:./xerces.jar:./xalan.jar')
```

JDE2NCINTEGRATIONNODEHANDLER CLASS

This class, which extends the Thread class, receives the following input parameters:

- WMQ inbound queue name.
- WMQ outbound queue name.
- WMQ error queue name.
- Properties file.

The run() method in this class performs the following steps:

- Sets WMQ message options.
- Reads the message in OneWorld XML format from the inbound queue.
- Transforms the message to WCS XML format by invoking the transform() method of the WCSMsg class.
- Puts the transformed message in WCS XML format in the outbound queue.

- Puts the message in the error queue if an error occurs during the message transformation.

NC2JDEINTEGRATIONNODEHANDLER CLASS

This class, which extends the Thread class, receives the following input parameters:

- WMQ inbound queue name
- WMQ outbound queue name
- WMQ error queue name
- Properties file.

The run() method in this class performs the following steps:

- Sets WMQ message options.
- Reads the message in WCS XML format from the inbound queue.
- Transforms the message to OneWorld XML format by invoking the transform() method of the JdeMsg class.
- Puts the transformed message in OneWorld XML format in the outbound queue.
- Puts the message in the error queue if an error occurs during the message transformation.

JDE MESSAGE CLASS

This class transforms messages from OneWorld XML format to the corresponding WCS XML format, using the XSLProcessor class provided with the LotusXSL package. The processor performs the translation using an XSL stylesheet and an XML message as input. To determine which XSL stylesheet to use, the SAX parser that is provided with LotusXSL (class SAXParser) parses the XML message.

The call-back methods in the JdeMessage class provide the logic that:

- Determines the message being translated. The type attribute in the <transaction> element is examined to determine the stylesheet to use in the translation. The <transaction> element looks like this:

