76

MQ update

October 2005

In this issue

3 Automatic starting and stopping of WebSphere InterChange Server connectors

6 Setting up RACF security for WMQ

19 Using timeouts for determinism in IBM WebSphere Business Integration Message Broker flow testing

25 WebSphere Portal installation on z/Linux – part 3

39 WebSphere Business Integration Message Broker – simplified functional validation

51 MQ news

A itc 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.

Automatic starting and stopping of WebSphere InterChange Server connectors

PROBLEM

Our organization uses WebSphere InterChange server Version 4.3.0 on a Windows 2003 server for synchronizing and automating various business activities. We synchronize the updates to our application databases, DB2 on an AS/400 system and SQL Server on Windows using the InterChange Server (ICS) using WebSphere MQ (WMQ) and JDBC connectors.

WMQ Connector reads the MQ messages from MQ queues on Windows 2003. The adapter then transfers the data to ICS for data transformation. Once the data is transformed, the ICS system writes the data to the DB2 database using the JDBC adapter.

Every Sunday morning, the AS/400 system that feeds the ICS system is brought down for IPLing. The IPL (Initial Program Load) process involves shutting down the AS/400 system and then restarting it, and the process often runs for 2–3 hours. During the IPL process, all the JDBC connections are lost, leaving ‘stale’ connections in the memory of the ICS system. Once the AS/400 system comes back on-line, the ICS system tries to make use of these ‘stale’ connections to write to the DB2 database. Since the connection is not right, the records fail to be written to the database, and go to the failed flows. To prevent the records failing, one needs to stop the JDBC connector as the AS/400 is shut down for the IPL process. Once the AS/400 system comes back on-line, the JDBC connector needs to be started again.

Since the IPL process occurs at a pre-determined time on a regular basis, it has been on our wish list to automate the stopping/starting of the JDBC connectors during that time, so that the transactions would not fail because of the stale

database connections. The ICS system comes with a start script for the JDBC connector. However, there is no stop script provided by IBM for stopping the JDBC connector. The only way to stop a connector is to close the command window where the start script runs or type **q** in the command window. Either of these actions needs manual intervention and one needs to be up in the early hours to perform this mundane task.

Our problem was to find a way to automate the stopping/ starting of the JDBC connectors.

SOLUTION

We came up with a novel idea to resolve the problem. The idea is pretty simple and is proving to be quite effective.

When a JDBC connector is started using the start batch script, the Windows operating system creates a process ID (PID). This process ID is used by the operating system to track the

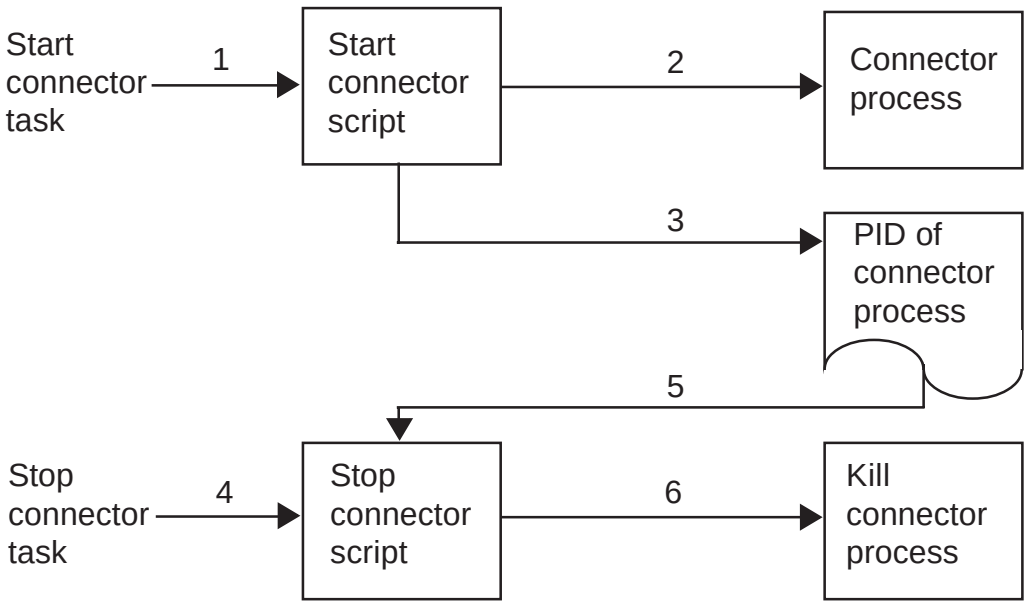

Figure 1: Automatic starting/stopping of ICS connector

script that runs the JDBC connector. A batch script is created to identify the PID and kill the PID of the connector. Now we have two batch scripts – one to start the connector and another to stop it. Using the Windows task scheduler, the stop/start scripts are scheduled to run in sync with the schedule of the IPL process. Figure 1 shows the logic.

A zip file containing the scripts required to set up the above process is available for download from www.xephon.com/extras/ics.zip. I have made use of the following utilities in the process:

- getpids.exe from http://www.scheibli.com/v3/projects/getpids. This utility is used to get the PID of a Windows process.
- Pskill.exe from http://www.sysinternals.com/Utilities/PsKill.html. The utility is used to kill a process given the Windows PID.

The utilities are included in the zip package for the convenience of the user. The user is strongly advised to visit the Web sites that sourced the utilities to find out more about them.

Let us step through the process:

1 Create a Windows task to start the script:

```
<dir>\GetJDBCWriteConnectorPID.bat JDBCWriteConnector
```

2 *GetJDBCWriteConnectorPID.bat* is a DOS batch script and starts the connector passed as an input parameter by calling the script *startJDBCWriteConnector.bat*. After starting the connector, it records the PID of the *GetJDBCWriteConnectorPID.bat* into a text file. The PID is obtained though a call to the *getPID.exe*.

Usage:

```
GetJDBCWriteConnectorPID.bat <connector name>
```

For example:

```
GetJDBCWriteConnectorPID.bat JDBCWriteConnector
```

This script starts the JDBCWriteConnector and writes the PID of the batch file to the *JDBCWriteConnector.txt* file. Later we will make use of this file to stop the connector process by killing the PID through the *KillJDBCWriteConnectorPID.bat* script.

3 Create a Windows task to start the script:

```
<dir>\ \KillJDBCWriteConnectorPID.bat JDBCWriteConnector.
```

4 *KillJDBCWriteConnectorPID.bat* is a DOS batch script that kills the connector indicated in the input parameter. The script reads the PID of the parent batch script that initiated the connector process. Then it kills the process by using the PSKILL utility.

Usage:

```
KillJDBCWriteConnectorPID.bat <connector name>
```

For example:

```
KillJDBCWriteConnectorPID.bat JDBCWriteConnector
```

The script reads the PID of the connector process (recorded in step 2) from the *JDBCWriteConnector.txt* file and kills the process by calling the PSKILL utility.

All of the code is in the ZIP file ics.zip.

Padmasree Upadhyayula
Programmer/Analyst
USXPRESS Enterprises (USA)

Setting up RACF security for WMQ

As with most things, setting up RACF security within WebSphere MQ is quite straightforward – once you know how and where to start. The IBM documentation (*WebSphere MQ for z/OS System Setup Guide*) is very good, but MQ security

is implemented in some unusual ways in RACF and you need to think about naming conventions. Even with proper naming conventions, you will end up with quite a lot of RACF resource profiles – but without them, you will not be able to manage security, especially if you have several queue managers.

It is necessary to read *Part 5* of the *System Setup Guide*, but, together with the official documentation, this little article might help you to get a better understanding of what is important to think about before you start setting up security. In addition, it shows one possible way to implement security and you can use the templates below, as they are, to get started straight away.

NAMING CONVENTIONS

Naming standards are most essential to make your RACF profiles manageable and reduce their number to a minimum. As with dataset names, it is good practice to organize the MQSeries object's names hierarchically to allow the use of generic profiles. What is important is not what the qualifiers within the object names stand for (for example organizational units, application names, or function names), but that these qualifiers are built in a hierarchical manner. The following naming conventions have proved to work quite well, but you might have to adjust them to reflect the individual needs of your organization:

- Queues:

  ```
  Business-Area.Application.Qx.free-format
  ```

  where *x* represents the type of queue:

  – L – local queue

  – A – alias queue

  – R – remote queue

  – M – model queue

  – I – initiation (trigger) queue.

- Processes:

  ```
  Business-Area.Application.PR.free-format
  ```

- Namelists:

  ```
  Business-Area.Application.NL.free-format
  ```

- Channels:

  ```
  Sender-Queue-Manager.Receiver-Queue-Manager
  ```

These object types are the most important from a security point of view. The first two qualifiers allow differentiation between the business areas and potentially between the applications within them. When starting from scratch it is advisable to make the distinction, from a security point of view, on only the first qualifier. Nevertheless, define at least two qualifiers in front of the queue type qualifier in your naming standards and add more if you need to. Using generic RACF profiles does not necessarily result in more RACF profiles, but does make the management and usually the monitoring of MQSeries objects easier. Regardless of the number of qualifiers you define, the principle shown in the RACF templates below will not change.

Differentiating between types of queues (Qx) does allow you to give different permissions for different queue types. For example, it is good practice to define alias queues for each physical queue and allow users to access only the alias queues so that you can protect the underlying physical implementation, which makes the management easier.

Other MQSeries objects such as storage classes or buffer pools are relevant only for checking MQ commands issued against them. Because only one administrator group usually manages this, one generic RACF profile is sufficient to protect these objects and the naming convention is not so important. Naming the transmission queues after the receiving queue managers allows you to protect them with generic profiles as well.

As well as naming standards for the MQSeries objects, you

need to organize the users in RACF groups and permit access rights only to RACF groups rather than to individual userids. The following RACF groups are used in this implementation:

- WMQADM – connect the MQSeries administrator userids to this group. This group will have the highest level of access.
- WMQSYS – connect all system users such as started task users of MQSeries, CICS, automation products, scheduling products, etc that cannot be used for log-on from individuals. This group will also have a very high access level, since with RESLEVEL access NONE these userids are checked in addition to the individual userids and hence must have the complete set of rights that all other userids have.
- WMQOPR – connect all operator userids to this group. Depending on your organization, Help Desk or first/second line support userids have to be connected as well. This group has rights to operate MQSeries, for example start/stop channels, but has no access to messages and cannot modify objects.
- WMQBusiness-Area – connect to these groups all userids that are using the specific applications in their business areas. These groups have access to the MQSeries objects and messages within their area, but the access level can vary for different environments such as development, test, or production. Theoretically, the product of Business-Area.Application is the number of RACF groups you need to define but, as mentioned above, when starting from scratch you should differentiate on only the first qualifier (Business-Area), so you need to define as many RACF groups as there are different Business-Areas.

