61

MQ update

July 2004

In this issue

© Xephon Inc 2004

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

Nicole Thomas
E-mail: nicole@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; $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 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.

© Xephon Inc 2004. All rights reserved. None of the text in this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior permission of the copyright owner. Subscribers are free to copy any code reproduced in this publication for use in their own installations, but may not sell such code or incorporate it in any commercial product. No part of this publication may be used for any form of advertising, sales promotion, or publicity without the written permission of the publisher.

Printed in England.

Using amqmdain to manage WebSphere MQ for Windows

INTRODUCTION

In issue 51 of *MQ Update* (September 2003) an article of mine was published that showed how to automatically start WebSphere MQ (WMQ) programs on Unix systems (see *Auto-starting WMQ on Unix*). Since then I've seen some questions about how to do similar things on Windows machines. This article is about the configuration program called **amqmdain**, which can be used to configure both automatic starting and a number of other attributes for Windows systems.

HOW WMQ WORKS ON WINDOWS

Programs on Windows that can execute either automatically when the machine is booted or when there is no user logged on to the machine are run as **services**. These can be seen and controlled from the *Administrative Tools* folder from the Start menu.

A program that is running as a service cannot normally directly interact with a user unless it is running under the Local System account. Instead, calls can be made to the service via COM functions from a program that the user might be running. The service may have a number of attributes that can be configured. It can be started either automatically on system restart or manually. It might have security controls. It may have dependencies defined so that it will start only after some other service has been started.

WMQ includes all the components needed for it to be run as a service. When it is installed, an entry is made in the Services list, for 'IBM MQSeries'. (The old product name is maintained here, for compatibility with any older tools that customers might have written.) By default, a userid called MUSR_MQADMIN is created

© 2004. Reproduction prohibited. Please inform Xephon of any infringement.

during installation, and the service is configured to run as that user. This userid is in the mqm group and can therefore do all WMQ administrative tasks. The actual program being run is called **amqsvc.exe**. This program implements a COM interface and is able to start and stop the WMQ programs. It can also start and stop arbitrarily-configured user programs, based on the status of associated queue managers. The **amqsvc** program uses little resource, and can always be left running. However, if you do want to stop it, you can use:

```
NET STOP MQSeriesServices
```

from a command line.

THE WEBSPHERE MQ SERVICES GUI

The easy-to-use and interactive way of configuring WMQ queue managers, listeners, and other programs is to use the WMQ Services MMC snap-in. Do not confuse this with the WMQ Explorer snap-in, which is used as the graphical equivalent of the **runmqsc** program. (A good way of configuring your system is to customize the MMC view to show both snap-in components in a single panel.) As well as configuring start-up values, the snap-in also sets values in the Windows registry that correspond to entries in the *mqs.ini* or *qm.ini* files on other operating systems. While simple text editors such as vi can be used on a Unix machine, and little permanent damage can be caused by making a mistake editing an ini file, there are lots of things that can be broken if you err when updating the Windows registry by hand. So WMQ provides programs that will do the updates in a controlled way.

THE AMQMDAIN PROGRAM

While the GUI is useful as an interactive tool, there are times when you need to control things from a script or program. And this is where **amqmdain** comes in.

At its simplest, you should use **amqmdain** to start and stop queue managers, instead of using **strmqm** and **endmqm**. I use

© 2004. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

this facility for queue managers that I do not use frequently on my machine, for manual control instead of having them start automatically. For example, if I am using the WBI Message Broker product, I have a batch program that starts the queue manager and database components before starting the broker. Then, when I'm finished, I can end all the programs and free their resources. It's interesting to see how the different products have implemented their control commands; as I said earlier, WMQ has a single service program that then drives all configured queue managers under its control. DB2, by contrast, has all of its individual pieces executing as separate Windows services. Starting the Message Broker in a batch command looks like:

```
net start DB2-0
net start DB2CTLSV-0
net start DB2DAS00

amqmdain start WBRK_QM

mqsistart configmgr
mqsistart usernameserver
mqsistart WBRK_BROKER

pause
```

DEFINING INI FILE ENTRIES

All the stanzas that can be found in *qm.ini* or *mqs.ini* on other platforms can be modified by **amqmdain**. There are also a few Windows-specific entries that can be changed, such as the behaviour when a machine is suspended or hibernated.