  ```
  <transaction action='transactionInfo' type='JDEAB'>
  ```

- Determines the CustomerNew or CustomerUpdate message. OneWorld uses the same transaction type, JDEAB, for CustomerNew and CustomerUpdate messages. To distinguish the message being processed, examine the TransactionAction attribute of the <column> element, child of the <table> element, when the transaction type is JDEAB. The possible values are:

  – TransactionAction='A' (CustomerNew)

  – TransactionAction='UA' (CustomerUpdate).

- Determines the values of the phone number attributes in the CustomerNew and Customer Create messages. PhoneAreaCode, PhoneNumber, and PhoneNumberType are separate attributes in the OneWorld messages but they must be combined when translated to the WCS messages. The SAX parser parses these values. The XSL processor passes these values as parameters to the XSL stylesheet.

XSL STYLESHEETS

Extensible Stylesheet Language (XSL) is one of several specifications associated with XML. XSL is a programming language that can select and filter data in the original XML document while generating the results document.

The transformation engine (LotusXSL in this scenario) uses the stylesheet and the XML document as input and generates the specified output. The XSL stylesheet consists of a series of template rules, which are used by the processor and matched against the original XML. When there is a match the output is generated according to the specified rule. In this project we developed one XSL stylesheet for each message to transform.

DEBUGGING TECHNIQUES

During your tests you may find the following techniques useful for determining errors during the transformation process.

- Study examples of well-formed XML messages in WCS and OneWorld format. Compare those messages with the XML message that results from the transformation.
- Set the MS_LOGLEVEL in the Commerce Suite WMQ Adapter ncei_ece.ini file to 2 or higher. Message log level 2 or higher performs debug, status, and error logging. Debug logging writes detailed transaction information to the WCS ncei_ecexxxxxxxxxxx.log log file.
- Look for error messages in the ncei_ecexxxxxxx.log file. Here, xxxxx is the timestamp. Some useful error messages include:
  - CMN8103S: XML parsing message. An example is: 20001004160000 STATUS CMN8103S: XML parsing message for inbound message. File ICIMembufId, line 33, char 19: Expected end of tag 'Address'
  - CMN0206E: 'Parameter xxxx is missing'. An example is: 20001004161537 ERROR CMN0206E: Parameter 'salname' is missing
  - CMN0206D: 'Setting error handler task .....'
  - CMN80035: 'Unterminated entity'.
- If you do not find a helpful message in the log, retrieve the message from the ERROR.Q and examine it. It is probable that the parsing was successful but WCS ran into a problem processing the message. For example, the message 'Update_NC_Customer' would indicate that the customer (Login-ID) did not exist in the WCS database.

DESIGN CONSIDERATIONS

Think about the following design considerations before you implement the Integration Node.

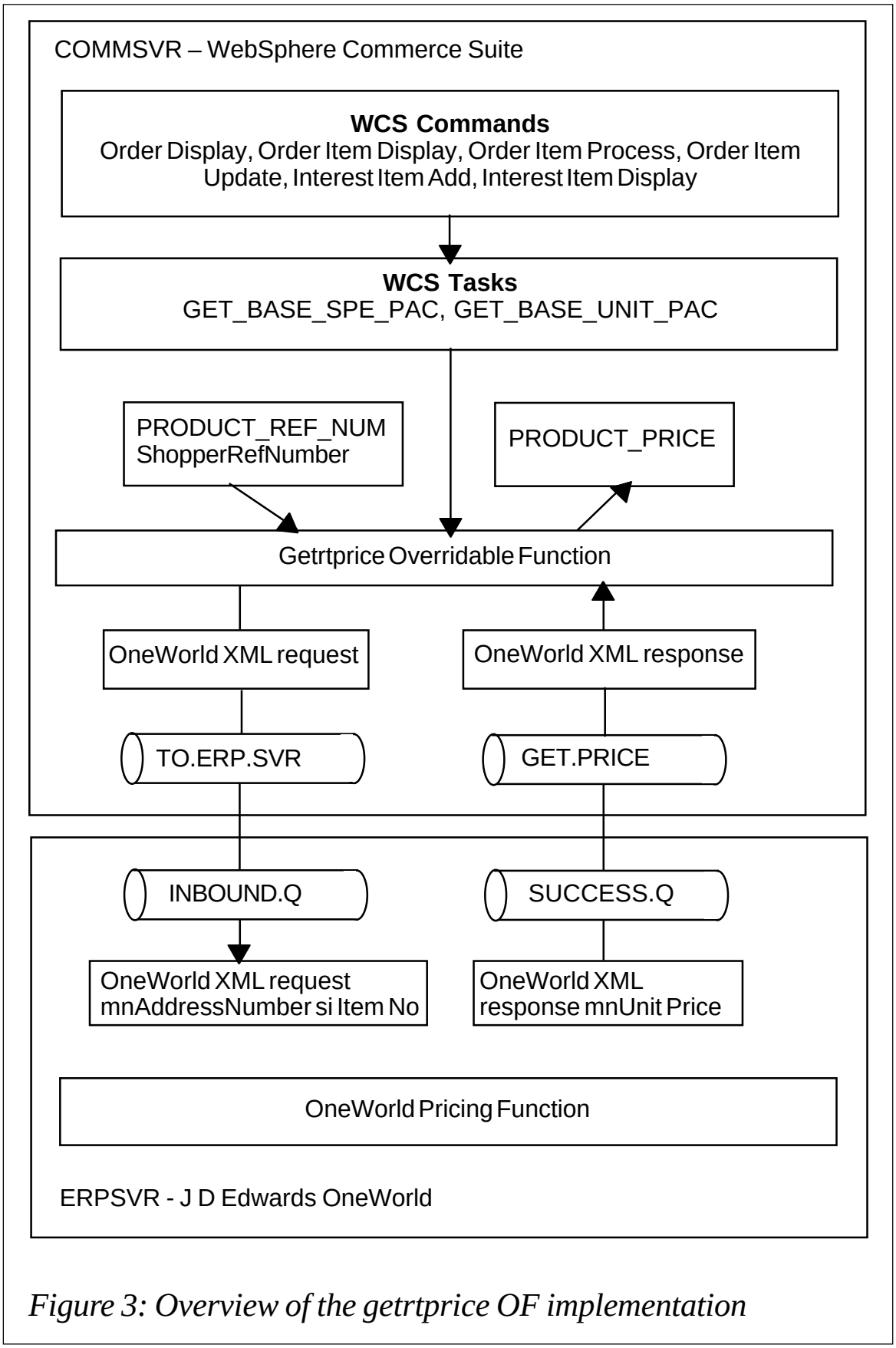

Figure 3: Overview of the getrtprice OF implementation

- When deciding the location of the Integration Node components consider the following options:
  - split the Integration Node components between the back-end ERPSVR server and the storefront COMMSVR server
  - place all Integration Node components in one of the two servers, COMMSVR or ERPSVR.

  Keeping all the components together simplifies maintenance and operation. Install the Integration Node components in COMMSVR because your programming and operations staff should have easier access to this server. In other environments different considerations may apply in determining which server should contain the application node. Some areas to consider are security, availability, and whether multiple WCS storefronts or other applications require the Integration Node to access the back-end system.
- The choice of programming language with which to develop the Integration Node. The supporting technologies used to implement the Integration Node, SAX parser, LotusXSL processor, and WMQ all have a good affinity with Java. They provide Java APIs for all the functions required to implement the Integration Node.
- Initiation of the Integration Node application. WMQ provides a facility for starting an application automatically when certain conditions are met (for example, when the number of messages in the queue reaches a specified number). This facility is called triggering. Using triggering causes a separate invocation of the Java application for each XML message that arrives in the queue with a negative impact on performance. We decided not to use this facility and, instead, manually called the IntegrationNode main() method (using a CL program), which starts one or multiple threads. The program must be ended manually.
- The handling of password reset by the Update_NC_Customer message. Password and VerifyPassword are required values in the Update_NC_Customer message. This causes the

application to reset the customer's password in the WCS database every time information changes in the customer record on the ERPSVR.

We set the CreateCustomer.xsl and UpdateCustomer.xsl stylesheet passwords to 'lifenet'. The business customer is notified by e-mail when the password is set to lifenet. The password should be changed for subsequent logon.

- Store environment variable names and attributes in the properties file. Use a properties file to provide the WMQ object names, stylesheet names, maximum number of threads, and any other information that the Integration Node application requires. This information can be changed without changing or recompiling the application objects.

CUSTOMIZING PRICE WITH AN OVERRIDABLE FUNCTION

The ERP system at the back-end server controls the pricing algorithm. Therefore, an overridable function (OF) was developed to extend the default behaviour of WCS. The approach described in this section is a specific example of how we customized WCS to suit our requirements. WCS provides several other areas of customization, such as adding new commands, customizing the database, and customizing the Net Data macros.

Implementing the getrtprice OF

To implement the getrtprice overridable function (OF) in this scenario, perform the following steps:

- Identify the need to modify the default behaviour of the pricing OF that WCS provides. The default behaviour is to retrieve the price of a product from the WCS tables. The pricing algorithm in the ERP system calculates the price of a product.
- Identify the areas where the default behaviour of WCS does not meet the requirements. In other words, identify the commands, tasks, and, most importantly, overridable functions that retrieve the price of a product.

- Identify the parameters that the pricing algorithm at the back-end system requires to calculate the price.
- Work out how to retrieve the parameters required by the back-end pricing algorithm from the WCS front-end. Also, identify the WCS environment variable to receive the price returned by the back-end application.
- Design the transport mechanism for communication with the back-end Principal algorithm.
- Code and test the getrtprice OF that implements the new pricing behaviour.
- Register the new getrtprice OF in the WCS database.
- Replace the default price OFs with the getrtprice OF using WCS administration forms.

Figure 3 shows an overview of the getrtprice OF implementation.

Identifying WCS pricing commands, tasks, and overridable functions

To identify the OFs you need to override, look up the related OFs and their default behaviour in the manual *Commands, Tasks, Overridable Functions, and Database Tables*. The following OFs related to price are listed in the documentation.

- *GetBaseSpePrc_1.0 (IBM,NC)*. This OF is designed to be called by the GET_BASE_SPE_PRC process task. The calling commands for the task are **OrderDisplay**, **OrderItemDisplay**, **OrderItemProcess**, and **OrderItemUpdate**.
- *GetBaseUnitPrc_1.0 (IBM,NC)*. This OF is designed to be called by the GET_BASE_UNIT_PRC and PROD_BASE_UNIT_PRC process tasks. The calling commands for the GET_BASE_UNIT_PRC process task are **InterestItemAdd** and **IntertestItemDisplay**. The calling command for the PROD_BASE_UNIT_PRC process task is **ProductDisplay**.

Identifying parameters required by the back-end pricing algorithm

In this scenario, the OneWorld MBF (Master Business Function) B4500460 controls the pricing algorithm. This program can receive several parameters to perform the price calculation. The getrtprice OF passes only two parameters to the back-end function – customer number and product number.

Communicating with the back-end ERP system

Communication with the back-end ERP system is through WMQ queues. The following queues are used for communication between the getrtprice OF and the ERP pricing application:

- COMMSVR server
  - outbound queue: TO.ERP.SERVER
  - inbound queue: GET.PRICE.
- ERPSVR
  - outbound queue: SUCCESS.Q
  - inbound queue: INBOUND.Q.

Referring to Figure 3 again it is important to note the following points:

- The messages sent and received by the getrtprice OF bypass both the Commerce Suite WMQ Adapter and the Integration Node program. The getrtprice OF formats the requests in the XML format expected by the OneWorld Adapter for WMQ for the pricing function. In this way you avoid the overhead of identifying the messages that do not need to be translated and forward them to the appropriate queue.
- At the ERPSVR server the OneWorld Adapter for WMQ processes the requests and responses:
  - the requests arrive through the standard OneWorld inbound queue (INBOUND.Q)

  - the responses are sent on the standard OneWorld success queue (SUCCESS.Q). By default the success queue stores messages that contain return code information for the OrderEntry message (handled by the Integration Node) and stores responses to the getrtprice overridable function requests. To separate both kinds of message configure the OSUCCESS queue to store return code messages for OrderEntry messages.

- Store the names of the outbound and inbound queues and the queue manager in COMMSVR in the nccommerce.ini configuration file. The objective is to not hard-code the name of the queues in the getrtprice OF for greater flexibility. The corresponding entries in the nccommerce.ini file are:
  - OUTBOUND_QUEUE_MANAGER COMM.SERVER
  - OUTBOUND_QUEUE TO.ERP.SERVER
  - INBOUND_QUEUE_MANAGER COMM.SERVER
  - INBOUND_QUEUE GET.PRICE.

Coding the getrtprice OF

When coding an OF, the starting point is the skeleton described in the document *Commands, Tasks, Overridable Functions, and Database Tables*, and in *Net.Commerce V3.2 for AS/400: A Case Study for Doing Business in the New Millennium,* SG24-5198. You can find an OF skeleton in the file template_of.cpp, which can be found in the integrated file system directory /Qibm/ProdData/CommerceSuite/adt/templates/.

Like commands, OFs have a base class (NC_OverridableFunction). An OF is basically a function that only implements the Process method. WCS calls the Process method, which in turn may call other methods that you implement. Listed below are the actions that the getrtprice overridable function performs.

- Get PRODUCT_REF_NUM from the environment.

- Get ShopperRefNumber from the cookie (method: getShopperRefNumberFromCookie).
- Generate XML message encoded according to OneWorld XML message format syntax (method: generateXML).
- Send XML message on the outbound queue to TO.ERP.SERVER (method: sendMessage).
- Receive XML response from OneWorld on the inbound queue GET.PRICE (method: getMessage).
- Parse the XML response to retrieve the product price (method: getPriceFrom JDEOM).
- Set PRODUCT_PRICE in the environment.

Compiling the getrtprice OF

The OF must be written in C++; however, you can code the OF business logic in any language and develop a C++ wrapper to call your program. Perform the following steps in order to compile the OF:

- Create the binding directory:

```
CRTBNDDIR BNDDIR(NETCBE/NETCBND) TEXT('WCS 4.1 Binding Directory')
```

- Add the necessary WCS service programs using the Add Binding Directory Entry (**ADDBNDDIRE**) command shown below:

```
ADDBNDDIRE BNDDIR(NETCBE/NETCBND) OBJ((QNETCOMM/QNECONTAIN))
ADDBNDDIRE BNDDIR(NETCBE/NETCBND) OBJ((QNETCOMM/QNEMESSAGE))
ADDBNDDIRE BNDDIR(NETCBE/NETCBND) OBJ((QNETCOMM/QNECOMMON))
ADDBNDDIRE BNDDIR(NETCBE/NETCBND) OBJ((QNETCOMM/QNEPAYOBJS))
ADDBNDDIRE BNDDIR(NETCBE/NETCBND) OBJ((XML4C310/LIBXERCESC))
ADDBNDDIRE BNDDIR(NETCBE/NETCBND) OBJ((QNETCOMM/QNEUTIL400))
ADDBNDDIRE BNDDIR(NETCBE/NETCBND) OBJ((QNETCOMM/QNEAUCMSG))
ADDBNDDIRE BNDDIR(NETCBE/NETCBND) OBJ((QNETCOMM/QNEAUCOBJ))
ADDBNDDIRE BNDDIR(NETCBE/NETCBND) OBJ((QNETCOMM/QNENOTIFY))
ADDBNDDIRE BNDDIR(NETCBE/NETCBND) OBJ((QNETCOMM/QNESVROBJ))
ADDBNDDIRE BNDDIR(NETCBE/NETCBND) OBJ((QNETCOMM/QNEWEBTOKE))
ADDBNDDIRE BNDDIR(NETCBE/NETCBND) OBJ((QNETCOMM/QNEDBC))
ADDBNDDIRE BNDDIR(NETCBE/NETCBND) OBJ((QMQM/LIBMQM))
```

- Create the C++ module using the Create C++ Module (**CRTCPPMOD**) command:

```
CRTCPPMOD MODULE(QTEMP/GETRTPRICE) SRCSTMF('/QIBM/USERDATA/
COMMERCESUITE/
INSTANCE/MANCO/CUSTOMOF/GETRTPRICE/GETRTPRICE.CPP') OUTPUT('/QIBM/
USER
DATA/COMMERCESUITE/INSTANCE/MANCOTST/CUSTOMOF/GETRTPRICE/
GETRTPRICE.LST')
DEFINE(AS4ØØ '__TRACE_NCAPIS__')
INCDIR('/QIBM/PRODDATA/COMMERCESUITE/ADT/INCLUDE''/QIBM/PRODDATA/
MQM/INC')
```

- Create the service program using the Create Service Program (**CRTSRVPGM**) command:

```
CRTSRVPGM SRVPGM(QNETCOMM/GETRTPRICE) MODULE(QTEMP/GETRTPRICE) +
EXPORT(*ALL) BNDDIR(NETCBE/NETCBND)
```

Registering the getrtprice OF with WCS

Registering an OF in the database is done manually by inserting a row into the WCS instance table named OFS. We used the SQL statement:

```
insert into lifenet/ofs (refnum , dll_name, vendor, product,
name,version,
description) select max(refnum) + 1,'QNETCOMM/GETRTPRICE', 'IBM',
'WCS','getrtprice', + 1.Ø ,'ERP calculates price'from manco/ofs
```

Assigning the OF to WCS tasks

Use the Commerce Suite Administrator tool to assign the customized OF to a task. Refer to WCS online help (*Create overridable functions – Activate Customized Overridable Functions*) for information on how to activate customized OFs. In this scenario we assigned the getrtprice OF to the GET_BASE_SPE_PRC and GET_BASE_UNIT_PRC process tasks.

The OF can be assigned at the mall or store level; assign it at the store level in this scenario.

Design considerations

There are some design decisions to consider before implementing the getrtprice OF:

- *Using WMQ as the transport mechanism*. Consider using a sockets program on both COMMSVR and ERPSVR to send the price quote request and to receive the response. This approach may provide better performance, which is an important consideration in any environment where real-time transactions are involved.

  Using WMQ simplifies programming, guarantees delivery, and minimizes the programming effort at the back-end ERP system. Since WMQ is already used as the transport mechanism for the other transactions between COMMSVR and ERPSVR, the cost of the product is not an important factor.

- *Parsing the response to retrieve the product price*. Consider using the IBM XML for C++ parser (XML4C) for the iSeries platform, which can be downloaded from: *http://www.alphaworks.ibm.com*. You can write the function to parse the response string instead. The advantages of implementing your own parsing function include:

  – it is a simpler technique than using the XML for C++ parser because there is only one message type to parse.

  – you get better performance by avoiding the overhead of starting the parser and parsing the response only to get one piece of information (price).

  – the XML for C++ parser is experimental and subject to change.

- *Selecting the programing language for the OF*. WCS V4.1 can call only OFs written in C++. Consider coding the OF in Java because it is generally the preferred language for portability and for human resource skills. Coding the function in Java requires a C++ wrapper, which adds the overhead of starting the Java Virtual Machine (JVM) every time the OF is called and has a negative impact on performance. For this reason we selected C++ to code the OF.

- *Selecting the simplified price business function at the back-end ERP system*. Instead of using the standard OneWorld

pricing business function (B4201500) you can use a simpler interface that JDE makes available with Electronic Software Update (ESU) 4328880. The simplified function is B4500460. This function is intended to provide a relatively simple business function, with fewer parameters, to retrieve a price from the OneWorld pricing application.

Correlating WMQ messages in the OF

Responses to the price quote requests from multiple users are received in the GET.PRICE queue. It is necessary to devise a mechanism to guarantee the correlation of a request to the corresponding response. The following recommendations will help in the implementation of a correlation mechanism:

- WMQ provides Message-ID and Correlation-ID fields in each message it sends, along with the ability to populate the fields so that the Message-ID and Correlation-ID always form a unique key to identify any message. The WMQ receiving queues also provide the capability selectively to remove a message from the queue, based on its Message-ID and Correlation-ID combination.

  Using this default WMQ behaviour is the best way to correlate an outbound message with its response. You have no guarantee that the OneWorld Adapter for WMQ honours the Message-ID and Correlation-ID fields in reply messages.

- Generate and insert a unique Message-ID into the OneWorld XML request. Add a parameter to the XML document to hold the generated ID. Current understanding of OneWorld holds that, if an XML message contains a parameter that is not needed or expected, it will simply be passed back as part of the response. To retrieve the OneWorld response from the inbound queue, browse the queue message by message to locate the message containing the generated ID.

- Use the Shopper Reference Number and Product Reference Number, which form part of the original request XML document, to identify the proper message in the response queue. Browse

the queue to locate the message containing the reference numbers that were sent in the original price request document. This is the recommended approach if the OneWorld Adapter for WMQ does not honour the WMQ Message-ID and Correlation-ID fields.

SUMMARY

This article provides an overview of how to use WMQ to transport messages from a Web storefront to an ERP system. Further work is being performed on this project to handle returns from customers. The return requests will be created as requests from the Web store and will be augmented to the JDE accounting and inventory subsystems using Overridable Functions. For details on the programming specifications of the interfaces please contact the author at *vikas.baruah@ams.com*.

Vikas Baruah (vikas.baruah@ams.com)
American Management Systems (USA)

MQBRIDGE

This WebSphere MQSeries program is designed to act as a common interface between WMQ client programs and the CICS 3270 Bridge.

```
CBL NODYNAM,LIB,OBJECT,RENT,RES,APOST
       IDENTIFICATION DIVISION.
       PROGRAM-ID. MQBRIDGE.
      *REMARKS
      *   author : nour-eddin al-sarairah.                          *
      *  Environment      : cics online run by infc transaction     *
      *  HOW IT WORKS .                                             *
      *  - MODIFY THE PROGRAM TO INCLUDE YOUR TARGET QUEUE          *
      *    AND REPLY QUEUE.                                         *
      *  - DEFINE A TRANSACTION WITH INFC NAME POINT TO THIS        *
      *    PROGRAME (MQBRIDGE) .                                    *
      *  - ON THE HOST SITE DEFINE A TRIGGER QUEUE(TARGET) WITH     *
      *    PROCESS NAME INFC WHICH TRIGGERS INFC TRANS.             *
```