ACTIVATE SECURITY

Once per RACF system you have to activate the RACF classes used for MQSeries and make sure that you can use

generic profiles. These are the necessary RACF commands:

```
SETR  CLASSACT(MQADMIN,MQCMDS,MQQUEUE,MQCONN,MQPROC,MQNLIST)
SETR  GENERIC(MQADMIN,MQCMDS,MQQUEUE,MQCONN,MQPROC,MQNLIST)
```

QUEUE MANAGER-LEVEL SECURITY

The first template shows all the RACF commands you need to set up security once for every queue manager regardless of any application. The following assumptions have been made:

- The queue manager name in this sample is QX00.
- The security is only at a queue manager level and there is no queue-sharing group defined.
- No switch profiles have been defined, hence resource checking for all types of MQSeries object is activated.
- All objects are protected. There is at least one generic profile covering all objects of a certain type. For some objects, profiles that are more specific are defined.
- All userid checks are made at the most restrictive level (RESLEVEL access NONE).
- There is no distinction between different types of connections. To connect to the queue manager a userid just has to be connected to one of the WMQ*xxx* groups, regardless of whether connecting from batch, CICS, IMS, etc.
- Commands are checked at the command level as well as at the command resource level. There is at least one generic profile covering all objects of a certain type, and, for some objects, profiles that are more specific are defined.
- The dead-letter queue is called QX00.DEAD.QUEUE, which is an alias queue pointing to QX00.DLQ. Everybody can write to QX00.DEAD.QUEUE, but this queue is GET DISABLED. QX00.DLQ is GET ENABLED, but only MQSeries administrators have update access to this

queue. For further information about dead-letter queue security, see Chapter 13 in the *System Setup Guide*.

```
/*************************************************************/
/* RESOURCE LEVEL PROFILE
/*************************************************************/

RDEFINE MQADMIN QX00.RESLEVEL UACC(NONE)

/*************************************************************/
/* CLASS: MQADMIN
**************************************************************/

/* CONTEXT                                                   */

RDEFINE MQADMIN QX00.CONTEXT.** UACC(NONE)
PERMIT QX00.CONTEXT.** ACCESS(CONTROL) CLASS(MQADMIN) ID(WMQSYS)
PERMIT QX00.CONTEXT.** ACCESS(CONTROL) CLASS(MQADMIN) ID(WMQADM)

/* ALTERNATE USERID                                          */

RDEFINE MQADMIN QX00.ALTERNATE.USER.* UACC(NONE)
PERMIT QX00.ALTERNATE.USER.* ACCESS(UPDATE) CLASS(MQADMIN) ID(WMQSYS)
PERMIT QX00.ALTERNATE.USER.* ACCESS(UPDATE) CLASS(MQADMIN) ID(WMQADM)

/* COMMAND RESOURCE SECURITY                                 */

RDEFINE MQADMIN QX00.AUTHINFO.** UACC(NONE)
PERMIT QX00.AUTHINFO.** ACCESS(ALTER) CLASS(MQADMIN) ID(WMQSYS)
PERMIT QX00.AUTHINFO.** ACCESS(ALTER) CLASS(MQADMIN) ID(WMQADM)

RDEFINE MQADMIN QX00.CHANNEL.** UACC(NONE)
PERMIT QX00.CHANNEL.** ACCESS(ALTER) CLASS(MQADMIN) ID(WMQSYS)
PERMIT QX00.CHANNEL.** ACCESS(ALTER) CLASS(MQADMIN) ID(WMQADM)
PERMIT QX00.CHANNEL.** ACCESS(ALTER) CLASS(MQADMIN) ID(WMQOPR)

RDEFINE MQADMIN QX00.QUEUE.** UACC(NONE)
PERMIT QX00.QUEUE.** ACCESS(ALTER) CLASS(MQADMIN) ID(WMQSYS)
PERMIT QX00.QUEUE.** ACCESS(ALTER) CLASS(MQADMIN) ID(WMQADM)

RDEFINE MQADMIN QX00.PROCESS.** UACC(NONE)
PERMIT QX00.PROCESS.** ACCESS(ALTER) CLASS(MQADMIN) ID(WMQSYS)
PERMIT QX00.PROCESS.** ACCESS(ALTER) CLASS(MQADMIN) ID(WMQADM)

RDEFINE MQADMIN QX00.NAMELIST.** UACC(NONE)
PERMIT QX00.NAMELIST.** ACCESS(ALTER) CLASS(MQADMIN) ID(WMQSYS)
PERMIT QX00.NAMELIST.** ACCESS(ALTER) CLASS(MQADMIN) ID(WMQADM)

/*************************************************************/
/* CLASS: MQCMDS
```

```
/*************************************************************/

RDEFINE MQCMDS QX00.** UACC(NONE)
PERMIT QX00.** ACCESS(ALTER) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.** ACCESS(ALTER) CLASS(MQCMDS) ID(WMQADM)

RDEFINE MQCMDS QX00.ALTER.** UACC(NONE)
PERMIT QX00.ALTER.** ACCESS(ALTER) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.ALTER.** ACCESS(ALTER) CLASS(MQCMDS) ID(WMQADM)
RDEFINE MQCMDS QX00.ALTER.PROCESS UACC(NONE)
PERMIT QX00.ALTER.PROCESS ACCESS(ALTER) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.ALTER.PROCESS ACCESS(ALTER) CLASS(MQCMDS) ID(WMQADM)
RDEFINE MQCMDS QX00.ALTER.QALIAS UACC(NONE)
PERMIT QX00.ALTER.QALIAS ACCESS(ALTER) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.ALTER.QALIAS ACCESS(ALTER) CLASS(MQCMDS) ID(WMQADM)
RDEFINE MQCMDS QX00.ALTER.QLOCAL UACC(NONE)
PERMIT QX00.ALTER.QLOCAL ACCESS(ALTER) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.ALTER.QLOCAL ACCESS(ALTER) CLASS(MQCMDS) ID(WMQADM)
RDEFINE MQCMDS QX00.ALTER.QMODEL UACC(NONE)
PERMIT QX00.ALTER.QMODEL ACCESS(ALTER) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.ALTER.QMODEL ACCESS(ALTER) CLASS(MQCMDS) ID(WMQADM)
RDEFINE MQCMDS QX00.ALTER.QREMOTE UACC(NONE)
PERMIT QX00.ALTER.QREMOTE ACCESS(ALTER) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.ALTER.QREMOTE ACCESS(ALTER) CLASS(MQCMDS) ID(WMQADM)

RDEFINE MQCMDS QX00.ARCHIVE.** UACC(NONE)
PERMIT QX00.ARCHIVE.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.ARCHIVE.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQADM)
PERMIT QX00.ARCHIVE.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQOPR)

RDEFINE MQCMDS QX00.BACKUP.** UACC(NONE)
PERMIT QX00.BACKUP.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.BACKUP.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQADM)

RDEFINE MQCMDS QX00.CLEAR.** UACC(NONE)
PERMIT QX00.CLEAR.** ACCESS(ALTER) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.CLEAR.** ACCESS(ALTER) CLASS(MQCMDS) ID(WMQADM)

RDEFINE MQCMDS QX00.DEFINE.** UACC(NONE)
PERMIT QX00.DEFINE.** ACCESS(ALTER) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.DEFINE.** ACCESS(ALTER) CLASS(MQCMDS) ID(WMQADM)
RDEFINE MQCMDS QX00.DEFINE.PROCESS UACC(NONE)
PERMIT QX00.DEFINE.PROCESS ACCESS(ALTER) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.DEFINE.PROCESS ACCESS(ALTER) CLASS(MQCMDS) ID(WMQADM)
RDEFINE MQCMDS QX00.DEFINE.QALIAS UACC(NONE)
PERMIT QX00.DEFINE.QALIAS ACCESS(ALTER) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.DEFINE.QALIAS ACCESS(ALTER) CLASS(MQCMDS) ID(WMQADM)
RDEFINE MQCMDS QX00.DEFINE.QLOCAL UACC(NONE)
PERMIT QX00.DEFINE.QLOCAL ACCESS(ALTER) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.DEFINE.QLOCAL ACCESS(ALTER) CLASS(MQCMDS) ID(WMQADM)
```

```
RDEFINE MQCMDS QX00.DEFINE.QMODEL UACC(NONE)
PERMIT QX00.DEFINE.QMODEL ACCESS(ALTER) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.DEFINE.QMODEL ACCESS(ALTER) CLASS(MQCMDS) ID(WMQADM)
RDEFINE MQCMDS QX00.DEFINE.QREMOTE UACC(NONE)
PERMIT QX00.DEFINE.QREMOTE ACCESS(ALTER) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.DEFINE.QREMOTE ACCESS(ALTER) CLASS(MQCMDS) ID(WMQADM)

RDEFINE MQCMDS QX00.DELETE.** UACC(NONE)
PERMIT QX00.DELETE.** ACCESS(ALTER) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.DELETE.** ACCESS(ALTER) CLASS(MQCMDS) ID(WMQADM)
RDEFINE MQCMDS QX00.DELETE.PROCESS UACC(NONE)
PERMIT QX00.DELETE.PROCESS ACCESS(ALTER) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.DELETE.PROCESS ACCESS(ALTER) CLASS(MQCMDS) ID(WMQADM)
RDEFINE MQCMDS QX00.DELETE.QALIAS UACC(NONE)
PERMIT QX00.DELETE.QALIAS ACCESS(ALTER) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.DELETE.QALIAS ACCESS(ALTER) CLASS(MQCMDS) ID(WMQADM)
RDEFINE MQCMDS QX00.DELETE.QLOCAL UACC(NONE)
PERMIT QX00.DELETE.QLOCAL ACCESS(ALTER) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.DELETE.QLOCAL ACCESS(ALTER) CLASS(MQCMDS) ID(WMQADM)
RDEFINE MQCMDS QX00.DELETE.QMODEL UACC(NONE)
PERMIT QX00.DELETE.QMODEL ACCESS(ALTER) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.DELETE.QMODEL ACCESS(ALTER) CLASS(MQCMDS) ID(WMQADM)
RDEFINE MQCMDS QX00.DELETE.QREMOTE UACC(NONE)
PERMIT QX00.DELETE.QREMOTE ACCESS(ALTER) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.DELETE.QREMOTE ACCESS(ALTER) CLASS(MQCMDS) ID(WMQADM)

RDEFINE MQCMDS QX00.DISPLAY.** UACC(NONE)
PERMIT QX00.DISPLAY.** ACCESS(READ) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.DISPLAY.** ACCESS(READ) CLASS(MQCMDS) ID(WMQADM)
PERMIT QX00.DISPLAY.** ACCESS(READ) CLASS(MQCMDS) ID(WMQOPR)

RDEFINE MQCMDS QX00.MOVE.** UACC(NONE)
PERMIT QX00.MOVE.** ACCESS(ALTER) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.MOVE.** ACCESS(ALTER) CLASS(MQCMDS) ID(WMQADM)

RDEFINE MQCMDS QX00.PING.** UACC(NONE)
PERMIT QX00.PING.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.PING.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQADM)
PERMIT QX00.PING.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQOPR)

RDEFINE MQCMDS QX00.RECOVER.** UACC(NONE)
PERMIT QX00.RECOVER.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.RECOVER.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQADM)

RDEFINE MQCMDS QX00.REFRESH.** UACC(NONE)
PERMIT QX00.REFRESH.** ACCESS(ALTER) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.REFRESH.** ACCESS(ALTER) CLASS(MQCMDS) ID(WMQADM)

RDEFINE MQCMDS QX00.RESET.** UACC(NONE)
PERMIT QX00.RESET.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQSYS)
```

```
PERMIT QX00.RESET.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQADM)
RDEFINE MQCMDS QX00.RESET.QSTATS UACC(NONE)
PERMIT QX00.RESET.QSTATS ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.RESET.QSTATS ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQADM)

RDEFINE MQCMDS QX00.RESOLVE.** UACC(NONE)
PERMIT QX00.RESOLVE.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.RESOLVE.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQADM)

RDEFINE MQCMDS QX00.RESUME.** UACC(NONE)
PERMIT QX00.RESUME.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.RESUME.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQADM)

RDEFINE MQCMDS QX00.RVERIFY.** UACC(NONE)
PERMIT QX00.RVERIFY.** ACCESS(ALTER) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.RVERIFY.** ACCESS(ALTER) CLASS(MQCMDS) ID(WMQADM)

RDEFINE MQCMDS QX00.SET.** UACC(NONE)
PERMIT QX00.SET.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.SET.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQADM)

RDEFINE MQCMDS QX00.START.** UACC(NONE)
PERMIT QX00.START.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.START.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQADM)
PERMIT QX00.START.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQOPR)
RDEFINE MQCMDS QX00.START.CHANNEL UACC(NONE)
PERMIT QX00.START.CHANNEL ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.START.CHANNEL ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQADM)
PERMIT QX00.START.CHANNEL ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQOPR)

RDEFINE MQCMDS QX00.STOP.** UACC(NONE)
PERMIT QX00.STOP.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.STOP.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQADM)
PERMIT QX00.STOP.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQOPR)
RDEFINE MQCMDS QX00.STOP.CHANNEL UACC(NONE)
PERMIT QX00.STOP.CHANNEL ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.STOP.CHANNEL ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQADM)
PERMIT QX00.STOP.CHANNEL ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQOPR)

RDEFINE MQCMDS QX00.SUSPEND.** UACC(NONE)
PERMIT QX00.SUSPEND.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQSYS)
PERMIT QX00.SUSPEND.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQADM)

/*************************************************************/
/* CLASS: MQQUEUE
/*************************************************************/

RDEFINE MQQUEUE QX00.** UACC(NONE)
PERMIT QX00.** ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQSYS)
PERMIT QX00.** ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQADM)
```

```
RDEFINE MQQUEUE QX00.QX00.DLQ UACC(NONE)
PERMIT QX00.QX00.DLQ ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQSYS)
PERMIT QX00.QX00.DLQ ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQADM)

RDEFINE MQQUEUE QX00.QX00.DEAD.QUEUE UACC(NONE)
PERMIT QX00.QX00.DEAD.QUEUE ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQSYS)
PERMIT QX00.QX00.DEAD.QUEUE ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQADM)

RDEFINE MQQUEUE QX00.SYSTEM.** UACC(NONE)
PERMIT QX00.SYSTEM.** ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQSYS)
PERMIT QX00.SYSTEM.** ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQADM)

RDEFINE MQQUEUE QX00.SYSTEM.COMMAND.** UACC(NONE)
PERMIT QX00.SYSTEM.COMMAND.** ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQSYS)
PERMIT QX00.SYSTEM.COMMAND.** ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQADM)
PERMIT QX00.SYSTEM.COMMAND.** ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQOPR)

RDEFINE MQQUEUE QX00.SYSTEM.CSQOREXX.** UACC(NONE)
PERMIT QX00.SYSTEM.CSQOREXX.** ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQSYS)
PERMIT QX00.SYSTEM.CSQOREXX.** ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQADM)
PERMIT QX00.SYSTEM.CSQOREXX.** ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQOPR)

RDEFINE MQQUEUE QX00.SYSTEM.CSQUTIL.** UACC(NONE)
PERMIT QX00.SYSTEM.CSQUTIL.** ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQSYS)
PERMIT QX00.SYSTEM.CSQUTIL.** ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQADM)
PERMIT QX00.SYSTEM.CSQUTIL.** ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQOPR)

/*************************************************************/
/* CLASS: MQCONN
/*************************************************************/

RDEFINE MQCONN QX00.* UACC(NONE)
PERMIT QX00.* ACCESS(READ) CLASS(MQCONN) ID(WMQSYS)
PERMIT QX00.* ACCESS(READ) CLASS(MQCONN) ID(WMQADM)
PERMIT QX00.* ACCESS(READ) CLASS(MQCONN) ID(WMQOPR)

/*************************************************************/
/* CLASS: MQPROC
/*************************************************************/

RDEFINE MQPROC QX00.* UACC(NONE)
PERMIT QX00.* ACCESS(READ) CLASS(MQPROC) ID(WMQSYS)
PERMIT QX00.* ACCESS(READ) CLASS(MQPROC) ID(WMQADM)

/*************************************************************/
/* CLASS: MQNLIST
/*************************************************************/

RDEFINE MQNLIST QX00.* UACC(NONE)
```

```
PERMIT QX00.* ACCESS(READ) CLASS(MQNLIST) ID(WMQSYS)
PERMIT QX00.* ACCESS(READ) CLASS(MQNLIST) ID(WMQADM)
```