The root of all WMQ registry information can be found under the key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion
```

I will refer to this point as MQROOT. Information about each queue manager can then be found in:

```
{MQROOT}\configuration\QueueManager\QMGRNAME
```

Note that when there might, in the text-based ini files, be several stanzas of the same type, the Windows registry has to slightly

© 2004. Reproduction prohibited. Please inform Xephon of any infringement.

restructure the data and use a unique element (which is usually the *name* attribute from the ini file) to contain the attributes for that stanza.

There are a number of examples given in the product documentation of how to use the command to modify these stanzas. I won't repeat that information here.

DEFINING SERVICES

An aspect of **amqmdain** that I feel is less fully explained in the books (especially as some function was added after the books were written) is how the custom services are configured and stored.

There are a few services, started with the queue manager, that are well known and predefined. These are the listener, command server, and channel initiator. A trigger monitor can also be defined, although one is not configured by default. Named channels can also be started, using the **runmqchi** command, although that is less likely to be used when the channel initiator automatically starts your channels. Multiple listeners can be configured, and these will appear in the registry as folders labelled Listener, Listener (2), and so on. The predefined classes of entries can be found at:

```
{MQROOT}\configuration\Services\QMGRNAME
```

Custom services are any other programs that should be started with the queue manager. One WMQ-provided program that might be useful here is the dead-letter queue handler, **runmqdlq**. Because **runmqdlq** takes its rules processing configuration from stdin instead of from a command line, it cannot be executed directly from the services panel. Instead we need to create a one-line batch file that can then redirect the configuration file into the real program.

In the GUI, you can see two sets of options for custom services. The two sets allow selection of PROCESS or COMMAND, and AFTER or BEFORE.

© 2004. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

The PROCESS/COMMAND switch determines whether the controlling program should wait for the custom service to complete or not before going on to the next service that needs starting. Programs that are long running (such as the **runmqdlq** example) should be started as a PROCESS; the status of these operations can be monitored by the controlling service. Programs that execute for only a short time (one example might be to delete log files from previous runs of an application) should be run as a COMMAND.

The AFTER/BEFORE option is used to select where the service should be started in relation to the queue manager. Note that there are no dependencies defined between services that are associated with a common queue manager. There is no way to force our dead-letter handler example to start before any other application, except by altering the order in which the services are listed in the registry.

The same configuration options are available from **amqmdain**, but they are provided to the program in a rather different way.

While ini file parameters are passed to **amqmdain** on the command line, custom services are first entered into a text file. That file is then given as a parameter to **amqmdain**, with the **cstmig** option.

The format of the file is to have multiple lines, where each line has the fields *Cmd Name*, *Start Cmd*, *End Cmd*, *Flags*, and *Queue Manager*.

The command name is the key under which the rest of the parameters are saved in the registry. The start and end commands are fairly obvious. If there is no command to end the service, then leave that field blank, still separating it from the other fields by commas. If the start and end commands require the queue manager to be passed as parameters, then it needs to be entered here. There is no automatic appending or insertion of the associated queue manager name into the commands. The default location for commands, when no explicit path is given, is set to be the bin subdirectory under the WMQ installation directory.

© 2004. Reproduction prohibited. Please inform Xephon of any infringement.

The *Flags* field is a string such as:

```
SUFFIX|ROOT|STARTUP|COMMAND
```

There are no quotes around this string and the various keywords are joined together with the OR symbol (vertical bar). Spaces could also be used between the keywords, but the OR makes it clear how they are linked together.

In the V5.3 documentation, the final field on the line is listed as Reserved, but it can actually be used to associate the custom service with a named queue manager. This queue manager name should, of course, match with any queue manager name that has been included in the start and end commands.

The complete set of keywords that can be put in the flags field, which also shows mutually exclusive settings, is:

- ROOT (0x02) or QMGR (0x01)

  If QMGR is set, then the service is associated with the named queue manager, the final field on the input line. If ROOT is set (or neither keyword is used), then the custom service will be started before or after *all* of the queue managers on the machine have been started. ROOT is the default setting of this pair of options.

- COMMAND (0x10) or PROCESS (0x20)

  These have the same meaning as in the GUI. COMMAND is the default setting of this pair of options.

- PREFIX (0x04) or SUFFIX (0x08)

  These are equivalent to the BEFORE and AFTER settings in the GUI. PREFIX is the default setting for this pair of options.

- STARTUP (0x40)

  Use the start command, as listed in the file. STARTUP is always assumed to be set, but it is helpful to put it in the input file for clarity.

- SHUTDOWN (0x80)

© 2004. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

Use the stop command, as listed in the file. If this flag is not set, even if there is a stop command, it will not be executed.

The hex values that I've listed cannot be used for input purposes, but are useful as we'll see later for decoding the storage of the flags in the registry.

In the GUI there is also the ability to have the service configured as *Automatic* or *Manual*. There does not appear to be a similar capability available from **amqmdain**, which always sets the service to be started automatically. The GUI also sets some additional retry parameters that are visible in the registry but not exposed through the user interface or through **amqmdain**.

It's important to know that when you use the **cstmig** option, all existing custom service definitions are deleted. The new input file replaces the services totally. This is because it would be rather difficult to know exactly how to merge the services with any previous ones. If all updates are made through the command line, it is possible to manage a system by controlling the input file, which will always be the master definition.

DISPLAYING SERVICES AND INI FILE ENTRIES

The **amqmdain** program includes an option to display the settings for ini file stanzas. However, there are two restrictions to this. The first is that there is no way in a single command to display all the stanzas: **amqmdain** must be issued multiple times, once per stanza. The second limitation is that the output format is not directly suitable for reuse as an input command. While readable, it needs to be reformatted if you want to repeat the commands that generated these settings.