```
*  - ON THE CLIENT SIDE IT PUTS (MESSAGE) WITH ITS INFORMATION  *
*    INCLUDING IN THE FIRST 4 FIELDS THE  TRANSACTION           *
*    NAME THAT WILL BE INVOKED (327Ø) ON THE HOST (CICS).       *
*  - ONCE THE DATA PUT ON THE REMOTE QUEUE FROM THE CLIENT SIDE *
*    AND ONCE ITS PUT ON THE TARGET QUEUE ON THE HOST. IT WILL  *
*    TRIGGER INFC PROCESS, WHICH WILL CALL INFC TRANSACTION     *
*    CONNECTED ON THE SPECIFIED CICS INITIATION QUEUE.          *
*  - THE PROGRAM WILL READ THE TARGET QUEUE AND GET ITS MESSAGE *
*  THEN IT WILL CONTINUE PROCESS BY CALCULATING THE SIZE OF     *
*  THE DATA CAPTURED AND THE NAME OF THE TRANSACTION TO BE      *
*  PROCESSED.                                                   *
*  - IT PREPARES THE PROGRAM TO PUT A MESSAGE ON THE            *
*    CICS MQ BRDIGE (SYSTEM.CICS.BRIDGE.QUEUE) WITH ALL         *
*    SUITABLE VALUE AND MESSAGE OPTIONS PASSING TO IT           *
*    THE REPLY-TO-QUEUE NAME .                                  *
*  - THEN THE PROGRAM FINISHES ITS WORK    BY SENDING           *
*    MANY MESSAGES TO THE CICS LOG.                             *
*  - THE 327Ø INTERFACE WILL INVOKE THE TRANSACTION             *
*    AND PASS TO IT THE INFORMATION AND PUT THE REPLY ON        *
*    THE REPLY-TO-Q WHICH THE CLIENT CAN READ AND PROCESS.      *
* NOTE :-                                                       *
* THE PROGRAM CAN BE CHANGED TO BE DYNAMICALY PARSING           *
* THE REQUEST QUEUE NAME AND REPLY-Q-NAME PASSED FROM THE CLIENT*
* AND TAKE CARE TO CHANGE THE QMGR NAME, WHICH IS MQSB IN THIS  *
* EXAMPLE.                                                      *
*    WØØ - General work fields
 Ø1  WØØ-RETURN-CODE             PIC S9(4) BINARY VALUE ZERO.
 Ø1  WØØ-LOOP                    PIC S9(9) BINARY VALUE Ø.
 Ø1  WØØ-NUMPUTS                 PIC S9(9) BINARY VALUE Ø.
 Ø1  WØØ-ERROR-MESSAGE           PIC X(48) VALUE SPACES.
*   Parameter variables
 Ø1  WS-DATALENGTH               PIC 9(4) VALUE ZEROES.
 Ø1  WS-OP-SUCCESSFUL   PIC X(2Ø) VALUE 'OPEN SUCCESSFUL'.
 Ø1  WS-OP-BRIDGESUCCESSFUL   PIC X(2Ø) VALUE 'OPEN BRIDESUCCSF'.
 Ø1  WS-OPEN                     PIC X(8) VALUE 'CSQCOPEN'.
 Ø1  WS-COMMIT                   PIC X(8) VALUE 'CSQCCOMM'.
 Ø1  WS-CLOSE                    PIC X(8) VALUE 'CSQCCLOS'.
 Ø1  WS-CONN                     PIC X(8) VALUE 'CSQCCONN'.
 Ø1  WS-DISC                     PIC X(8) VALUE 'CSQCDISC'.
 Ø1  WS-GET                      PIC X(8) VALUE 'CSQCGET'.
 Ø1  WS-PUT                      PIC X(8) VALUE 'CSQCPUT'.
 Ø1  WS-PUT1                     PIC X(8) VALUE 'CSQCPUT1'.
 Ø1  WØØ-QMGR                    PIC X(48).
 Ø1  WØØ-QNAME                   PIC X(48).
 Ø1  WØØ-BNAME                   PIC X(48).
 Ø1  WS-MSGID                    PIC X(24).
 Ø1  WS-CORRELID                 PIC X(24).
 Ø1  WØØ-PADCHAR                 PIC X(1) VALUE '*'.
 Ø1  WØØ-MSGBUFFER               PIC X(1ØØØ) VALUE SPACES.
 Ø1  WØØ-NUMMSGS-CHAR            PIC X(4) VALUE SPACES.
```

```
 Ø1  WØØ-NUMMSGS                PIC S9(9) BINARY VALUE 1.
 Ø1  WØØ-MSGLENGTH              PIC S9(9) BINARY VALUE +1ØØØ.
 Ø1  WS-MSGLENGTH               PIC S9(9) BINARY VALUE Ø.
 Ø1  WØØ-DATALENGTH             PIC S9(9) BINARY.
 Ø1  WS-GET-DATA.
     Ø5 WS-TRANSID PIC X(4) VALUE SPACES.
     Ø5 FILLER     PIC X(996) VALUE SPACES.
 Ø1  WØØ-PERSISTENCE            PIC X(1) VALUE 'N'.
     88 PERSISTENT     VALUE 'P'.
     88 NOT-PERSISTENT VALUE 'N'.
*    WØ3 - API fields
 Ø1  WØ3-HCONN                  PIC S9(9) BINARY VALUE Ø.
 Ø1  WØ3-HOBJ                   PIC S9(9) BINARY VALUE Ø.
 Ø1  WBR-HOBJ                   PIC S9(9) BINARY VALUE Ø.
 Ø1  WØ3-OPENOPTIONS            PIC S9(9) BINARY.
 Ø1  WBR-OPENOPTIONS            PIC S9(9) BINARY.
 Ø1  WØ3-COMPCODE               PIC S9(9) BINARY.
 Ø1  WØ3-REASON                 PIC S9(9) BINARY.
 Ø1  WS-REASON                 PIC 9(1Ø) VALUE ZEROES.
*    API control blocks
 Ø1  MQM-OBJECT-DESCRIPTOR.
     COPY CMQODV.
 Ø1  MQM-MESSAGE-DESCRIPTOR.
     COPY CMQMDV.
 Ø1  MQM-PUT-MESSAGE-OPTIONS.
     COPY CMQPMOV.
 Ø1  MQM-GET-MESSAGE-OPTIONS.
     COPY CMQGMOV.
 Ø1  MQM-CONSTANTS.
     COPY CMQV SUPPRESS.
 Ø1  OUTPUT-BUFFER.
    Ø2 CICS-MESSAGE-HEADER.
**   MQCIH structure
       1Ø MQCIH.
**    Structure identifier
       15 MQCIH-STRUCID            PIC X(4) VALUE 'CIH '.
**    Structure version number
       15 MQCIH-VERSION            PIC S9(9) BINARY VALUE 2.
**    Length of MQCIH structure
       15 MQCIH-STRUCLENGTH        PIC S9(9) BINARY VALUE 18Ø.
**    Reserved
       15 MQCIH-ENCODING           PIC S9(9) BINARY VALUE Ø.
**    Reserved
       15 MQCIH-CODEDCHARSETID     PIC S9(9) BINARY VALUE Ø.
**    MQ format name
       15 MQCIH-FORMAT             PIC X(8) VALUE SPACES.
**    Reserved
       15 MQCIH-FLAGS              PIC S9(9) BINARY VALUE Ø.
**    Return code from bridge
       15 MQCIH-RETURNCODE         PIC S9(9) BINARY VALUE Ø.
```

```
**    MQ completion code or CICS EIBRESP
       15 MQCIH-COMPCODE           PIC S9(9) BINARY VALUE Ø.
**    MQ reason or feedback code, or CICS EIBRESP2
       15 MQCIH-REASON             PIC S9(9) BINARY VALUE Ø.
**    Unit-of-work control
       15 MQCIH-UOWCONTROL         PIC S9(9) BINARY VALUE 273.
**    Wait interval for MQGET call issued by bridge
       15 MQCIH-GETWAITINTERVAL    PIC S9(9) BINARY VALUE -2.
**    Link type
       15 MQCIH-LINKTYPE           PIC S9(9) BINARY VALUE 1.
**    Output commarea data length
       15 MQCIH-OUTPUTDATALENGTH   PIC S9(9) BINARY VALUE -1.
**    Bridge facility release time
       15 MQCIH-FACILITYKEEPTIME   PIC S9(9) BINARY VALUE Ø.
**    Send/receive ADS descriptor
       15 MQCIH-ADSDESCRIPTOR      PIC S9(9) BINARY VALUE Ø.
**    Whether task can be conversational
       15 MQCIH-CONVERSATIONALTASK PIC S9(9) BINARY VALUE Ø.
**    Status at end of task
       15 MQCIH-TASKENDSTATUS      PIC S9(9) BINARY VALUE Ø.
**    BVT token value
       15 MQCIH-FACILITY           PIC X(8) VALUE LOW-VALUES.
**    MQ call name or CICS EIBFN function name
       15 MQCIH-FUNCTION           PIC X(4) VALUE SPACES.
**    Abend code
       15 MQCIH-ABENDCODE          PIC X(4) VALUE SPACES.
**    Password or passticket
       15 MQCIH-AUTHENTICATOR      PIC X(8) VALUE SPACES.
**    Reserved
       15 MQCIH-RESERVED1          PIC X(8) VALUE SPACES.
**    MQ format name of reply message
       15 MQCIH-REPLYTOFORMAT      PIC X(8) VALUE SPACES.
**    Remote sysid to use
       15 MQCIH-REMOTESYSID        PIC X(4) VALUE 'MQSB'.
**    Remote transid to attach
       15 MQCIH-REMOTETRANSID      PIC X(4) VALUE SPACES.
**    Transaction to attach
       15 MQCIH-TRANSACTIONID      PIC X(4) VALUE SPACES.
**    Terminal emulated attributes
       15 MQCIH-FACILITYLIKE       PIC X(4) VALUE SPACES.
**    AID key
       15 MQCIH-ATTENTIONID        PIC X(4) VALUE SPACES.
**    Transaction start code
       15 MQCIH-STARTCODE          PIC X(4) VALUE SPACES.
**    Abend transaction code
       15 MQCIH-CANCELCODE         PIC X(4) VALUE SPACES.
**    Next transaction to attach
       15 MQCIH-NEXTTRANSACTIONID  PIC X(4) VALUE SPACES.
**    Reserved
       15 MQCIH-RESERVED2          PIC X(8) VALUE SPACES.
```