For further queue managers, just copy the statements above and change the queue manager name. You might want to add permits for group WMQOPR for some other commands (class MQCMDS) in case the operators should be allowed to carry out other tasks (for example SUSPEND/RESUME).

APPLICATION-LEVEL SECURITY

The second template shows the RACF commands for a Business-Area, in this case called IVP. IVP contains a set of queues, processes, channels, etc used for initial checks and testing. In addition, this makes it possible to check whether security is set up correctly. The following assumptions have been made:

- There is no distinction on the level of the second qualifier (Application).
- Users connected to group WMQIVP cannot use ALTERNATE USER.
- Users connected to group WMQIVP can connect to QX00 from all subsystems, but can only access objects starting with IVP.

(This sample is typical for a development environment because the users can define, delete, and alter their MQSeries objects. They can read from all their queues and update alias queues.)

```
/*************************************************************/
/* CLASS: MQADMIN
/*************************************************************/

RDEFINE MQADMIN QX00.CHANNEL.QX*.QX*.** UACC(NONE)
PERMIT QX00.CHANNEL.QX*.QX*.** ACCESS(ALTER) CLASS(MQADMIN) ID(WMQSYS)
PERMIT QX00.CHANNEL.QX*.QX*.** ACCESS(ALTER) CLASS(MQADMIN) ID(WMQADM)
PERMIT QX00.CHANNEL.QX*.QX*.** ACCESS(CONTROL) CLASS(MQADMIN) ID(WMQIVP)

RDEFINE MQADMIN QX00.QUEUE.IVP.*.Q%.** UACC(NONE)
PERMIT QX00.QUEUE.IVP.*.Q%.** ACCESS(ALTER) CLASS(MQADMIN) ID(WMQSYS)
PERMIT QX00.QUEUE.IVP.*.Q%.** ACCESS(ALTER) CLASS(MQADMIN) ID(WMQADM)
```

```
PERMIT QX00.QUEUE.IVP.*.Q%.** ACCESS(READ) CLASS(MQADMIN) ID(WMQIVP)

RDEFINE MQADMIN QX00.PROCESS.IVP.*.PR.** UACC(NONE)
PERMIT QX00.PROCESS.IVP.*.PR.** ACCESS(ALTER) CLASS(MQADMIN) ID(WMQSYS)
PERMIT QX00.PROCESS.IVP.*.PR.** ACCESS(ALTER) CLASS(MQADMIN) ID(WMQADM)
PERMIT QX00.PROCESS.IVP.*.PR.** ACCESS(ALTER) CLASS(MQADMIN) ID(WMQIVP)

RDEFINE MQADMIN QX00.NAMELIST.IVP.*.NL.** UACC(NONE)
PERMIT QX00.NAMELIST.IVP.*.NL.** ACCESS(ALTER) CLASS(MQADMIN) ID(WMQSYS)
PERMIT QX00.NAMELIST.IVP.*.NL.** ACCESS(ALTER) CLASS(MQADMIN) ID(WMQADM)
PERMIT QX00.NAMELIST.IVP.*.NL.** ACCESS(ALTER) CLASS(MQADMIN) ID(WMQIVP)
/*************************************************************/
/* CLASS: MQCONN
/*************************************************************/

PERMIT QX00.* ACCESS(READ) CLASS(MQCONN) ID(WMQIVP)

/*************************************************************/
/* CLASS: MQCMDS
/*************************************************************/

PERMIT QX00.ALTER.PROCESS ACCESS(ALTER) CLASS(MQCMDS) ID(WMQIVP)
PERMIT QX00.ALTER.QALIAS ACCESS(ALTER) CLASS(MQCMDS) ID(WMQIVP)
PERMIT QX00.ALTER.QLOCAL ACCESS(ALTER) CLASS(MQCMDS) ID(WMQIVP)
PERMIT QX00.ALTER.QMODEL ACCESS(ALTER) CLASS(MQCMDS) ID(WMQIVP)
PERMIT QX00.ALTER.QREMOTE ACCESS(ALTER) CLASS(MQCMDS) ID(WMQIVP)

PERMIT QX00.DEFINE.PROCESS ACCESS(ALTER) CLASS(MQCMDS) ID(WMQIVP)
PERMIT QX00.DEFINE.QALIAS ACCESS(ALTER) CLASS(MQCMDS) ID(WMQIVP)
PERMIT QX00.DEFINE.QLOCAL ACCESS(ALTER) CLASS(MQCMDS) ID(WMQIVP)
PERMIT QX00.DEFINE.QMODEL ACCESS(ALTER) CLASS(MQCMDS) ID(WMQIVP)
PERMIT QX00.DEFINE.QREMOTE ACCESS(ALTER) CLASS(MQCMDS) ID(WMQIVP)

PERMIT QX00.DELETE.PROCESS ACCESS(ALTER) CLASS(MQCMDS) ID(WMQIVP)
PERMIT QX00.DELETE.QALIAS ACCESS(ALTER) CLASS(MQCMDS) ID(WMQIVP)
PERMIT QX00.DELETE.QLOCAL ACCESS(ALTER) CLASS(MQCMDS) ID(WMQIVP)
PERMIT QX00.DELETE.QMODEL ACCESS(ALTER) CLASS(MQCMDS) ID(WMQIVP)
PERMIT QX00.DELETE.QREMOTE ACCESS(ALTER) CLASS(MQCMDS) ID(WMQIVP)

PERMIT QX00.CLEAR.** ACCESS(ALTER) CLASS(MQCMDS) ID(WMQIVP)

PERMIT QX00.DISPLAY.** ACCESS(READ) CLASS(MQCMDS) ID(WMQIVP)

PERMIT QX00.MOVE.** ACCESS(ALTER) CLASS(MQCMDS) ID(WMQIVP)

PERMIT QX00.PING.** ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQIVP)

PERMIT QX00.RESET.QSTATS ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQIVP)

PERMIT QX00.START.CHANNEL ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQIVP)
```

```
PERMIT QX00.STOP.CHANNEL ACCESS(CONTROL) CLASS(MQCMDS) ID(WMQIVP)

/*************************************************************/
/* CLASS: MQQUEUE
/*************************************************************/

PERMIT QX00.QX00.DLQ ACCESS(READ) CLASS(MQQUEUE) ID(WMQIVP)
PERMIT QX00.QX00.DEAD.QUEUE ACCESS(UPDATE) CLASS(MQQUEUE) ID(WMQIVP)
PERMIT QX00.SYSTEM.COMMAND.** ACCESS(UPDATE) CLASS(MQQUEUE) ID(WMQIVP)
PERMIT QX00.SYSTEM.CSQOREXX.** ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQIVP)
PERMIT QX00.SYSTEM.CSQUTIL.** ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQIVP)

RDEFINE MQQUEUE QX00.IVP.*.Q%.** UACC(NONE)
PERMIT QX00.IVP.*.Q%.** ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQSYS)
PERMIT QX00.IVP.*.Q%.** ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQADM)
PERMIT QX00.IVP.*.Q%.** ACCESS(READ) CLASS(MQQUEUE) ID(WMQIVP)

RDEFINE MQQUEUE QX00.IVP.*.QA.** UACC(NONE)
PERMIT QX00.IVP.*.QA.** ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQSYS)
PERMIT QX00.IVP.*.QA.** ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQADM)
PERMIT QX00.IVP.*.QA.** ACCESS(ALTER) CLASS(MQQUEUE) ID(WMQIVP)
/*************************************************************/
/* CLASS: MQPROC
/*************************************************************/
RDEFINE MQPROC QX00.IVP.*.PR.** UACC(NONE)
PERMIT QX00.IVP.*.PR.** ACCESS(READ) CLASS(MQPROC) ID(WMQSYS)
PERMIT QX00.IVP.*.PR.** ACCESS(READ) CLASS(MQPROC) ID(WMQADM)
PERMIT QX00.IVP.*.PR.** ACCESS(READ) CLASS(MQPROC) ID(WMQIVP)
/*************************************************************/
/* CLASS: MQNLIST
/*************************************************************/
RDEFINE MQNLIST QX00.IVP.*.NL.** UACC(NONE)
PERMIT QX00.IVP.*.NL.** ACCESS(READ) CLASS(MQNLIST) ID(WMQSYS)
PERMIT QX00.IVP.*.NL.** ACCESS(READ) CLASS(MQNLIST) ID(WMQADM)
PERMIT QX00.IVP.*.NL.** ACCESS(READ) CLASS(MQNLIST) ID(WMQIVP)
```

For further Business-Areas, just copy the definition from IVP and rename IVP with the appropriate Business-Area name. In addition, it is quite easy to permit different access levels for different environments – you just need to change the access rights in the four lines in italics above. For example, you can give ALTER or UPDATE access for objects in development, READ in acceptance test, and NONE in production.