```
C:\> amqmdain reg * -c display -s LogDefaults -v *
5724-B41 (C) Copyright IBM Corp. 1994, 2002.  ALL RIGHTS RESERVED.
Displaying registry value for Queue Manager '*'
        Attribute = LogDefaultPath, Value = C:\mqm\log
        Attribute = LogPrimaryFiles, Value = 3
        Attribute = LogSecondaryFiles, Value = 2
        Attribute = LogFilePages, Value = 256
        Attribute = LogType, Value = CIRCULAR
        Attribute = LogBufferPages, Value = 0
```

© 2004. Reproduction prohibited. Please inform Xephon of any infringement.

There is also no option built in to **amqmdain** that will display custom service entries.

However, the information can be extracted from the registry directly by using the Windows **regedit** command. This is usually run as an interactive GUI command, but the ***/e <filename>*** option extracts the given key and subfolders to a file, which can then be browsed. All custom services, whether dependent on a named queue manager or not, can be found under the key at:

```
{MQROOT}/services/.custom
```

Just as with the ini file entries, the format of displayed services cannot be directly used as input to **amqmdain**. It is not too difficult to reformat them into the correct style, however. (As a mostly-Unix programmer I would do it in a simple **awk** or **perl** script.) The only difficult parameter to decode is the *CustomType* field – a hex word. This corresponds to the *Flags* field in the input control file described earlier. Each keyword has a value, and these are added together to form the total that is set in the *CustomType* field.

An example session showing a configuration file and the generated registry entry is below:

```
C:\> type dlq.bat
runmqdlq < dlqrules.txt


C:\> type cst.reg
DLQ Handler,dlq.bat,,SUFFIX|QMGR|STARTUP|PROCESS, QMGRNAME

C:\> amqmdain cstmig cst.reg
5724-B41 (C) Copyright IBM Corp. 1994, 2002.  ALL RIGHTS RESERVED.
Added service 'DLQ Handler'
The custom service configured successfully
C:\>regedit /e cst.out HKEY_LOCAL_MACHINE\
SOFTWARE\IBM\MQSeries\CurrentVersion\configuration\
services\.custom

C:\> type cst.out

Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion\configuration\
services\.custom]
```


© 2004. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion\configuration\
services\.custom\DLQ Handler]
"CustomType"=dword:00000069
"StrCommand"="dlq.bat"
"EndCommand"=""
"Startup"=dword:00000001
"Depends"="QMGRNAME"

C:\>
```

SUMMARY

This article has shown how a Windows machine can be configured and managed through the command line – an essential element to automatic and unattended operation. While some additional utilities might be useful to extract and reformat an existing configuration for back-up purposes, the **amqmdain** and **regedit** commands can be used to provide all the data that is required.

Mark E Taylor
Technical Strategist
WebSphere MQ Development
IBM Hursley (UK)

© IBM 2004

Viewing a queue's properties from the Java console – revisited

Following the publication of the code in the article of the same name in the April 2004 issue of *MQ Update*, the author has more MQ Java code, which will get information from a queue and write it to a file. It reads the message from a separate file and puts it in the queue.