```
**    Reserved
       15 MQCIH-RESERVED3          PIC X(8) VALUE SPACES.
**    Cursor position
       15 MQCIH-CURSORPOSITION     PIC S9(9) BINARY VALUE Ø.
**    Offset of error in message
       15 MQCIH-ERROROFFSET        PIC S9(9) BINARY VALUE Ø.
**    Item number of last message read
       15 MQCIH-INPUTITEM          PIC S9(9) BINARY VALUE Ø.
**    Reserved
       15 MQCIH-RESERVED4          PIC S9(9) BINARY VALUE Ø.
*   Ø2 BRMQ-VECTOR-HEADER.
*   LENGTH OF VECTOR
       15 BRMQ-VECTOR-LENGTH PIC S9(8) COMP VALUE Ø.
*   SEND OR RECEIVE - TEXT
*   OR MAP ETC
       15 BRMQ-VECTOR-DESCRIPTOR PIC X(4) VALUE 'Ø4Ø2'.
*   'O' OUTBOUND VECTOR 'I'
*   INBOUND VECTOR
       15 BRMQ-VECTOR-TYPE PIC X(4) VALUE 'I'.
*   'Ø' VECTOR VERSION NUMBER
       15 BRMQ-VECTOR-VERSION PIC X(4) VALUE 'ØØØØ'.
*   Ø2 VECTOR-DETAIL-RECEIVE.
* INBOUND VECTOR - RECEIVE
*      15 FILLER PIC X(16).
*   'N' DELETE SEND
*   AREAS ALREADY GENERATED 'Y'
*   TRANSMIT SEND AREAS
     15 BRMQ-RE-TRANSMIT-SEND-AREAS PIC X(4) VALUE 'N'.
*   'N' BUFFER NOT
*   SPECIFIED 'Y' BUFFER SPECIFIED
       15 BRMQ-RE-BUFFER-INDICATOR PIC X(4) VALUE 'N'.
*   AID KEY VALUE
       15 BRMQ-RE-AID PIC X(4).
*   CURSOR POSITION
       15 BRMQ-RE-CPOSN PIC S9(8) COMP.
*   LENGTH OF DATA ON RECEIVE
       15 BRMQ-RE-DATA-LEN PIC S9(8) COMP VALUE Ø.
*THIS VECTOR IS IDENTICAL TO RECEIVE, EXCEPT THAT THE
*COMMAND COD INDICATES CONVERSE, RATHER THAN RECEIVE
*AND BUFFER IS NOT SUPPORTED ON CONVERSE.
       Ø3 MESSAGE-DATA PIC X(1ØØØ) VALUE SPACES.
*    MQV contains constants (for filling in the control blocks)
*    and return codes (for testing the result of a call)
 Ø1  INPUT-BUFFER.
    Ø2 CICS-MESSAGE-HEADER.
      1Ø MQCIHK.
**    Structure identifier
      15 MQCIH-STRUCIDK           PIC X(4) VALUE 'CIH '.
**    Structure version number                    *@L1A*
      15 MQCIH-VERSIONK           PIC S9(9) BINARY VALUE 2.
```

```
**    Length of MQCIH structure                    *@L1A*
      15 MQCIH-STRUCLENGTHK       PIC S9(9) BINARY VALUE 18Ø.
**    Reserved
      15 MQCIH-ENCODINGK          PIC S9(9) BINARY VALUE Ø.
**    Reserved
      15 MQCIH-CODEDCHARSETIDK    PIC S9(9) BINARY VALUE Ø.
**    MQ format name
      15 MQCIH-FORMATK            PIC X(8) VALUE SPACES.
**    Reserved
      15 MQCIH-FLAGSK             PIC S9(9) BINARY VALUE Ø.
**    Return code from bridge
      15 MQCIH-RETURNCODEK        PIC S9(9) BINARY VALUE Ø.
**    MQ completion code or CICS EIBRESP
       15 MQCIH-COMPCODEK          PIC S9(9) BINARY VALUE Ø.
**    MQ reason or feedback code, or CICS EIBRESP2
       15 MQCIH-REASONK            PIC S9(9) BINARY VALUE Ø.
**    Unit-of-work control
       15 MQCIH-UOWCONTROLK        PIC S9(9) BINARY VALUE 273.
**    Wait interval for MQGET call issued by bridge
       15 MQCIH-GETWAITINTERVALK   PIC S9(9) BINARY VALUE -2.
**    Link type
       15 MQCIH-LINKTYPEK          PIC S9(9) BINARY VALUE 1.
**    Output commarea data length
       15 MQCIH-OUTPUTDATALENGTHK  PIC S9(9) BINARY VALUE -1.
**    Bridge facility release time
       15 MQCIH-FACILITYKEEPTIMEK  PIC S9(9) BINARY VALUE Ø.
**    Send/receive ADS descriptor
       15 MQCIH-ADSDESCRIPTORK     PIC S9(9) BINARY VALUE Ø.
**    Whether task can be conversational
       15 MQCIH-CONVERSATIONALTASKK PIC S9(9) BINARY VALUE Ø.
**    Status at end of task
       15 MQCIH-TASKENDSTATUSK      PIC S9(9) BINARY VALUE Ø.
**    BVT token value
       15 MQCIH-FACILITYK          PIC X(8) VALUE LOW-VALUES.
**    MQ call name or CICS EIBFN function name
       15 MQCIH-FUNCTIONK          PIC X(4) VALUE SPACES.
**    Abend code
       15 MQCIH-ABENDCODEK         PIC X(4) VALUE SPACES.
**    Password or passticket
       15 MQCIH-AUTHENTICATORK     PIC X(8) VALUE SPACES.
**    Reserved
       15 MQCIH-RESERVED1K         PIC X(8) VALUE SPACES.
**    MQ format name of reply message
       15 MQCIH-REPLYTOFORMATK     PIC X(8) VALUE SPACES.
**    Remote sysid to use
       15 MQCIH-REMOTESYSIDK       PIC X(4) VALUE SPACES.
**    Remote transid to attach
       15 MQCIH-REMOTETRANSIDK     PIC X(4) VALUE SPACES.
**    Transaction to attach
       15 MQCIH-TRANSACTIONIDK     PIC X(4) VALUE SPACES.
```

```
      **    Terminal emulated attributes
             15 MQCIH-FACILITYLIKEK      PIC X(4) VALUE SPACES.
      **    AID key
             15 MQCIH-ATTENTIONIDK       PIC X(4) VALUE SPACES.
      **    Transaction start code
             15 MQCIH-STARTCODEK         PIC X(4) VALUE SPACES.
      **    Abend transaction code
             15 MQCIH-CANCELCODEK        PIC X(4) VALUE SPACES.
      **    Next transaction to attach
             15 MQCIH-NEXTTRANSACTIONIDK PIC X(4) VALUE SPACES.
      **    Reserved
             15 MQCIH-RESERVED2K         PIC X(8) VALUE SPACES.
      **    Reserved
             15 MQCIH-RESERVED3K         PIC X(8) VALUE SPACES.
      **    Cursor position
*@L1A*
             15 MQCIH-CURSORPOSITIONK    PIC S9(9) BINARY VALUE Ø.
      **    Error offset
*@L1A*
             15 MQCIH-ERROROFFSETK       PIC S9(9) BINARY VALUE Ø.
      **    Input item
*@L1A*
             15 MQCIH-INPUTITEMK         PIC S9(9) BINARY VALUE Ø.
      **    Reserved
*@L1A*
             15 MQCIH-RESERVED4K         PIC S9(9) BINARY VALUE Ø.
      * OUTBOUND VECTOR - SEND
          Ø2 BRMQ-SEND.
             Ø3 FILLER PIC X(16).
      *   'N' NO ERASE OPTION
      *   SPECIFIED 'D' DEFAULT SPECIFIED
      *   ON COMMAND 'A' ALTERNATE
      *   SPECIFIED ON CMND
             Ø3 BRMQ-SE-ERASE-INDICATOR PIC X(4) VALUE 'E'.
      *   CONTROL CHARACTER (CTLCHAR)
      *   SPEC 'C' DEFAULT IF CTLCHAR NOT
      *   SPEC
             Ø3 BRMQ-SE-CTLCHAR PIC X(4).
      *   'N' STRUCTURED FIELD
      *   INDICATOR (STRFIELD) NOT
      *   SPECIFIED ON CMND 'Y'
      *   STRUCTURED FIELD INDICATOR
      *   (STRFIELD) SPECIFIED ON COMMAND
             Ø3 BRMQ-SE-STRFIELD-INDICATOR PIC X(4).
      *   '3' DEFINITE RESPONSE                          Ø
      *   INDICATOR (DEFRESP) NOT
      *   SPECIFIED ON CMND 'Y' DEFINITE
      *   RESPONSE INDICATOR (DEFRESP)
      *   SPECIFIED ON COMMAND
             Ø3 BRMQ-SE-DEFRESP-INDICATOR PIC X(4).
```