Do not forget that the RACF resource classes MQADMIN, MQNLIST, MQPROC, and MQQUEUE are RACLISTed so you

need to refresh them in RACF when making changes. Also, refresh security or verify security in MQSeries when changing profiles or access rights for users.

Gerald Stark
Mainframe Technical Specialist
MQ Architect (UK)

Using timeouts for determinism in IBM WebSphere Business Integration Message Broker flow testing

INTRODUCTION

IBM WebSphere Business Integration Message Broker can be used to build flows that are highly complex, contain multiple branches, and involve the serial processing of multiple messages, as part of the business method. Testing and debugging these flows can prove difficult, particularly as test cases need to be deterministic and repeatable. When multiple messages are involved, an important part of the testing is to ensure that the flow works correctly despite the order in which the messages are processed by the flow.

The new Timeout nodes provided with Version 6.0 of WebSphere Business Integration Message Broker can be used in simple message flows to implement these tests – where determinism in the order of message delivery is required.

THE TIMEOUT NODES

The Timeout Control and Notification nodes (see Figure 1) are used to implement flows where some kind of delay is required. The Control node receives incoming messages that contain a fixed-format timeout request (see below), which it validates

Figure 1: Timeout Control and Notification nodes

before storing the message internally in the broker. The Notification node (which is associated with the Control node with a unique identifier) is an input node, and reads these stored request messages and propagates them when the timeout expires.

The timeout requests can be configured for one or multiple messages, starting at either relative or absolute times. In the case of multiple timeouts, a count and interval are specified. The Control node can read the timeout request either from a default location in the LocalEnvironment or from a specified location within the message body (configured as a property on the node).

A WORKED EXAMPLE

The simplest example of a multiple-message processing flow is an Aggregation fan-in flow. This can be implemented completely within the message flow environment, using only the supplied built-in nodes. Aggregation is implemented by several nodes, and is an extension of the simple request-reply model, where a request message is split across multiple request-reply applications, and all the resulting replies aggregated into a single reply message. Aggregation usually consists of two flows:

- A fan-out flow, where the request is split into multiple request messages for the individual request-reply applications.
- A fan-in flow, where the resulting replies are received and combined into a single aggregated reply.

In this example, a further flow is needed. This is the test flow (for the fan-in flow) containing the Timeout nodes, and it simulates the back-end request-reply applications. Given that this example is a three-way aggregation, to completely unit test the fan-in flow there need to be six of these request-reply flows. This example shows only one of these because the changes to create the other five are trivial. (For completeness, a brief summary of the fan-out and fan-in flows is included below.)

The request-reply test flow would normally be very simple – MQInput nodes wired directly into an MQReply node. To use the Timeout nodes, some additional flow design is required, as shown in Figure 2.

Each MQInput node feeds into a Compute node, which sets a different timeout request in the LocalEnvironment. The Timeout Control node has all its default settings except for its 'Unique Identifier', which is set to 'TestWait' in this case. The Timeout Notification node processes the requests stored by the Control

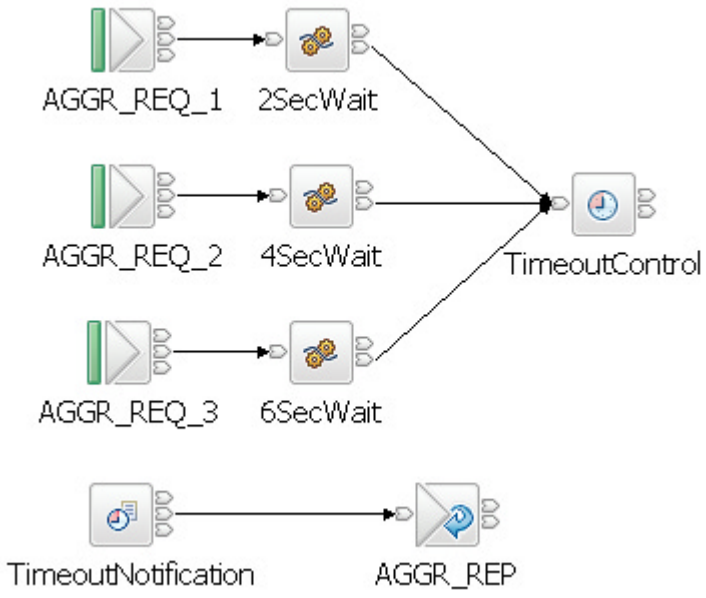

Figure 2: The request-reply Timeout flow

node, and has the following configuration settings:

- 'Unique Identifier' set to 'TestWait' (to associate the two Timeout nodes together).
- 'Operation Mode' set to 'Controlled' (so the Notification node receives the timeout requests set by the Control node).

The ESQL in the Compute nodes is very simple, and the code for the two-second wait is shown in Figure 3. An important point to note is that the 'Identifier' of the timeout request must be unique within the scope of the Timeout nodes, so the four-second and six-second waits need different values (as well as a different interval).

The end result of the above flow is that the three messages that are passed through the flow are predictably staged at two-second intervals (rather than all being processed immediately and the arrival order not being determined). A suite of unit test flows similar to this will ensure the complete testing of the driven flow (the Aggregation fan-in flow) by exercising all logic paths.

There is additional capability in the Timeout nodes that has not been discussed here, including the following:

- Storing part of the incoming message instead of the entire message.
- Reading the timeout request from within the message body (in the example above, the default location of the LocalEnvironment has been used).
- Over-writing and cancelling stored timeout requests.
- Timeout requests that use absolute dates and times.
- Timeout requests with counts and intervals (for multiple message propagation).
- Timeout Notification node running in Automatic mode.

For more information on the full functionality of the Timeout

```
-- Sample program for use with
--  IBM WebSphere Business Integration Message Broker v 6.0
--
-- (C) COPYRIGHT International Business Machines Corp., 2005
-- All Rights Reserved * Licensed Materials - Property of IBM
--
-- This sample program is provided AS IS and may be used, executed,
-- copied and modified without royalty payment by customer
--
--  (a) for its own instruction and study,
--  (b) in order to develop applications designed to run with an IBM
--        WebSphere product, either for customer's own internal use or for
--        redistribution by customer, as part of such an application, in
--        customer's own products.

CREATE COMPUTE MODULE TwoSecWait
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        SET OutputRoot = InputRoot;
        DECLARE jump INTERVAL;
        SET jump = INTERVAL '2' SECOND;
        DECLARE start TIME;
        SET start = CURRENT_TIME + jump;
        SET OutputLocalEnvironment.TimeoutRequest.Action = 'SET';
        SET OutputLocalEnvironment.TimeoutRequest.Identifier = 'twosecond';
        SET OutputLocalEnvironment.TimeoutRequest.StartDate = 'TODAY';
        SET OutputLocalEnvironment.TimeoutRequest.StartTime = start;
        RETURN TRUE;
    END;
END MODULE;
```

Figure 3: Setting a timeout request in ESQL

nodes, please refer to the WebSphere Business Integration Message Broker InfoCenter.

THE AGGREGATION FLOWS

The fan-out flow receives an incoming request from the AGGR_IN queue and writes the same request message to the AGGR_REQ_1, AGGR_REQ_2, and AGGR_REQ_3 queues (simulating three request–reply applications) – Figure 4.

The fan-in flow receives incoming replies for consolidation from the AGGR_REP queue, and when the last reply of an aggregation operation is received it propagates the aggregated

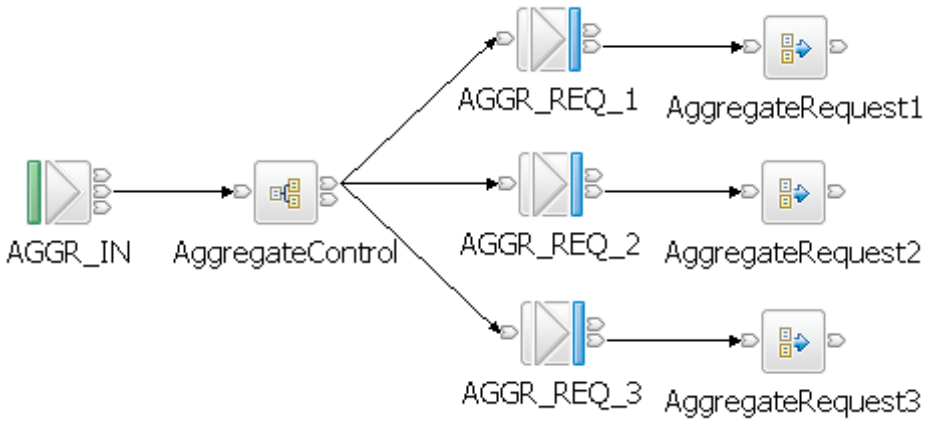

Figure 4: The Aggregation fan-out flow

reply message to the AGGR_OUT queue, via a Compute node to format the message so it can be written to the queue.

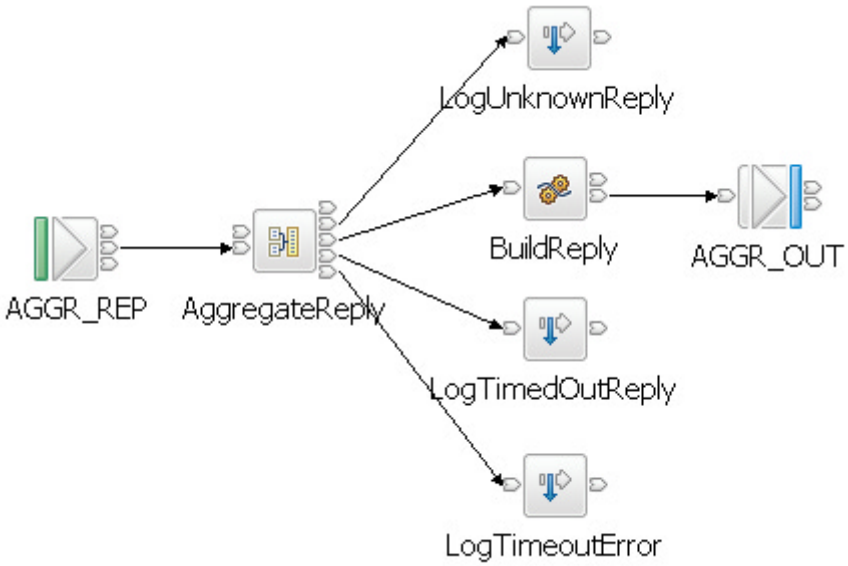

Figure 5: The Aggregation fan-in flow

For more information on using the Aggregation nodes, please refer to the WebSphere Business Integration Message Broker InfoCenter.

Ewan Withers
Software Developer
IBM (UK)

WebSphere Portal installation on z/Linux – part 3

This month we conclude the article describing the multi-tier installation of IBM WebSphere Portal Server with various bundled components on a z/Linux platform.

- *WpsDsName=wps50DS* – this value is the name of the data source to be used for the WebSphere Portal database. Note: if the WebSphere Portal data source name is changed because of database migration, which is reflected in *wp_root/config/wpconfig.properties*, update the resource mappings by executing the following steps:

  i Log in to WebSphere Application Server Administrative Console.

  ii Select *Application/Enterprise Applications*.

  iii Select the WebSphere Portal application.

  iv Select the *Map resource references to resources* option.

  v Change the JNDI name for reference binding *jdbc/wpsDS* to specify the new data source name. Click *OK*.

  vi Save the configuration changes.

  Default value: wps50DS.

- *WpsXDbName=wps5TCP* – the TCP/IP alias for the database to be used as dataset name. This value is used to specify the data source on machines that are not running Windows.

  Default value: wps5TCP. Note: this value cannot exceed eight characters and can contain only letters and numbers. Refer to DB2 documentation for more information.

- *WpsDbNode=dbserver* – this value is the node for the WebSphere Portal database and is needed only in non-Windows environments. If you created your databases manually, this is the *was_node* value that you identified when you catalogued the TCP/IP node in the Creating database section. If you plan to use the configuration task to automatically create the databases, use this value.

  Default value: WPSNODE.