MQGETPUTFILE

```
/**************************************************************************/
/* Program name: mqGetPutFile                                             */
```

© 2004. Reproduction prohibited. Please inform Xephon of any infringement.

```
/* Author : Balaji SR                                                    */
/* mail id : balaji_srajan@yahoo.com                                     */
/************************************************************************/
/* Function:                                                             */
/*    Input : This is a Java console application which take a file      */
/*                          as an argument                               */
/*    Output: This displays the queue attributes on the console         */
/************************************************************************/
import com.ibm.mq.*;
import java.util.*;
import java.io.*;
public class mqAttrInquiry {
      private MQQueueManager mqQueueManager;         // for QMGR object
      private MQQueue queue;                         // for Queue object
      private int openOptionInquire;                 // Open options
      private String hostName;              // for host name -> QMGR
      private String channel;               // server connection channel
      private String port;   // port number on which the QMGR is running
      private String qmgrName;
      private String qName;
      public static void main(String arg[])
      {
            try{
                  if (arg.length == 1)
                  {
          System.out.print(" Propertie file name ..." + arg[Ø] + "\n" );
                  }
                  else
                  {
          System.out.print("Please enter the properties file name \n" );
                        System.exit(1);
                  }
                  mqAttrInquiry mqattrinquiry = new mqAttrInquiry();
                  mqattrinquiry.readPropertyFile(arg[Ø]);
                  mqattrinquiry.init();
                  }
            catch( Exception e)
            {
      System.out.print("Please enter the properties file name only\n" );
                  e.printStackTrace();
                  System.exit(1);
            }
      }
      public void init( )
      {
            try{
                  System.out.println("In init");
                  this.mqInit();
            }
            catch( Exception e)
```


© 2004. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
            {
                  e.printStackTrace();
            }
      }
      private void mqInit( )
      {      // Initiation of the MQ parameter
            try
            {
               System.out.println("hostName            : " + hostName);
               System.out.println("qmgrName            : " + qmgrName);
               System.out.println("port                : " + port);
               System.out.println("channel             : " + channel);
               System.out.println("QName               : " + qName);
                  getConnected();
            }
            catch (Exception exp)
            {
                  System.out.println(" Error in mqInit ...\n");
                  exp.printStackTrace();
                  System.exit(1);
            }
      }
      public void getConnected() throws Exception
 {    // gets connected to the Queue & checks the queue depth high event
            try
            {
                  mqConnect();
                  mqOpen();
                  chexQType();
                  mqClose();
                  mqDisconnect();
            }
            catch (Exception exp)
            {
                  exp.printStackTrace();
            }
      }      //getConnected ends here
      private void mqConnect() throws Exception
      {      // Connection to the queue manager
            try
            {
                  MQEnvironment.hostname    = hostName;
                  MQEnvironment.channel     = channel;
                  MQEnvironment.port        = Integer.parseInt(port);
System.out.println( hostName + "  ————— " + channel + "—————  " + port);
                  mqQueueManager = new MQQueueManager(qmgrName);
   System.out.println("Qmgr : " + qmgrName + " connection successfull");
            }
            catch ( MQException mqExp)
            {
```

© 2004. Reproduction prohibited. Please inform Xephon of any infringement.