```
*   'N' INVITE INDICATOR
*   NOT SPEC 'Y' INVITE INDICATOR
*   SPEC
       Ø3 BRMQ-SE-INVITE-INDICATOR PIC X(4) VALUE 'N'.
*   'N' LAST INDICATOR NOT
*   SPECIFIED 'Y' LAST INDICATOR
*   SPECIFIED
       Ø3 BRMQ-SE-LAST-INDICATOR PIC X(4) VALUE 'Y'.
*   'N' FLUSH BUFFER
*   INDICATOR (WAIT) NOT SPECIFIED
*   ON COMMAND 'Y' FLUSH BUFFER
*   INDICATOR (WAIT) SPECIFIED ON
*   COMMAND
       Ø3 BRMQ-SE-WAIT-INDICATOR PIC X(4) VALUE 'Y'.
*   LENGTH OF TEXT ON SEND
       Ø3 BRMQ-SE-DATA-LEN PIC S9(8) COMP VALUE +598.
*THIS DSECT IS COMMON TO SEND CONTROL, SEND MAP AND
*SEND TEXT. THE BRMQ_SEND_MAP AND BRMQ_SEND_TEXT
*DSECT'S ARE CONCATENATED ON TOTHE END OF THIS
*DEPENDING THE REQUEST BEING PROCESSED.
     Ø3 Message-data-outbound pic x(598) value spaces
A-MAIN SECTION.
** MOVE THE NAME OF THE TARGET QUEUE TO BE READ                - *
     MOVE 'MQSB.REMOTE.AIXQ' TO WØØ-QNAME
** MOVE THE NAME OF THE CICS BRDIGE QUEUE (MUST)               - *
     MOVE 'SYSTEM.CICS.BRIDGE.QUEUE' TO WØØ-BNAME
     MOVE WØØ-NUMMSGS-CHAR   TO WØØ-NUMMSGS.
*    Connect to the queue manager
     MOVE MQOO-INPUT-SHARED TO WØ3-OPENOPTIONS.
     MOVE WØØ-QNAME   TO MQOD-OBJECTNAME.
     CALL WS-OPEN  USING WØ3-HCONN
                         MQOD
                         WØ3-OPENOPTIONS
                         WØ3-HOBJ
                         WØ3-COMPCODE
                         WØ3-REASON.
*    If open failed then display error message
*    and exit.
     IF (WØ3-COMPCODE NOT = MQCC-OK) THEN
        MOVE 'MQOPEN LOCAL'   TO WØØ-ERROR-MESSAGE
        MOVE WØ3-REASON TO WS-REASON
        PERFORM DISPLAY-ERROR-MESSAGE
        PERFORM DØØØØ-MQCLOSE
        EXEC CICS ABEND ABCODE('OERR') END-EXEC
*       GO TO A-MAIN-DISCONNECT
     ELSE
     EXEC CICS WRITEQ TD QUEUE('CSSL')
                      FROM(WS-OP-SUCCESSFUL)
                      LENGTH(2Ø)
                      END-EXEC.
```

```
           PERFORM WS-GET-SECTION.
           PERFORM WØØØ-OPEN-BRIDGE.
      *    Set PARAMETERS FO MQCIH
               MOVE MQENC-NATIVE TO MQMD-ENCODING
               MOVE WS-MSGID  TO MQMD-MSGID
               MOVE MQCI-NEW-SESSION TO MQMD-CORRELID
               MOVE MQMT-REQUEST TO MQMD-MSGTYPE
               MOVE MQFMT-CICS  TO MQMD-FORMAT
      ********
      ********  MOVE THE NAME OF REPLY-TO-QUEUE PASSED FROM CLIENT *
      ********
               MOVE 'MQIX.REPLY.MQSBQ' TO MQMD-REPLYTOQ "REPLY TO
QUEUE"
               MOVE MQPMO-NO-SYNCPOINT TO MQPMO-OPTIONS
               MOVE +12ØØ  TO MQMD-EXPIRY
               MOVE MQCSC-TERMINPUT TO MQCIH-STARTCODE
               MOVE WS-TRANSID     TO  MQCIH-TRANSACTIONID
      *******     "FIRST 4 CHARACTER READ FROM MESSAGE"
               MOVE MQCFAC-NONE    TO MQCIH-FACILITY
               MOVE MQCCT-NO       TO MQCIH-CONVERSATIONALTASK
               MOVE MQCTES-NOSYNC  TO MQCIH-TASKENDSTATUS
               MOVE MQCUOWC-ONLY   TO  MQCIH-UOWCONTROL
      *        COMPUTE MQCIH-ADSDESCRIPTOR = MQCADSD-RECV +
      *                                      MQCADSD-SEND +
      *                                      MQCADSD-MSGFORMAT
      *        COMPUTE MQCIH-ADSDESCRIPTOR = MQCADSD-RECV +
      *                                      MQCADSD-SEND +
      *                                      MQCADSD-MSGFORMAT
               MOVE MQCLT-TRANSACTION TO MQCIH-LINKTYPE
               MOVE MQCCT-NO       TO  MQCIH-CONVERSATIONALTASK
               MOVE WS-GET-DATA TO MESSAGE-DATA
               CALL WS-PUT  USING WØ3-HCONN
                                  WBR-HOBJ
                                  MQMD
                                  MQPMO
                                  WS-MSGLENGTH
                                  OUTPUT-BUFFER
                                  WØ3-COMPCODE
                                  WØ3-REASON
      *        If put failed then display error message
      *        and break out of loop
               IF (WØ3-COMPCODE NOT = MQCC-OK) THEN
                  MOVE 'MQPUT BRIDGE' TO WØØ-ERROR-MESSAGE
                  MOVE WØ3-REASON TO WS-REASON
                  PERFORM DISPLAY-ERROR-MESSAGE
                  EXEC CICS ABEND ABCODE('BRRR') END-EXEC
               ELSE
              MOVE 'MQPUT BRIDGE SUCC'   TO WØØ-ERROR-MESSAGE
              EXEC CICS WRITEQ TD QUEUE ('CSSL')
              FROM (WØØ-ERROR-MESSAGE) LENGTH(5Ø) END-EXEC
```

```
        END-IF.
*    Close the queue
     CALL WS-CLOSE  USING WØ3-HCONN
                          WBR-HOBJ
                          MQCO-NONE
                          WØ3-COMPCODE
                          WØ3-REASON.
     IF (WØ3-COMPCODE NOT = MQCC-OK) THEN
        MOVE 'MQCLOSE BRIDGE'  TO WØØ-ERROR-MESSAGE
        MOVE WØ3-REASON TO WS-REASON
        PERFORM DISPLAY-ERROR-MESSAGE
        EXEC CICS ABEND ABCODE('BRCL') END-EXEC
     ELSE
        MOVE 'MQCLOS BRIDGE SUCC'   TO WØØ-ERROR-MESSAGE
        EXEC CICS WRITEQ TD QUEUE ('CSSL')
        FROM (WØØ-ERROR-MESSAGE) LENGTH (5Ø) END-EXEC
     END-IF.
     EXEC CICS RETURN END-EXEC.
 USAGE-ERROR SECTION.
     DISPLAY            DISPLAY 'PARAMETERS FOR PROGRAM :'.
     DISPLAY '     QMGR        - QUEUE MANGER'.
     DISPLAY '     QNAME       - QUEUE NAME'.
     DISPLAY '     NUMMSGS     - NUMBER OF MESSAGES'.
     DISPLAY '     PADCHAR     - MESSAGE PADDING CHARACTER'.
     DISPLAY '     MSGLENGTH   - LENGTH OF MESSAGE(S)'.
     DISPLAY '     PERSISTENCE - PERSISTENCE OF MESSAGE(S)'.
     DISPLAY        USAGE-ERROR-END.
*    RETURN TO PERFORMING FUNCTION
     EXIT.
 WS-GET-SECTION   SECTION.
     MOVE MQENC-NATIVE TO MQMD-ENCODING
     MOVE MQGMO-CONVERT TO  MQGMO-OPTIONS
*                            MQGMO-FAIL-IF-QUIESCING.
     MOVE MQMI-NONE        TO MQMD-MSGID.
     MOVE MQCI-NONE        TO MQMD-CORRELID.
     MOVE MQOO-INPUT-SHARED TO WØ3-OPENOPTIONS.
     MOVE WØØ-QNAME         TO MQOD-OBJECTNAME.
     CALL WS-GET   USING WØ3-HCONN
                         WØ3-HOBJ
                         MQMD
                         MQGMO
                         WØØ-MSGLENGTH
                         WØØ-MSGBUFFER
                         WØØ-DATALENGTH
                         WØ3-COMPCODE
                         WØ3-REASON.
     IF (WØ3-COMPCODE NOT = MQCC-OK) THEN
        MOVE 'MQGET'   TO WØØ-ERROR-MESSAGE
        MOVE WØ3-REASON  TO WS-REASON
        PERFORM DISPLAY-ERROR-MESSAGE
```