Section of properties file *WebSphere Portal content publishing Database properties*:

- *WpcpDbNode=dbserver* – this value is the node for the WebSphere Portal content publishing database and is needed only in non-Windows environments. If you created your databases manually, this is the *was_node* value that you identified when you catalogued the TCP/IP node in the Creating database section. If you plan to use the configuration task to automatically create the databases, use this value. Note: required only for non-Windows platforms.

  Default value: wcmNode.

- *WpcpXDbName=wpcp5TCP* – this value is the TCP/IP alias for the WebSphere Portal content publishing database and is needed only in non-Windows environments. This value is used to specify the data source. Note: required only for non-Windows platforms.

Default value: wpcp5TCP. Note: this value cannot exceed eight characters and can contain only letters and numbers. Refer to DB2 documentation for more information.

- *FeedbackXDbName=fdbk5TCP* – this value is the TCP/IP alias for the feedback database. This value is used to specify the data source. Note: required only for non-Windows platforms.

  Default value: fdbk5TCP. Note: this value cannot exceed eight characters and can contain only letters and numbers. Refer to DB2 documentation for more information.

- *WpcpDbName=wpcp50* – this value represents the database name (or alias name if remote) where you want the WebSphere Portal content publishing objects to be created. Note: this value is also the database element in the WpcpDbUrl property.

  Default value: wps50. Note: this value cannot exceed eight characters and can contain only letters and numbers. Refer to DB2 documentation for more information.

- *WpcpDbUser=db2admin* – this value should be an administrative user in the database. If WebSphere Portal is creating the database, user must be an administrative user. If you choose to use one database to store all WebSphere Portal, Member Manager, and WebSphere Portal content publishing information, this user must be different from DbUser.

  Default value: wcmdbadm.

- *WpcpDbPassword=<password>* – this password must match the password for the database user ID that is indicated in WpcpDbUser.

  Default value: password.

- *WpcpDbUrl=dbc:db2:wpcp50* – the database URL that is used to access the WebSphere Portal content publishing database with JDBC, where <*hostname*> is the name of the remote server and <*port*> is the port where the appropriate database instance is listening. The value must conform to standard JDBC URL syntax. Note: the database element of this value should match the value of WpcpDbName.

  Default value: jdbc:db2j:wps50;create=true (Cloudscape).

- *WpcpDbEjbPassword=<password>* – the password for Enterprise Java Bean (EJB) user. Only needed for MS SQL Server and Oracle.

  Default value: ejb.

- *FeedbackDbName=fdbk50* – this value represents the database name (or alias name if remote) where you want the feedback objects to be created. Note: this value is also the database element in the FeedbackDbUrl property.

  Default value: wps50. Note: this value cannot exceed eight characters and can contain only letters and numbers. Refer to DB2 documentation for more information.

- *FeedbackDbUser=db2admin* – this value should be an administrative user in the database. If WebSphere Portal is creating the database, the user must be an administrative user.

  Default value: wcmdbadm.

- *FeedbackDbPassword=<password>* – this password must match the password for the database user ID that is indicated in FeedbackDbUser.

  Default value: password

- *FeedbackDbUrl=jdbc:db2:fdbk50* – the database URL

that is used to access the feedback database with JDBC, where <*hostname*> is the name of the remote server and <*port*> is the port where the appropriate database instance is listening. The value must conform to standard JDBC URL syntax. If you are installing WebSphere Portal content publishing under one database, this value will be the same as the value for the WpcpDbUrl property. Note: the database element of this value should match the value of FeedbackDbName.

Default value: jdbc:db2j:wps50;create=true (Cloudscape).

Section of properties file *Member Manager properties*:

- *WmmDsName=wmmDS* – this value is the name of the data source to be used for the Member Manager database.

  Default value: wmmDS.

- *WmmDbName=wps50* – this value represents the database name (or alias name if remote) where you want the Member Manager objects to be created. If Member Manager will share the WebSphere Portal database, this value will be the same as the value for WpsDbName. Note: this value is also the database element in the WmmDbUrl property.

  Default value: wps50. Note: this value cannot exceed eight characters and can contain only letters and numbers. Refer to DB2 documentation for more information. Note: if WebSphere Portal Version 5.0.2 and a previous version of WebSphere Portal coexist on the same machine, this value must be different from the WebSphere Member Services database name in the previous version of WebSphere Portal.

- *WmmDbUser=db2admin* – this value should be an administrative user in the database.

Default value: db2admin. Note: if you are migrating from a previous version of WebSphere Portal, this value must match the database user name for the WebSphere Member Services database from the previous WebSphere Portal version.

- *WmmDbPassword=<password>* – this password must match the password for the database user ID that is indicated in WmmDbUser.

  Default value: password.

- *WmmDbUrl=jdbc:db2:wps50* – the database URL that is used to access the Member Manager database with JDBC. The value must conform to standard JDBC URL syntax. If WebSphere Portal and Member Manager are sharing a database, you will use the same database value entered for WpsDbName. Note: the database element of this value should match the value of WmmDbName.

  Default value: jdbc:db2j:wps50;create=true (Cloudscape).

5 Save the file.

6 Unix only: use the following steps to export the db2instance environment in your profile:

- In your *.bashrc*, *.dshrc*, or *.profile* file, add:

```
if [ -f /home/db2admin/sqllib/db2profile ]; then . /home/db2admin/
sqllib/db2profile; fi
```

where *db2admin* represents your database instance.

- Reopen all the shells.
- Validate that your environment has set the DB2 profile environment variables, such as *DB2INSTANCE=db2admin*, where *db2admin* represents your database instance by running the **env** command.

7 Ensure that you are in the directory */opt/WebSphere/PortalServer/config.*

8 Enter the following commands to validate configuration properties:

```
./WPSconfig.sh validate-database-connection-wps -
DDbPassword=<password>

./WPSconfig.sh validate-database-connection-wmm -
DWmmDbPassword=<password>

./WPSconfig.sh validate-database-connection-wpcp -
DWpcpDbPassword=<password> -DFeedbackDbPassword=<password>

./WPSconfig.sh validate-database-driver
```

9 Enter the following command to transfer the database:

```
./WPSconfig.sh database-transfer-import
```

Note: if the configuration fails, verify the values in the *wpconfig.properties* file and then repeat this step.

10 After importing the database tables, perform a reorg check to improve performance. Connect to each database and run the following commands from the DB2 prompt:

```
reorgchk update statistics on table all
terminate
db2rbind <database_name> -l db2rbind.out -u db2admin -p
<password>
```

11 Start the WebSphere Portal application:

```
# cd /opt/WebSphere/AppServer/bin.
# ./startServer.sh WebSphere_Portal.
```

12 Verify that the WebSphere Portal application server is running by opening the following URL in a browser:

```
http://WPSNODE.company.com:9081/wps/portal
```

Enabling WebSphere security

Follow the steps below to edit the *wpconfig.properties* file and run the appropriate configuration tasks so that WebSphere Portal can work with the LDAP server:

1 Create a back-up copy before changing any values:

```
# cd /opt/WebSphere/PortalServer/config
# cp wpconfig.properties wpconfig.properties.b4ldap
```

2 Use a text editor to open the *wpconfig.properties* file and enter the values appropriate for your environment.

Note the following:

- Do not change any settings other than those specified in these steps.
- Use / instead of \ for all platforms.

Section of properties file *WebSphere Application Server*:

- *WasUserid= uid=wpsbind,cn=users,o=company, o=com* – the user ID for WebSphere Application Server security authentication. This should be the fully qualified distinguished name (DN). Note: if a value is specified for *WasUserid*, a value must also be specified for *WasPassword*. If *WasUserid* is left blank, *WasPassword* must also be left blank. Also note: for LDAP configuration this value should not contain spaces.

  Default LDAP value: *uid=wpsbind,cn=users, dc=yourco,dc=com*.

- *WasPassword=wpsbind* – the password for WebSphere Application Server security authentication. Note: if a value is specified for *WasPassword*, a value must also be specified for *WasUserid*. If *WasPassword* is left blank, *WasUserid* must also be left blank.

  Default value: <none>.

Section of properties file *WebSphere Portal configuration*:

- *PortalAdminId= uid=wpsadmin,cn=users,o=company, o=com* – the fully-qualified name of the WebSphere Portal administrator. This should be the fully-qualified distinguished name (DN). Note: for LDAP configuration this value should not contain spaces.

Default value: <none>.

- *PortalAdminIdShort=wpsadmin* – the short form of the user ID for the WebSphere Portal administrator, as defined in the PortalAdminId property.

  Default value: <none>.

- *PortalAdminPwd=wpsadmin* – the password for the WebSphere Portal administrator, as defined in the PortalAdminId property.

  Default value: <none>.

- *PortalAdminGroupId= cn=wpsadmins,cn=groups, o=company,o=com* – the group ID for the group to which the WebSphere Portal administrator belongs.

  Default value: <none>.

- *PortalAdminGroupIdShort=wpsadmins* – the short form of the group ID for the WebSphere Portal administrator, as defined in the PortalAdminGroupId property.

  Default value: <none>.

Section of properties file *WebSphere Portal Security LTPA configuration*:

- *LTPAPassword=<password>* – the password for the LTPA bind.
- *LTPATimeout=120* – sets the timeout for the LTPA bind.

  Default value: 120.

- *SSODomainName=company.com* – Single sign-on domain, for example SSODomainName=yourcompany.com.

Section of properties file *LDAP Properties Configuration*:

- *Lookaside=false* – the purpose of a Lookaside

database is to store attributes that cannot be stored in your LDAP server. You can either install with LDAP only or with LDAP using a Lookaside database. To enable a Lookaside database, set this property to true. If you intend to use a Lookaside database, set this value before configuring security, because it cannot be configured after security is enabled. Note: using a Lookaside database can slow down performance.

Default value: false.

- *LDAPHostName= ldapcls.company.com* – the host information for the LDAP server that WebSphere Portal will use; for example, yourserver.yourcompany.com.

  Default value: *wpsldap.ibm.com*.

- *LDAPPort=389* – the port number for the LDAP server that WebSphere Portal will use.

  Default value: 389 (636 for SSL).

- *LDAPAdminUId=cn=root* – the LDAP access ID; for example, LDAPAdminUId=cn=root. This is the ID that WebSphere Portal (Member Manager) will use to access the LDAP directory. This does not have to be the root admin ID for the directory, simply an ID that has sufficient privileges to the directory to allow the operations that WebSphere Portal will perform.

  If WebSphere Portal will only read from the directory, not make updates, an ID with read privileges to the directory is sufficient. If WebSphere Portal will update the directory (create users or make user profile updates to the directory) then an ID with write privileges is required.

  Default Value: cn=root.

- *LDAPAdminPwd=<password>* – the LDAP access

password.

Default value: <none>.

- *LDAPServerType=IBM_DIRECTORY_SERVER* – type of LDAP Server to be used.

  Default value: IBM_DIRECTORY_SERVER.

- *LDAPBindID=uid=wpsbind,cn=users,o=company, o=com* – user ID for LDAP bind authentication.

  Default value: uid=wpsbind,cn=users,dc=yourco,dc=com.

- *LDAPBindPassword= wpsbind* – password for LDAP bind authentication.

  Default value: <none>.

Section of properties file *Advanced LDAP Configuration*:

- *LDAPUserFilter=(&(uid=%v)(objectclass=inetOrgPerson))* – this key is used to configure the user filter.

  Default value: (&(uid=%v)(objectclass=inetOrgPerson)).

- *LDAPGroupFilter=(&(cn=%v)(objectclass=groupOfUniqueNames))* – this key is used to configure the group filter.

  Default value: (&(cn=%v)(objectclass=groupOfUniqueNames)).

- *LDAPSuffix=o=company,o=com* – LDAP suffix.

  Default value: dc=yourco,dc=com.

- *LdapUserPrefix=uid* – DN prefix attribute name for user entries.

  Default value: uid.

- *LDAPUserSuffix=cn=users* – DN suffix attribute name

for user entries.

Default value: cn=users.

- *LdapGroupPrefix=cn* – DN prefix attribute name for user entries.

  Default value: cn.

- *LDAPGroupSuffix=cn=groups* – DN suffix attribute name for group entries.