```
                System.out.println("Error in queue manager connect....");
                    System.out.println("QMGR Name : " + qmgrName);
                    System.out.println("CC   : " + mqExp.completionCode );
                    System.out.println("RC   : " + mqExp.reasonCode);
              }
        }
        private void mqDisconnect() throws Exception
        {      // disconnect to queue manager
              try
              {
                    mqQueueManager.disconnect();
System.out.println("Qmgr : " + qmgrName + " disconnection successful ");
              }
              catch ( MQException mqExp)
              {
              System.out.println("Error in queue manager disconnect....");
                    System.out.println("QMGR Name : " + qmgrName);
                    System.out.println("CC   : " + mqExp.completionCode );
                    System.out.println("RC   : " + mqExp.reasonCode);
              }
        } // end of mqDisconnect
        private void mqOpen() throws MQException
        {
              try
              {
                    int openOption = Ø;
                    openOption = MQC.MQOO_INQUIRE;
      queue = mqQueueManager.accessQueue(qName,openOption,null,null,null);
                    System.out.println( "Open queue sucessfull... ");
              }
              catch ( MQException mqExp)
              {
                    System.out.println("Error in opening queue ....");
                    System.out.println("Queue Name : " + qName);
                    System.out.println("CC   : " + mqExp.completionCode );
                    System.out.println("RC   : " + mqExp.reasonCode);
              }
        } //end of mqOpen
        private void mqClose() throws MQException
        {
              try
              {
                    queue.close();
                    System.out.println("Close queue successfull.....");
              }
              catch (MQException mqExp)
              {
                    System.out.println("Error in closing queue ....");
                    System.out.println("Queue Name : " + qName);
                    System.out.println("CC   : " + mqExp.completionCode );
```


© 2004. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
                System.out.println("RC   : " + mqExp.reasonCode);
          }
    } // end of mqClose
    private void chexQType() throws MQException
    {      // for checking the queue type
          try
          {
                int[] qSelectors = new int[1];
                int[] qIntAttrs = new int[1];
                byte[] qCharAttrs = new byte[1];
                int MQIA_Q_TYPE = 20;
                qSelectors[0] = MQIA_Q_TYPE ;
                qInquire(qSelectors,qIntAttrs,qCharAttrs);
                qType(qIntAttrs[0]);
          }
          catch (MQException mqExp)
          {
                System.out.println("Error in closing queue ....");
                System.out.println("Queue Name : " + qName);
                System.out.println("CC   : " + mqExp.completionCode );
                System.out.println("RC   : " + mqExp.reasonCode);
          }
    }      //end of chexQType
    private void qAliasInquiry() throws MQException
    {      // to get the Base queue name for Alias queue
          try
          {
                int MQCA_BASE_Q_NAME = 2002;
                int MQ_Q_NAME_LENGTH = 48;
                int[] qAliasSelectors = new int[1];
                int[] qAliasIntAttrs = new int[0];
                byte[] qAliasCharAttrs = new byte[MQ_Q_NAME_LENGTH];
                qAliasSelectors [0] = MQCA_BASE_Q_NAME ;
                System.out.println(" in qAliasInquiry ...");
           qInquire(qAliasSelectors, qAliasIntAttrs, qAliasCharAttrs);
                System.out.println(" Base queue name  : " + new
String(qAliasCharAttrs));
          }
          catch (MQException mqExp)
          {
                System.out.println("Error in qAliasInquiry....");
                System.out.println("Queue Name : " + qName);
                System.out.println("CC   : " + mqExp.completionCode );
                System.out.println("RC   : " + mqExp.reasonCode);
          }
    }      // end qAliasInquiry
    private void qRemoteInquiry() throws MQException
    {      // to get the Remote queue properties
          try
          {
```

© 2004. Reproduction prohibited. Please inform Xephon of any infringement.

```
                int MQCA_REMOTE_Q_MGR_NAME = 2017;
                int MQCA_REMOTE_Q_NAME = 2018;
                int MQCA_XMIT_Q_NAME = 2024;
                int MQ_Q_NAME_LENGTH = 48;
                int MQ_Q_MGR_NAME_LENGTH = 48;
                int[] qAliasSelectors = new int[3];
                int[] qAliasIntAttrs = new int[0];
              byte[] qAliasCharAttrs = new byte[MQ_Q_MGR_NAME_LENGTH  +
 MQ_Q_NAME_LENGTH + MQ_Q_NAME_LENGTH ];
                qAliasSelectors [0] = MQCA_REMOTE_Q_MGR_NAME  ;
                qAliasSelectors [1] = MQCA_REMOTE_Q_NAME ;
                qAliasSelectors [2] = MQCA_XMIT_Q_NAME ;
                System.out.println(" in qRemoteInquiry...");
            qInquire(qAliasSelectors, qAliasIntAttrs, qAliasCharAttrs);
System.out.println(" Base queue name  : " + new String(qAliasCharAttrs,
 0, MQ_Q_MGR_NAME_LENGTH ));
 System.out.println(" Base queue name  : " + new String(qAliasCharAttrs,
 MQ_Q_MGR_NAME_LENGTH , MQ_Q_MGR_NAME_LENGTH ));
 System.out.println(" Base queue name  : " + new String(qAliasCharAttrs,
 MQ_Q_MGR_NAME_LENGTH + MQ_Q_NAME_LENGTH,  MQ_Q_NAME_LENGTH ));
          }
          catch (MQException mqExp)
          {
                System.out.println("Error in qRemoteInquiry....");
                System.out.println("Queue Name : " + qName);
                System.out.println("CC   : " + mqExp.completionCode );
                System.out.println("RC   : " + mqExp.reasonCode);
          }
     }      // end qAliasInquiry
     private void  qLocalInquiry () throws MQException
     {
          //int currentDepth =0;
          try
          {
                //queue type / usage
                int MQIA_USAGE = 12;
                 //queue inquire
                int MQIA_DEF_PRIORITY = 6;
                int MQCA_Q_DESC = 2013;
                int MQ_Q_DESC_LENGTH = 64;
                int MQIA_DEF_PERSISTENCE = 5;
                int MQIA_MAX_Q_DEPTH = 15;
                int MQIA_CURRENT_Q_DEPTH = 3;
                int MQIA_TRIGGER_CONTROL = 24;
                //for Event
                int MQIA_Q_DEPTH_HIGH_EVENT       = 43;
                int MQIA_Q_DEPTH_MAX_EVENT       = 42;
                int MQIA_Q_DEPTH_HIGH_LIMIT  = 40;
                int MQIA_Q_DEPTH_LOW_EVENT   = 44;
                int MQIA_Q_DEPTH_LOW_LIMIT   = 41;
```