```
           PERFORM DØØØØ-MQCLOSE
           EXEC CICS ABEND ABCODE('GERR') END-EXEC
        ELSE
           MOVE WØØ-DATALENGTH TO WS-DATALENGTH
           MOVE WS-DATALENGTH TO  BRMQ-RE-DATA-LEN
           COMPUTE BRMQ-VECTOR-LENGTH = WS-DATALENGTH + 36
           COMPUTE WS-MSGLENGTH = BRMQ-VECTOR-LENGTH + 18Ø
           MOVE WØØ-MSGBUFFER  TO WS-GET-DATA
           MOVE MQMD-MSGID TO WS-MSGID
           MOVE MQMD-CORRELID TO WS-CORRELID
           EXEC CICS WRITEQ TD QUEUE('CSSL') FROM(WS-MSGID)
                LENGTH(24) END-EXEC
           EXEC CICS WRITEQ TD QUEUE('CSSL') FROM(WS-CORRELID)
                LENGTH(24) END-EXEC
           MOVE 'MQGET SUCCESSFUL'   TO WØØ-ERROR-MESSAGE
              EXEC CICS WRITEQ TD QUEUE ('CSSL')
              FROM (WØØ-ERROR-MESSAGE) LENGTH (5Ø)  END-EXEC.
     WS-GET-SECTION-EXIT.
           EXIT.
     DISPLAY-ERROR-MESSAGE SECTION.
        EXEC CICS WRITEQ TD QUEUE('CSSL')
                  FROM(WØØ-ERROR-MESSAGE)
                  LENGTH(5Ø)
                  END-EXEC.
                  MOVE WØ3-REASON TO WS-REASON.
        EXEC CICS WRITEQ TD QUEUE('CSSL')
                  FROM(WS-REASON)
                  LENGTH(1Ø)
                  END-EXEC.
     DISPLAY-ERROR-MESSAGE-END.
    *    RETURN TO PERFORMING FUNCTION
        EXIT.
     DØØØØ-MQCLOSE    SECTION.
        CALL WS-CLOSE USING  WØ3-HCONN
                             WØ3-HOBJ
                             MQCO-NONE
                             WØ3-COMPCODE
                             WØ3-REASON.
        IF (WØ3-COMPCODE NOT = MQCC-OK) THEN
           MOVE 'MQCLOSE'  TO WØØ-ERROR-MESSAGE
           EXEC CICS ABEND ABCODE('CERR') END-EXEC
           EXEC CICS RETURN END-EXEC
        ELSE
           MOVE 'MQCLOSE SUCCESSFUL' TO WØØ-ERROR-MESSAGE
              EXEC CICS WRITEQ TD QUEUE ('CSSL')
              FROM (WØØ-ERROR-MESSAGE) LENGTH(5Ø) END-EXEC.
        EXEC CICS RETURN END-EXEC.
     DØØØØ-MQCLOSE-EXIT.
        EXIT.
     WØØØ-OPEN-BRIDGE   SECTION.
```