  Default value: cn=groups.

- *LDAPUserObjectClass=inetOrgPerson* – user object class corresponding to your directory.

  Default value: inetOrgPerson.

- *LDAPGroupObjectClass=groupOfUniqueNames* – group object class corresponding to your directory.

  Default value: groupOfUniqueNames.

- *LDAPGroupMember=uniqueMember* – specifies the attribute name of the membership attribute of your group objectclass.

  Default value: uniqueMember.

- *LDAPsslEnabled=false* – specifies whether secure socket communications is enabled to the LDAP server.

  Value (SSL): true.

  Default value: false.

3 Since we have installed WebSphere Application Server as part of the WebSphere Portal installation and plan to use WebSphere Application Server single sign-on, ensure that the following additional properties in the *wpconfig.properties* file have the values listed below.

Section of properties file *WebSphere Portal Security LTPA and SSO Configuration*:

- SSOEnabled=true – specifies that the single sign-on function is enabled.

  Default value: true.

- SSORequiresSSL=false – specifies that single sign-on is enabled only when requests are over HTTPS Secure Sockets Layer (SSL) connections. Choose *false* unless SSL is already enabled for WebSphere Portal. In most cases, SSL for WebSphere Portal will not yet be in place. After SSL for WebSphere Portal is set up, change this value using the WebSphere Application Server administrative console.

  Default value: false.

4 Save the file.

5 Start the application servers:

```
# cd /opt/WebSphere/AppServer/bin
# ./startServer.sh server1
# ./stopServer.sh WebSphere_Portal
```

6 Change to the directory */opt/WebSphere/PortalServer/config*.

7 Enter the following command to run the configuration task:

```
# ./WPSconfig.sh validate-ldap
```

Note: if the configuration task fails, validate the values in the *wpconfig.properties* file.

8 Enter the following command to run the configuration task:

```
/WPSconfig.sh enable-security-ldap
```

9 This command will take some time (approximately 30 minutes) to finish. On completion of this command, note the message:

```
BUILD SUCESSFUL
```

Note: if the completion of the command returns BUILD FAILED, check the output for the error messages before proceeding with any additional tasks. If the configuration task fails, verify the values in the *wpconfig.properties* file.

Before running the task again, be sure to stop the WebSphere Portal application server:

```
# cd /opt/WebSphere/AppServer/bin
# ./stopServer.sh WebSphere_Portal -user wpsbind -password <password>
```

Once you have enabled security with your LDAP directory, you will need to provide the user ID and password required for security authentication on WebSphere Application Server when you perform certain administrative tasks with WebSphere Application Server. For example, to stop the WebSphere Portal application server, you would issue the following command:

```
# ./stopServer.sh WebSphere_Portal -user wpsbind -password <password>
```

Verifying configuration

Access WebSphere Portal via http://WPSNODE.company.com:9081/wps/portal and verify that you can log in.

Note: configuring WebSphere Portal to work with an LDAP directory automatically enables WebSphere Application Server Global Security. Once security is enabled, you must type the fully-qualified host name when accessing WebSphere Portal and the WebSphere Application Server Administrative Console.

Ensure that the Web server is started and then access WebSphere Portal via the port 80 of the HTTP Server to check the remote Web server configuration. Verify that you get the WebSphere Portal login page and you are able to log in using the url http://WPSNODE.company.com/wps/portal.

WebSphere Business Integration Message Broker – simplified functional validation

DEFINING WEBSPHERE BIMB

One of the hottest product families in the marketplace today is IBM's WebSphere Business Integration (WBI) product suite. Middleware is becoming pervasive throughout businesses globally, and the WBI products represent a significant proportion of installed middleware software. The products in the IBM WBI product family include:

- WebSphere Application Server (WAS)
- WebSphere Edge Server (WES)
- WebSphere Portal Server (WPS)
- WebSphere Process Choreographer (WPC)
- WebSphere MQSeries (WMQ)
- WebSphere Business Integration Message Broker (WBIMB)
- WebSphere DB2 UDB (WDB2)
- WebSphere Data Interchange (WDI).

Companies install a Web server (like IBM's HTTP, IIS, Apache, etc) and then connect it to WAS in order to provide a user community with browser-based feature-rich applications in addition to corporate Web pages.

As end user requirements become more complex, application access to external resources such as WDB2 and correlated applications becomes a must. In order to move data requests around the corporate infrastructure, WMQ is installed to provide consistent access methods across platforms with secure transmission and assured delivery.

Once the middleware infrastructure is in place, the complexity of Enterprise Application Integration (EAI) grows with time. More users in remote locations need access to more applications. Multiple applications require access to disparate data sources. As the number of possible connections grows, the number of paths that messages can travel increases as well. At this point, companies begin to consider WBIMB. WBIMB is a message transformation engine. Using the features of WBIMB, you can manage the flow of messages throughout the corporate middleware infrastructure.

Systems administrators are responsible for installing, maintaining, and managing WBIMB. WBIMB functionality is driven by message flows, which are coded modules that act on messages coming within their area of control. WBIMB is actually a collaboration of applications including:

- WBI Configuration Manager – stores and manages an object store containing all the components that are known to and used by WBIMB.
- WBI Message Broker – the actual message processing engine that executes the coded message flows and manipulates the messages according to program requirements.
- WDB2 UDB – used as the data store for all of the Configuration Manager objects and relationships within the WBIMB instance.

A message flow project (the package within which message flows are contained) allows designers programmatically to examine WMQ messages, and, depending on requirements, to manipulate those messages either by changing the message content or by altering the movement of the message through the middleware infrastructure, or both.

Message flow projects (and the message flows within) are normally created by the application development team as part of their design implementation. As a WBIMB administrator, your job is to take those message flow projects and install them within the WBIMB instance.

Installing message flow projects, like any other WebSphere application implementation process, is a well defined, if somewhat convoluted, process. If you have experienced application deployments under WebSphere Application Server, or other WBI-based products, you will doubtlessly have experienced times when the files delivered by application development will not deploy properly.

When a deployment goes wrong, for whatever reason, I can almost guarantee that the systems administrator will blame the message flow project, and the application developers will blame the WBIMB installation. Unless you have an actively working instance, which can lend credence to the administrator's claim that the message flow project must be bad, it can be very difficult to get the agreement of the application developers in fixing the problem.

So, how do we prove that the existing, or newly-installed, WBIMB instance is correct and active? The best way is by using the WBIMB tools, as provided, to create a simple message flow project and its associated message flow components. If we can deploy a test flow successfully, we have an active instance, which proves that the infrastructure is probably not to blame.

THE MESSAGE BROKERS TOOLKIT FOR WEBSPHERE STUDIO

Within the Windows installation media for WBIMB, you will find the Broker Administration – Message Brokers Toolkit for WebSphere Studio – Message Broker graphical user interface utility. This can be installed as part of the WBIMB instance installation on a Windows server or workstation, or it can be installed separately on any Win32 workstation or server. The Toolkit is the IBM provided GUI for use with WBI Message Broker and WBI Configuration Manager. Additionally, it contains utility features that allow WBI message flow development and debugging.

DEFINING THE WEBSPHERE MQ INFRASTRUCTURE COMPONENTS

In order to test the WBIMB message flow, it will be necessary to create a set of WebSphere MQSeries (WMQ) objects that can be used to pass messages into and out of the WBIMB queue manager instance. In the simplest implementation you need to have two local queues defined in the queue manager – for the sake of this article these queues are named WBI_TRAIN_INCOMING and WBI_TRAIN_OUTGOING. The queues are self-describing. WBI_TRAIN_INCOMING is the queue where we will put the test messages and WBI_TRAIN_OUTGOING is where the messages will be moved to when they are processed by the WBIMB message flow.