© 2004. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
                  //for Triggering
                  int MQCA_INITIATION_Q_NAME = 2008;
                  int MQCA_PROCESS_NAME = 2012;
                  int MQ_PROCESS_NAME_LENGTH = 48;
                  int MQ_Q_NAME_LENGTH = 48;
                  int MQIA_TRIGGER_TYPE = 28;
                  int[] selectors = new int[14];
                  int[] intAttrs = new int[11];
                  byte[] charAttrs = new byte[MQ_Q_DESC_LENGTH +
MQ_Q_NAME_LENGTH  +  MQ_PROCESS_NAME_LENGTH ];
                  selectors[1] = MQCA_Q_DESC;
                  selectors[4] = MQIA_DEF_PERSISTENCE ;
                  selectors[5] = MQIA_MAX_Q_DEPTH ;
                  selectors[6] = MQIA_CURRENT_Q_DEPTH;
                  //for triggering
                  selectors[3] = MQCA_INITIATION_Q_NAME;
                  selectors[7] = MQIA_TRIGGER_CONTROL;
                  selectors[8] = MQIA_TRIGGER_TYPE ;
                  selectors[9] = MQCA_PROCESS_NAME ;
                  //for Event
                  selectors[0] = MQIA_Q_DEPTH_HIGH_EVENT;
                  selectors[2] = MQIA_Q_DEPTH_MAX_EVENT;
                  selectors[10] = MQIA_Q_DEPTH_HIGH_LIMIT ;
                  selectors[11] = MQIA_Q_DEPTH_LOW_EVENT ;
                  selectors[12] = MQIA_Q_DEPTH_LOW_LIMIT ;
                  selectors[13] = MQIA_USAGE ;
                  qInquire(selectors,intAttrs,charAttrs);
                  if ( intAttrs[10] == 0 )
                  {
                        System.out.println("Queue usage  Normal ");
                  }
                  else if ( intAttrs[10] == 1 )
                  {
                        System.out.println("Queue usage  XMIT ");
                  }
                 System.out.println("Default Priority = " + intAttrs[0]);
                 System.out.println("Description : " +
new String(charAttrs,0,MQ_Q_DESC_LENGTH));
               System.out.println("Q_DEPTH_MAX_EVENT = " + intAttrs[1]);
                  if ( intAttrs[1] == 1 )
                  {      // Event enabled
                  System.out.println("Q monitoring event is enabled " );
                        System.out.println ("Q high depth event  "  +
enableDisable(intAttrs[1]));
System.out.println ("Q high depth limit " + intAttrs [7]);
                        System.out.println ("Q low depth event  "  +
enableDisable(intAttrs[8]));
               System.out.println ("Q low depth limit " + intAttrs [9]);
                  }
               System.out.println("Q_DEF_PERSISTENCE = " + intAttrs[2]);
```

© 2004. Reproduction prohibited. Please inform Xephon of any infringement.