```
      MOVE MQOO-OUTPUT TO WBR-OPENOPTIONS.
      MOVE WØØ-BNAME   TO MQOD-OBJECTNAME.
      CALL WS-OPEN  USING WØ3-HCONN
                          MQOD
                          WBR-OPENOPTIONS
                          WBR-HOBJ
                          WØ3-COMPCODE
                          WØ3-REASON.
*     If open failed then display error message
*     and exit.
      IF (WØ3-COMPCODE NOT = MQCC-OK) THEN
         MOVE 'MQOPEN BRIDGE'   TO WØØ-ERROR-MESSAGE
         PERFORM DISPLAY-ERROR-MESSAGE
         EXEC CICS ABEND ABCODE('BROP') END-EXEC
*        GO TO A-MAIN-DISCONNECT
      ELSE
      EXEC CICS WRITEQ TD QUEUE('CSSL')
                       FROM(WS-OP-BRIDGESUCCESSFUL)
                       LENGTH(2Ø)
                       END-EXEC.
 WØØØ-OP-EXIT.
       EXIT.
      EJECT
*                          END OF PROGRAM
```


Message tools in WBI Message Broker V5.0 Toolkit

The Enqueue Message File and Dequeue Message tools in the WebSphere Business Integration (WBI) Message Broker V5.0 Toolkit provide basic interaction with WMQ queue managers and queues on a local system. This article provides an explanation of how to use the tools to maximum advantage.

ENQUEUE MESSAGE FILE

The Enqueue Message File allows the user to put messages on a queue belonging to a particular queue manager. It provides a graphical layout of the task and is quite useful as the user can put messages onto a queue without having to move away from the toolkit.

Before creating the Enqueue Message File make sure there is a project in the workspace because the Enqueue Message File has to be part of a project.

1 Create a new Enqueue Message File either by navigating through the File menu or by clicking on the 'Put a message to a Queue' toolbar button as shown in Figure 1. Click on the drop down button and select 'Put Message…'.

2 In the New Enqueue Message File window that opens, select a project that the Enqueue Message File will belong to, and give the file a name. As Figure 2 illustrates, Enqueue is the name given to the file which will be a part of the Enqueue project. Click Finish.

3 In the workspace a pane with the new Enqueue Message File just created appears. The title of the pane shows the name of the file with the .enqueue extension. As Figure 3 illustrates, enter the values for the various fields as follows:

   – the name of the target queue and queue manager

   – in the message data area enter the data to be put on the queue; you can also use the Browse button to select a file and its contents will appear in the Message data box (note: the file has to be imported into the workspace)

   – the Data format box to the right of the Message data box

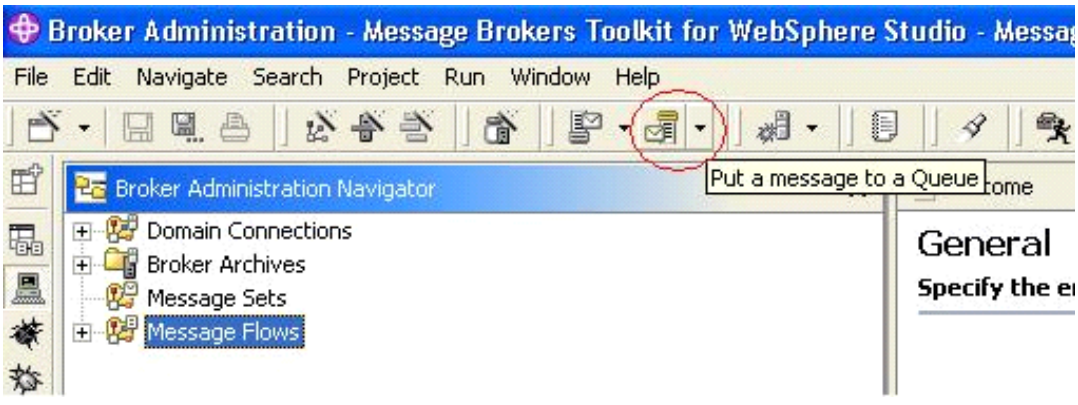

Figure 1: Creating a new Enqueue Message File

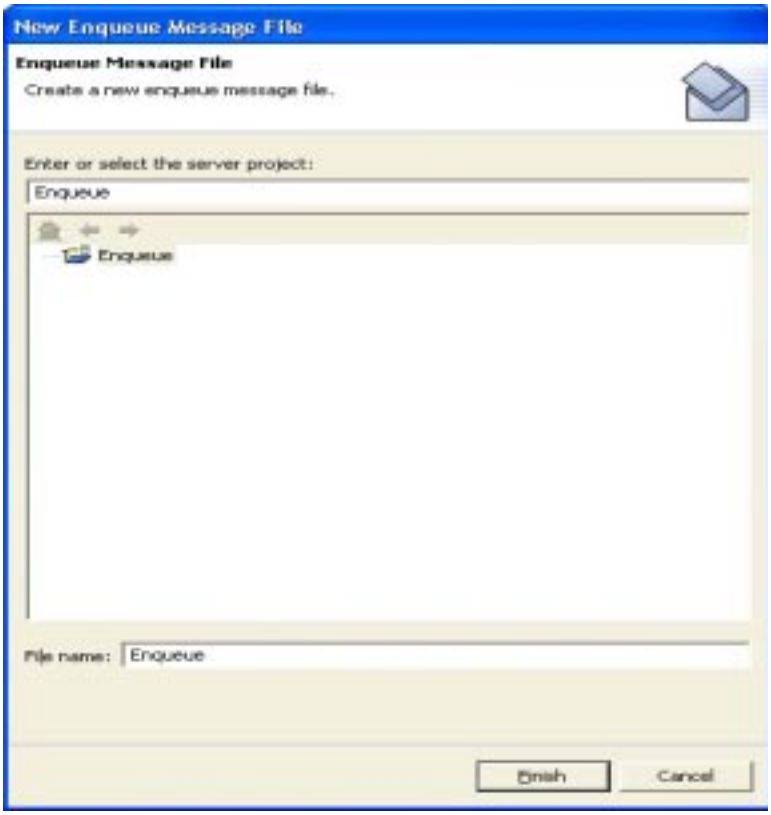

Figure 2: Naming the file that will be part of theEnqueue project

can be toggled to see either Raw or HEX representation of the message data.

4 Once the message has been selected click the 'Write To queue' button in the General tab to put the message to the queue. A dialogue box indicating a successful write to queue will appear.

DEQUEUE MESSAGE

The Dequeue Message tool is used simply to get a message off a queue and display it in the toolkit.

1 Open the Dequeue Message tool by clicking on the 'Get a message from a Queue' button as shown in Figure 4. Click on the drop down button and select 'Get Message…'.

Figure 3: Entering appropriate values for the various fields

2 In the Dequeue Message window that opens, shown in Figure 5, fill in the values of the target queue manager and the name of the queue the messages are to be retrieved from.

3 Click on the 'Read From Queue' button to get the messages off the queue and the message contents will appear in the box underneath. (Note: this tool does not browse but gets a message off a queue, therefore deleting it. Use the 'Save

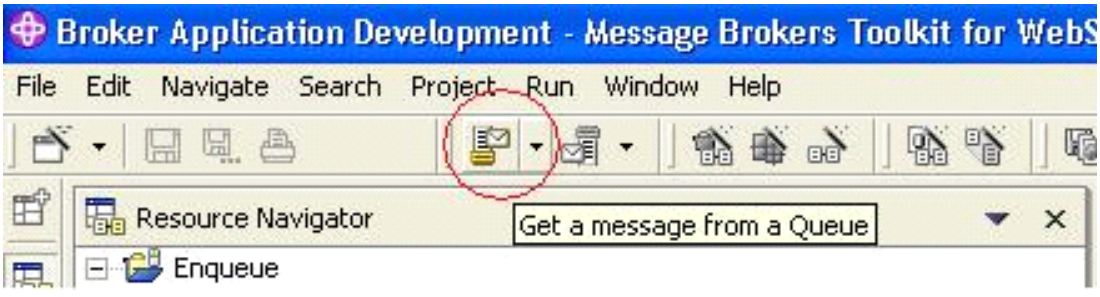

Figure 4: Opening the Dequeue Message tool

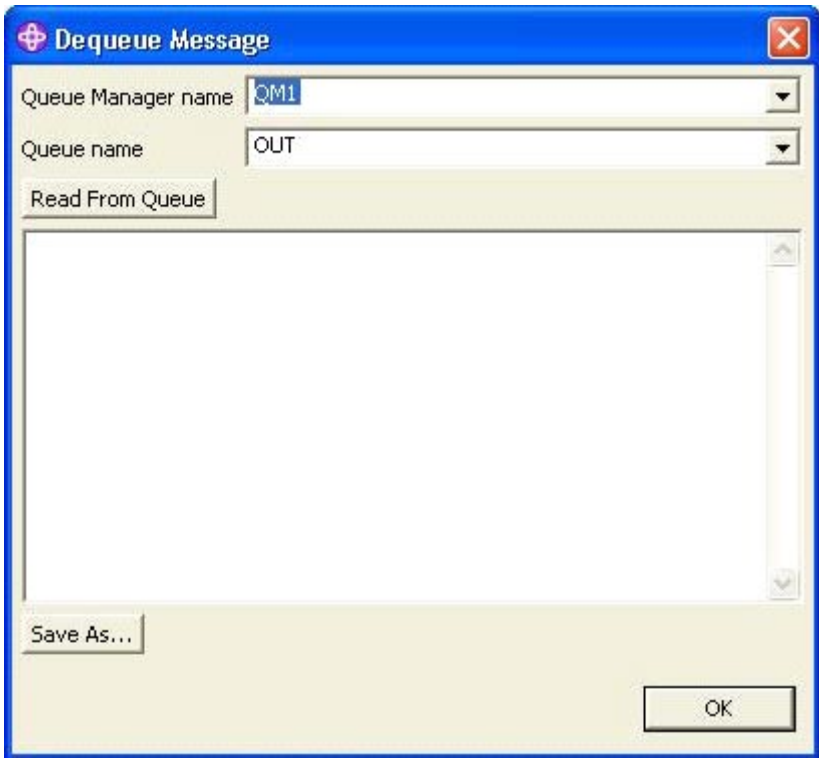

Figure 5: Selecting the queue and queue manager from which to retrieve messages

As…' button to save the message if needed. A dialogue box will appear when the end of the queue being browsed has been reached.)

Note that the Enqueue/Dequeue tools can be used only for local queues and queue managers. Remote queue definition browsing and writing are not supported.

Rohit Bhasin
IBM Hursley (UK)

Contributing to *MQ Update*

In addition to *MQ Update*, the Xephon family of *Update* publications now includes *CICS Update*, *MVS Update*, *TCP/SNA 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.

They have found that the effort needed to pass on valuable information to others is more than offset by our generous terms and conditions and the recognition they gain from their fellow professionals. Often, just a few hundred words are sufficient to describe a problem and the steps taken to solve it.

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.

Cape Clear Software has released Version 4.5 of its Web Services software suite. Cape Clear V4.5 includes three products: Studio, Server, and Manager, which are said to enable the design, development, integration, deployment, and management of Web services-based business services.

New features in V4.5 include asynchronous processing using WebSphere MQ or JMS as a transport for XML messages, notification services, event management, mobile device support, and support for the PARLAY-X telecommunications standard.

For more information contact:
Cape Clear Software, 2929 Campus Drive, Suite 400, San Mateo, CA 94403, USA.
Tel: (888) 227 3439.
Fax: (413) 622 4295.
Web: http://www.capeclear.com

Cape Clear Software, 61 Fitzwilliam Lane, Dublin 2, Ireland.
Tel: +353 1 241 9900.
Fax: +353 1 241 9901.

* * *

InSystems, a solutions provider for document-intensive business processes, has announced the availability of InSystems Calligo Doc Services. The company claims that the new enterprise software platform complements core business systems by managing event-driven document processing.

InSystems Calligo Doc Services can automatically trigger a range of activities, such as retrieval of documents from repositories, assembly of complex documents/packages, rendering to multiple formats, and delivery via multiple channels. The product is claimed to offer seamless integration with WebSphere Application Server and WebSphere MQ.

For more information contact:
InSystems,19 Allstate Parkway, Suite 400, Markham, Ontario, L3R 5A4, Canada.
Tel: (905) 513 1400.
Fax: (905) 513 1419.
Web:http://www.insystems.com

* * *

SpiritSoft has announced the availability of SpiritCache V3.0, offering seamless integration with WebSphere MQ. With its release the company claims that SpiritCache is the first commercially available cacheing product to leverage WMQ to share data and update cache logic across the LAN and WAN. SpiritCache V3.0 is said to provide an enterprise-wide, multi-tier, hierarchical data fabric that extends WMQ functionality to reduce database loading and network bandwidth requirements.

SpiritSoft, 100 Medway Road, Suite 203, Milford, MA 01757, USA.
Tel: +1 508 473 3227.
Fax: +1 508 473 3672.
Web: http://www.spiritsoft.net

SpiritSoft, 10 Eastcheap, London, EC3M 1AJ, UK.
Tel: +44 20 7398 5900.
Fax: +44 20 7398 5901.

* * *