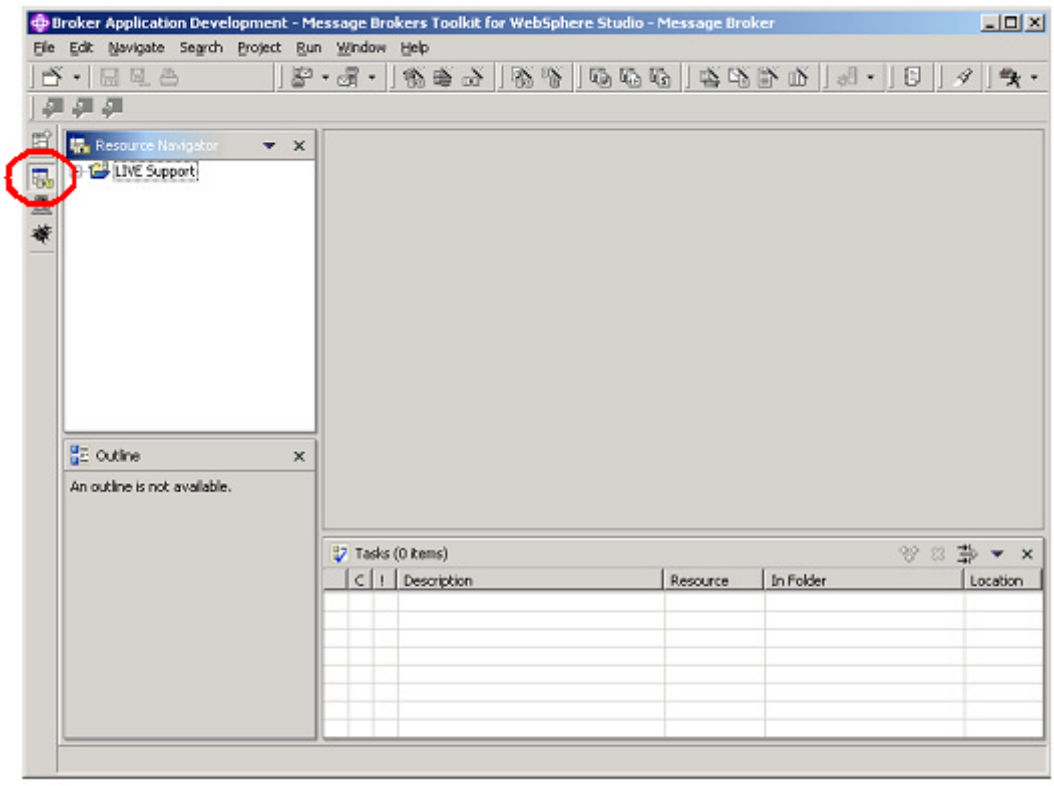

Figure 1: Starting the Broker ADP

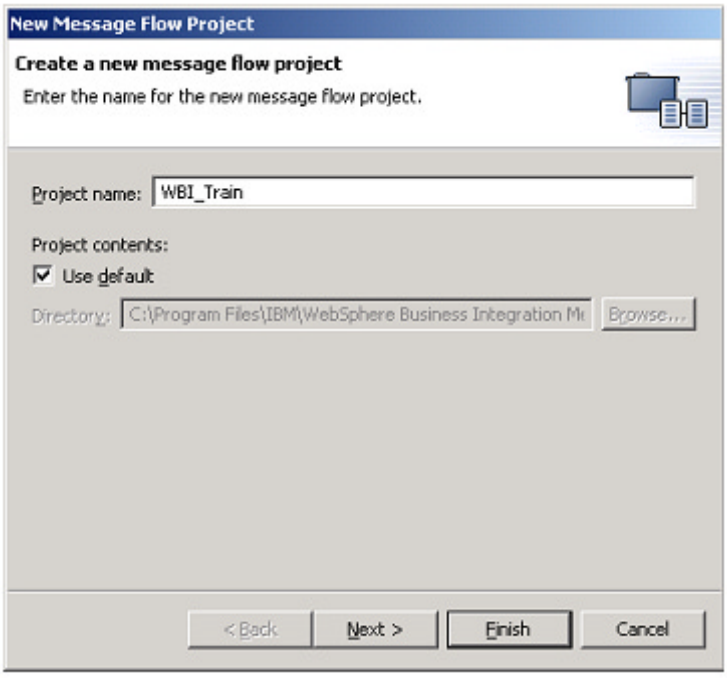

Figure 2: Broker ADP dialogue

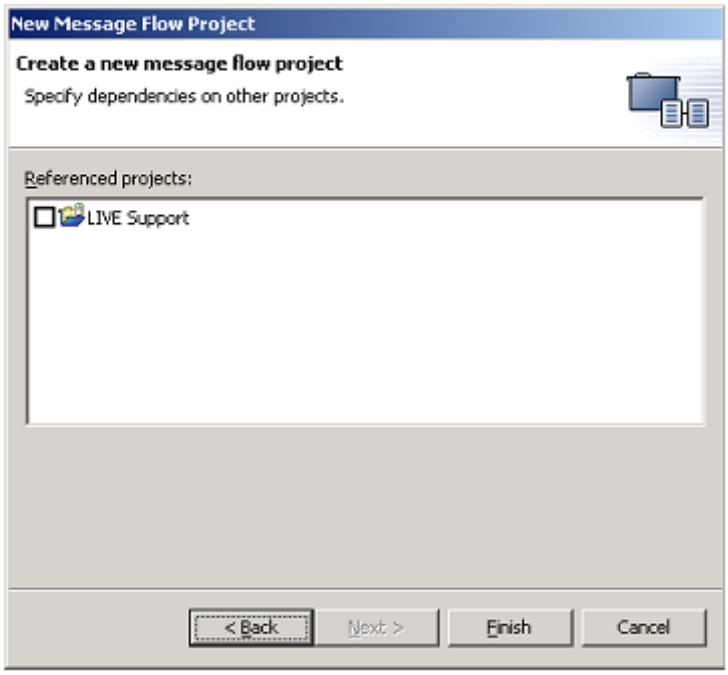

Figure 3: Listing dependencies

CREATING A TEST MESSAGE FLOW

The WBI Message Broker message flow is a self-descriptive object. Its purpose is to take incoming messages and perform a defined set of functions on either the content or movement of those messages.

To create a message flow, we will start from the Broker Application Development Perspective (ADP) within the Message Broker Toolkit. You can start the Broker ADP by clicking on the icon to the left of the Resource Navigator pane – see Figure 1.

To begin the creation of the message flow, you first need to create a Message Flow Project by right clicking on the upper left pane in the Broker ADP dialogue, and select **New/Message Flow Project** from the drop down menus – see Figure 2.

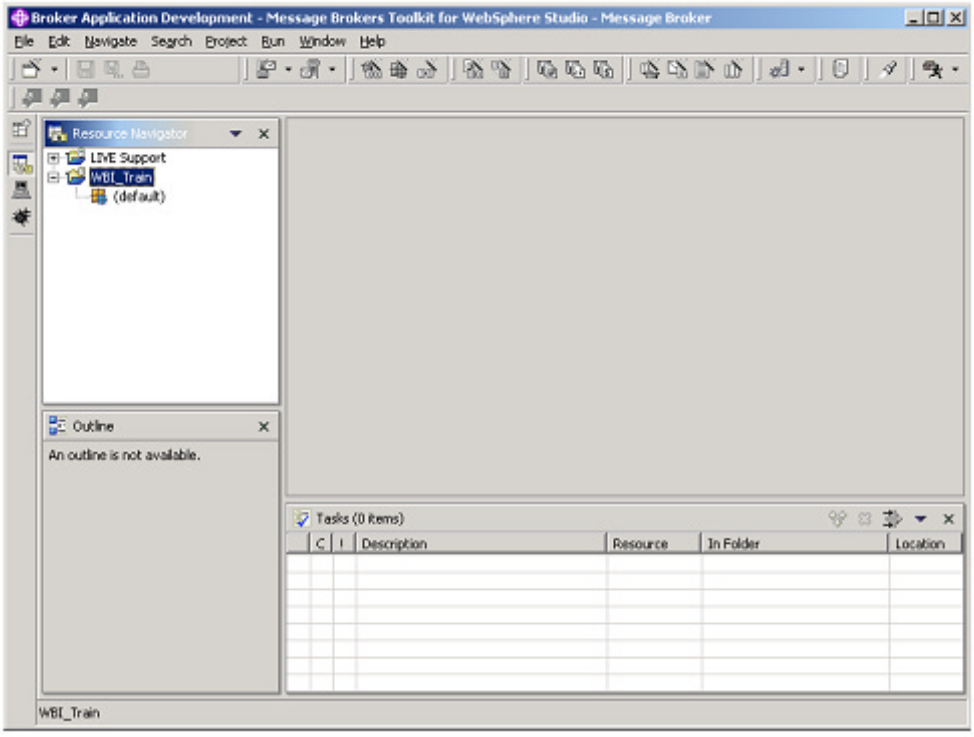

Figure 4: Project creation

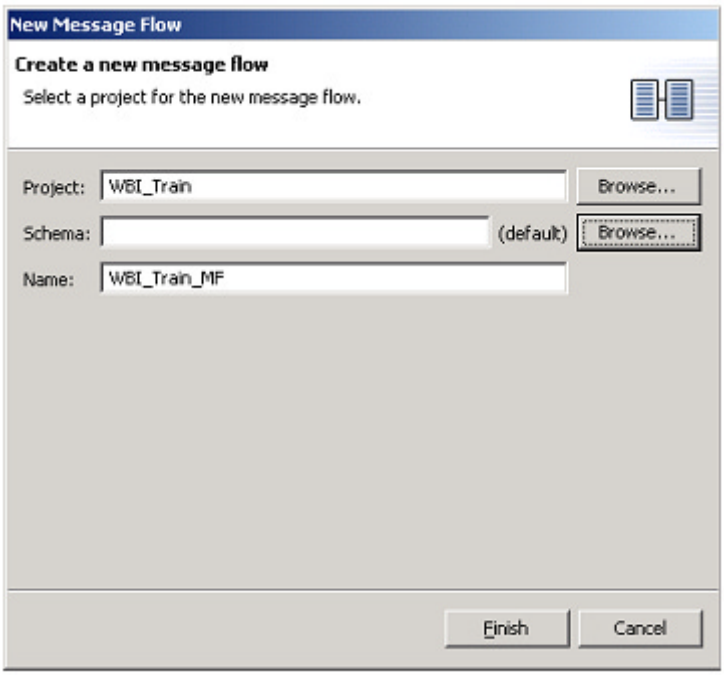

Figure 5: Message flow creation

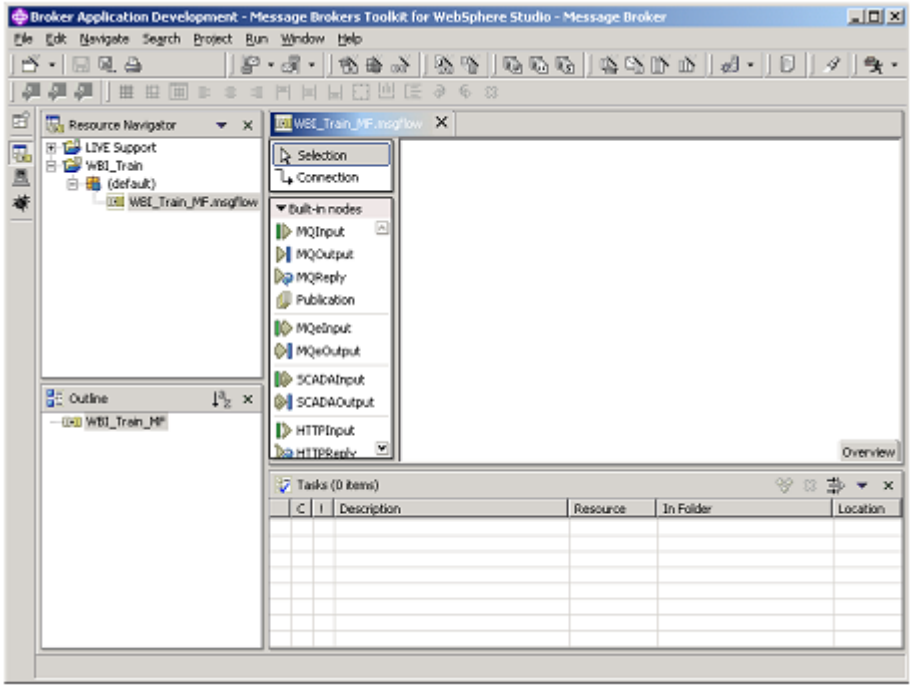

Figure 6: Message flow editor

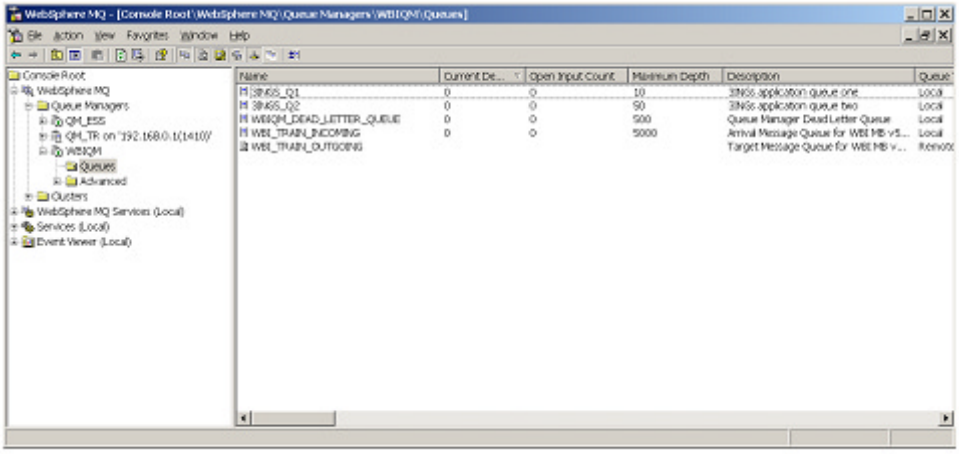

Figure 7: WMQ queues

Give the new project a name (as shown above) that is appropriate for your task. When you press the **Next** button, you will be asked whether this message flow is dependent on

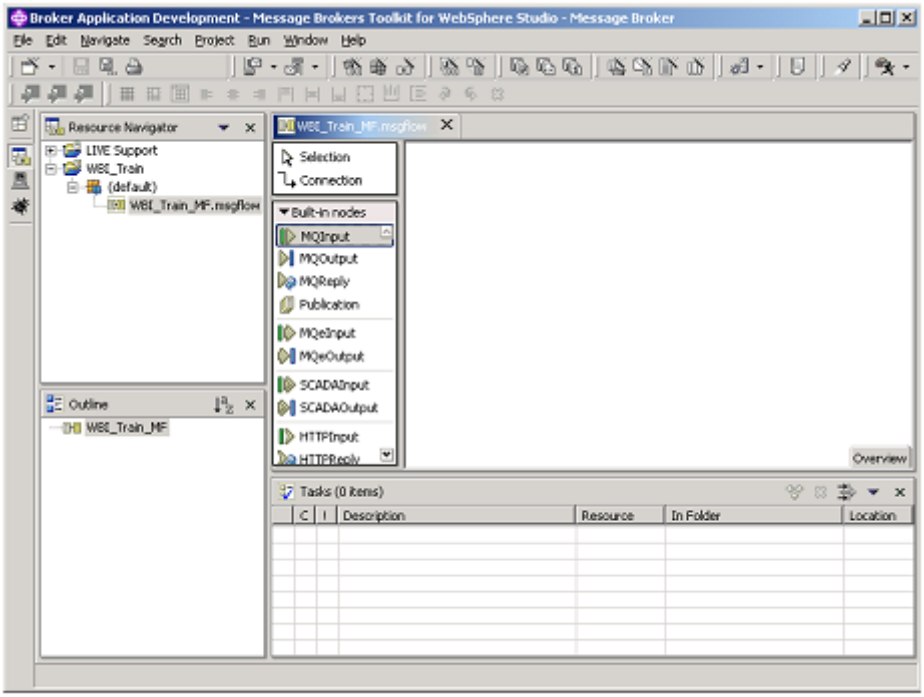

Figure 8: Adding an MQInput node

any other defined flows – see Figure 3.