```
                System.out.println("Q_MAX_Depth = " + intAttrs[3]);
               System.out.println("CURRENT_Q_DEPTH = " +  intAttrs[4]);
            //Trigger is set On then displays the trigger properties....
                if ( intAttrs[5] == 1 )
                {
                      System.out.println("Trigger is On");
           //variable decalration for getting the Trigger control values
                       System.out.println("TRIGGER_TYPE = " +  intAttrs[6]);
                     if ( intAttrs[6] == 3)
                          System.out.println("TRIGGER_TYPE is Depth");
                     else if ( intAttrs[6] == 2)
                          System.out.println("TRIGGER_TYPE is Every");
                     else if ( intAttrs[6] == 1)
                          System.out.println("TRIGGER_TYPE is First");
                     else if ( intAttrs[6] == Ø)
                           System.out.println("TRIGGER_TYPE is None");
                     //Initiation queue name
                     System.out.println("Init Q Name : " + new
 String(charAttrs,MQ_Q_DESC_LENGTH,MQ_Q_NAME_LENGTH ));
                      System.out.println("Process defination Name : " + new
 String(charAttrs,MQ_Q_DESC_LENGTH + MQ_Q_NAME_LENGTH,
 MQ_PROCESS_NAME_LENGTH));
                }
//queue inquire ends here
                queue.close();
           }
           catch ( MQException mqExp)
           {
                System.out.println("Error in Inquiry queue ....");
                System.out.println("Queue Name : " + qName);
                System.out.println("CC   : " + mqExp.completionCode );
                System.out.println("RC   : " + mqExp.reasonCode);
           }
      }      // currDepth ends here
      private void qInquire(int[] selectors , int[] intAttrs , byte[]
 charAttrs ) throws MQException
      {      // MQ queue inqiury
           try
           {
                System.out.println(" in qInquire  ....");
                queue.inquire(selectors,intAttrs,charAttrs);
           }
           catch (MQException mqExp)
           {
                System.out.println("Error in Inquiry queue ....");
                System.out.println("Queue Name : " + qName);
                System.out.println("CC   : " + mqExp.completionCode );
                System.out.println("RC   : " + mqExp.reasonCode);
           }
      }      //end of qInquire
```


© 2004. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
    private String enableDisable(int parameter)
    {// to check the argument is enabled or disabled
          String str="";
          if ( parameter == 1 )
                str = "Enabled";
          else if (parameter == Ø )
                str = "Disabled";
          return str;
    }// end of enableDisable
    private void qType(int qTypeInt)
    {
          try
          {
                //Queue Types
                if ( qTypeInt == 1 )
                {
                      System.out.println(" Queue Type : Local ");
                      qLocalInquiry ();
                }
                else if (qTypeInt == 3)
                {
                      System.out.println(" Queue Type      : Alias ");
                      qAliasInquiry();
                }
                else if (qTypeInt == 6)
                {
                     System.out.println(" Queue Type      : Remote ");
                      qRemoteInquiry();
                }
                else if (qTypeInt == 6)
                {
                      System.out.println(" Cluster Queue  ");
System.out.println(" Clusrter queue are supported in this version ");
                }
          }
          catch (Exception e)
          {
                System.out.println("Error in qType function  ");
                e.printStackTrace();
          }
    }      //end of qType
   private void readPropertyFile(String propFileName) throws Exception
    {
          try
          {
            System.out.println("Reading from file  " + propFileName );
                Properties mqProperties = new Properties();
       mqProperties.load(getClass().getResourceAsStream(propFileName));
           hostName            = mqProperties.getProperty("hostname");
           qmgrName            = mqProperties.getProperty("qmgrName");
```

© 2004. Reproduction prohibited. Please inform Xephon of any infringement.

```
            port                    = mqProperties.getProperty("portNumber");
            channel                    = mqProperties.getProperty("channel");
            qName                     = mqProperties.getProperty("QueueName");
               } // end of try
               catch (Exception exp)
               {
                  System.out.println("Error in reading property file ....");
                      exp.printStackTrace();
                      System.exit(0);
               } // end of catch
         } // end of readPropertyFile
}
```

Balaji SR (balaji_srajan@yahoo.com)
MQ Administrator
eFunds International (India) © Xephon 2004

Web services and the enterprise business environment

There's a revolution occurring in enterprises and it's all about Web services producing a minimum 50% reduction in the Total Cost of Ownership (TCO) and up to 3000% Return On technology Investment (ROI). Its use is moving from early adopters to early majority in 2004. The most common usage is integration between mainframe (eg MVS, OS/390, z/OS, CICS), client/server, and Web-based environments (intranets/extranets/Internet). I spoke on this topic at Comdex Las Vegas in November 2003 to a capacity 'enterprise' audience because of the high level of interest.

In this article, we outline the advantages and impediments, discuss examples, define Web services from a business and a technical perspective, and, finally, end with a roadmap for implementation.