If you are creating a stand-alone message flow, you can press **Finish** from the original panel, or press **Next** and then **Finish** to create the Project – see Figure 4.

Once you have the message flow project in place, you can create the message flow to go with it. Left click on the Message Flow Project to highlight it, and then right click in the top left pane on the dialogue window and select **New/Message Flow** – see Figure 5.

Name the message flow, and then press **Finish** to create the flow object and open the message flow editor at the top right of the Broker ADP dialogue – see Figure 6.

For the purposes of this exercise, we are going to use the WMQ queues that have been defined on the Broker queue manager as the source of the messages and the subsequent target. The message flow that will be created will simply take incoming WMQ messages and redirect them to a follow-on

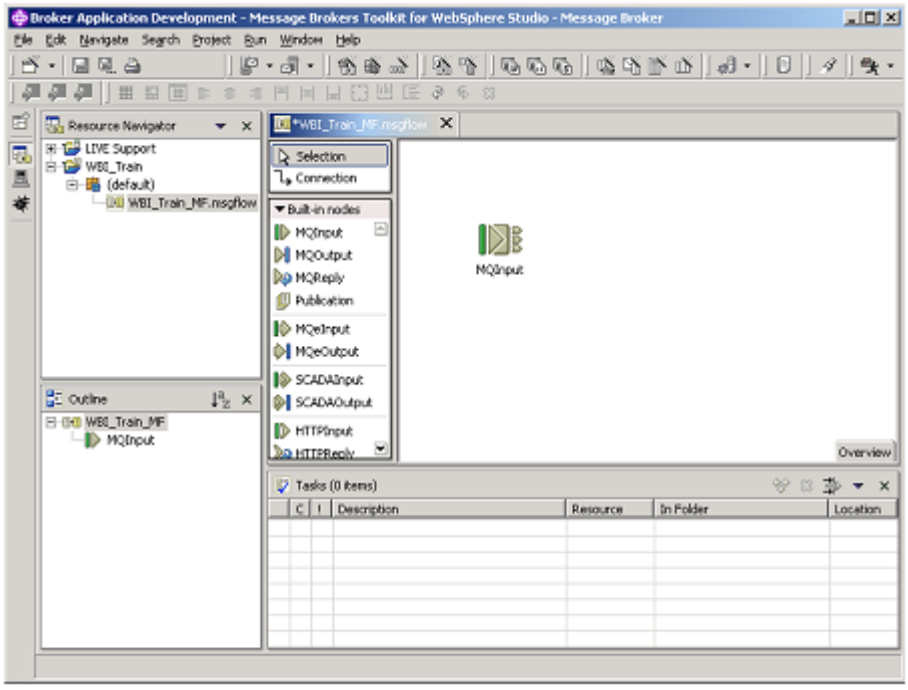

Figure 9: Editing

queue. The WMQ queues that have been defined (shown in Figure 7) are:

- Local queue – WBI_TRAIN_INCOMING.
- Remote queue – WBI_TRAIN_OUTGOING.

Note: in our WBIMB instance, the WMQ local queue WBI_TRAIN_INCOMING is referred to by a remote queue manager and the WMQ remote queue WBI_TRAIN_OUTGOING refers to a local queue on the same remote queue manager.

To build the message flow functionality, first it is necessary to add an MQInput node, which will identify the WMQ queue from where we will receive source messages. To add the node, click on the MQInput icon and highlight it in the menu to the left of the Message Flow editor screen – see Figure 8.

Next move the cursor into the editor screen and left click to drop the icon – see Figure 9.

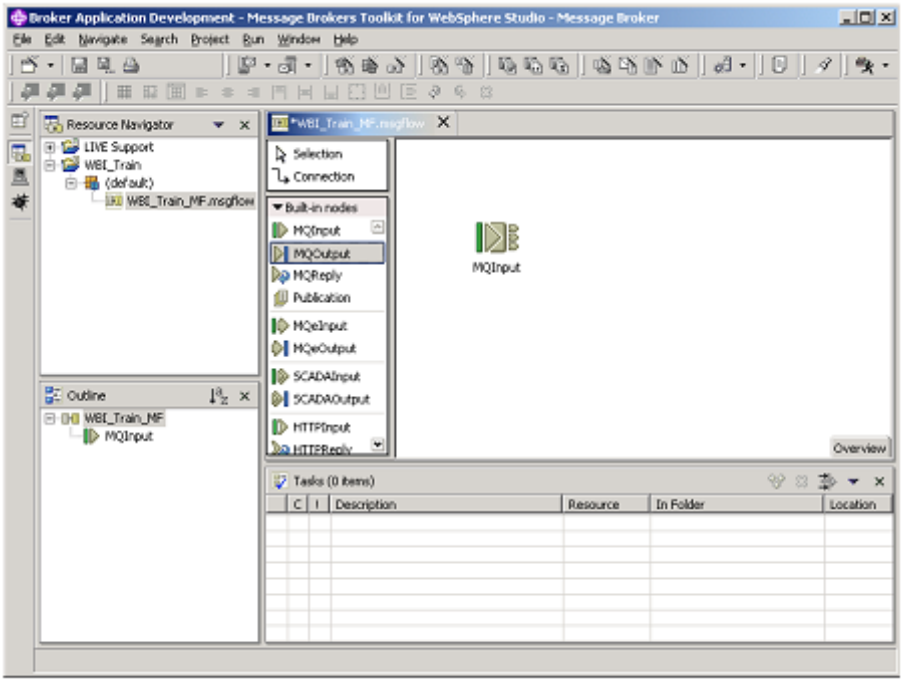

Figure 10: Adding an MQOutput node

You will notice that on the MQInput icon there are three tabs to the right, which represent message flow through the function. The three tabs, from top to bottom, are Failure, Out, and Catch.

You can see the function labels by hovering the cursor over a tab until the detail prompt appears in the toolkit window.

The next step is to add an MQOutput node to the message flow. To do this, as above, first highlight the MQOutput menu option to the left of the Message Flow editor screen – see Figure 10.

Next, left click in the message flow editor screen to drop the icon in place – see Figure 11.

On the MQOutput icon, there is one tab to the left, which represents messages moving into the function and which is labelled 'In', and two tabs to the right, which represent possible message destinations for the function and which are labelled, respectively, 'Failure' and 'Out'.

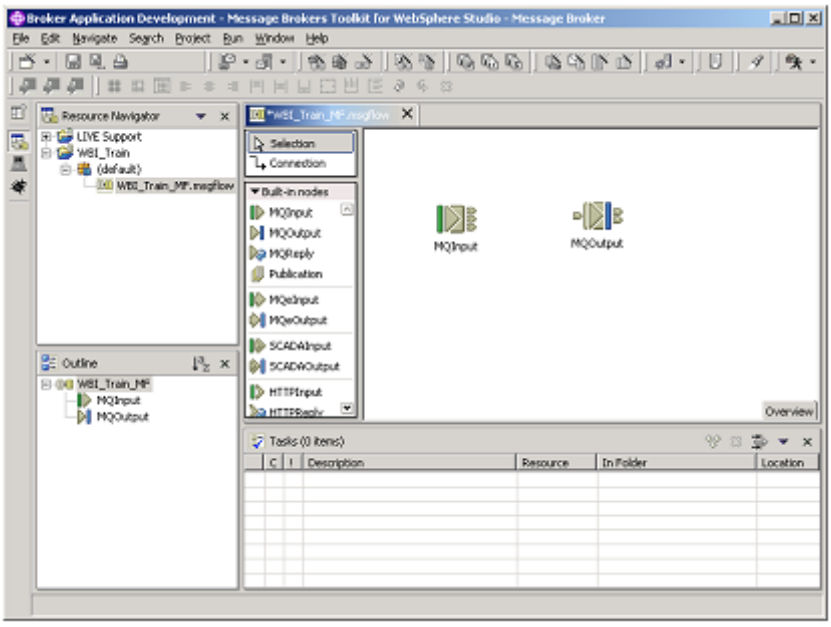

Figure 11: Editing

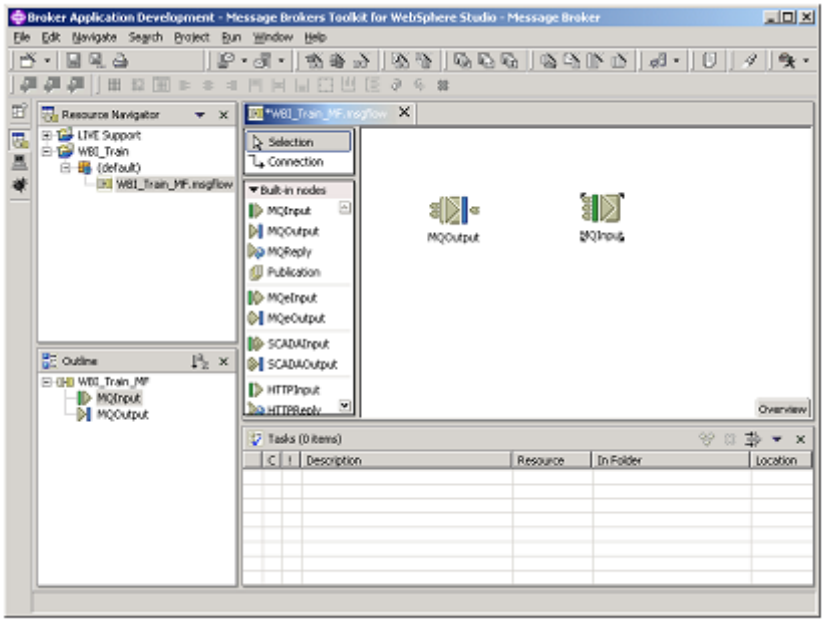

Figure 12: Reorganizing the icons

As before, you can see the function labels by hovering the cursor over a tab until the detail prompt appears in the toolkit window.

In the example above, the input icon is on the left and the output is on the right. For a Western European reader this makes logical sense. However, if you are dealing with a different perspective, such as an Israeli audience that are used to reading right to left, you can reorient the icons and the placement as you wish – see Figure 12.

Editor's note: this article will be continued next month.

Aaron Cain
Independent Consultant
3-INGs Limited (UK)

IBM has announced Version 6.0 of WebSphere Extended Deployment (XD), which is new software that delivers the combined capabilities of autonomic and grid computing to maximize the effectiveness of a business's application infrastructure.

The software is an add-on for WebSphere Application Server Network Deployment Version 5.1, and is designed to boost a users operational efficiency, particularly where there are spikes in traffic.

For further information contact:
URL: www.ibm.com/common/ssi/fcgi-bin/ssialias?infotype=an&subtype=ca&appname=GPA&htmlfid=897/ENUS205-174.

* * *

AdventNet has announced that its ManageEngine Applications Manager, Web application management software that provides integrated application, server, and systems monitoring, can now monitor WebSphere 6 and AIX.

This release also adds another feature called Script Monitoring through which existing *ad hoc* Windows/Linux scripts, used in-house, can be managed from the same Web console.

Applications Manager Enhancements include WebSphere 6 monitoring in network deployment and base configurations.

Applications Manager is available in three editions – a free edition, which can manage five applications; a professional edition, which helps small and medium-sized businesses manage their IT applications; and an enterprise edition with unlimited users pack, which helps large enterprises manage their diverse and complex applications, servers, and systems.

For further information contact:
URL: www.appmanager.com/download.html.

* * *

Cape Clear Software has announced Version 6.1 of its Enterprise Service Bus (ESB), which provides three levels of messaging support, including full built-in support for reliable Internet-based messaging via WS-ReliableMessaging, as well as HTTP, SMTP, and FTP. It extends its native and tested support for JMS (Java Message Service) products to include WebSphere MQ, JBoss JMS, Oracle JMS, SonicMQ, TIBCO, and WebLogic JMS.

Cape Clear 6.1 also ships with an optional, fully-integrated version of the JBoss JMS. In addition to expanded messaging support, the new release adds a range of enhancements to its Business Process Execution Language (BPEL) functionality.

For further information contact:
URL: www.capeclear.com/products.

* * *