ADVANTAGES TO ENTERPRISES

Web services are critical to strategic consideration and planning

 © 2004. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

of the organization and business models, and the value chain and operations. Business models are heavily impacted by the universal integration abilities of Web services. The integration process occurs in three phases: internally, externally on a limited basis, and then broadly through the Internet. Web services are currently being used heavily for internal integration, and external integration will gain momentum in 2005/6.

Phase I integration occurs behind the firewall on the intranet. It involves enterprise systems, such as ERP and CRM, legacy mainframes, and client/server and Web-based systems, plus aggregating applications and sources for self-service portals. Next, Phase II integration occurs with a limited number of customers, suppliers, and partners through a company's extranet. Ford and Dell are good examples of companies in this phase. Finally, in Phase III, this is broadened to larger trading networks, e-markets, and the public using the Internet.

Using Web services, your business becomes more agile since you have microscopic and global visibility into all your operations, which allows for quick adjustments. You will have a faster time to market for new products and services since the supporting applications can be implemented 70% faster by re-using existing systems.

Web services provide tighter relationships with your customers, suppliers, and partners because it's easier for them to do business with you. Outcomes will include lower inventories throughout the supply chain, improved data quality, better business intelligence on sales and marketing programs, faster product design through collaboration, and higher sales from streamlined delivery of business services.

Overall, substantial savings are made in cost and time through less development and maintenance, improvements in productivity and efficiency through streamlined operations, reduction in redundant applications and processes, not using traditional Enterprise Application Integration (EAI) technologies, and managing/reusing existing legacy assets. Furthermore, resources can be redeployed for other strategically important initiatives.

© 2004. Reproduction prohibited. Please inform Xephon of any infringement.

In addition, Web services allow you to expand to new markets, new sources of revenue, and new business models by redesigning business processes. For example, a large credit-rating firm is using Web services to expand revenues from small and medium-sized businesses. Web services support portal initiatives and the resulting improvements in self-service can offer new revenue opportunities. T-mobile and Amazon are good examples.

IMPEDIMENTS TO IMPLEMENTING WEB SERVICES

Web services can be deployed over the Internet, which impacts on reliability and availability. XML (eXtensible Mark-up Language) underlies Web services, so bulkier text processing can impede performance.

Moreover, the standards that affect interoperability are still evolving. This problem is being addressed by the Web Services Interoperability (ws-i.org) organization, which publishes profiles and guidelines to ensure interoperation. Two major groups guiding the standards are the Organization for the Advancement of Structured Information Standards (oasis-open.org) and the World Wide Web Consortium (w3.org).

Added challenges come from the training required on the technology and service-oriented architecture. Finally, can you trust a publicly-available Web service discovered dynamically without user intervention, since trading-partner relationships are typically based on interpersonal exchanges?

ENTERPRISE BUSINESS CASES FOR WEB SERVICES

Merrill Lynch used Web services to integrate its CICS applications, saving 96% over using integration-bus technologies. They developed a Web services SOAP gateway to their CICS environment, allowing full access to the CICS applications with no adapters required for their client applications. And, they saved on licensing costs usually associated with EAI.

The Bank of Montreal is using Web services to provide a unified

© 2004. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

customer view of the bank's array of specialized IBM mainframe applications such as savings, loans, and investments.

The US Government in its single portal, E-travel system, is providing centralized travel management, saving $300m annually, yielding a 70% reduction in processing time, with an estimated 650% ROI. The system is using Web services to provide a single reservation system integrating a myriad of travel care providers, reservation/voucher systems, travel agencies, and travel payment reimbursement systems.

AT&T connects its TIRKS (Trunks Inventory Record Keeping System), first developed in 1960, to more than 100 other applications using Web services, resulting in a 78% saving in maintenance time. Prior to using Web services, any changes to TIRKS resulted in changes to the 100+ applications.

Wachovia, a financial services giant, is providing a consolidated view of customer information by using Web services to access numerous legacy CICS/DB2 back-end systems. Furthermore, Web services allow support for browsers, rich clients, interactive voice response systems, pagers, and wireless systems.

Dunn and Bradstreet (D&B) and Swiss Interbank Clearing (SIC) use Web services, replacing EDI (Electronic Data Interchange), to provide improved services to their clients. For example, SIC clears most cheques written in Switzerland. Using EDI, the process could take up to six days in contrast to the one day using Web services.

In a recent discussion, Michael Liebow, Vice President of Web Services for IBM's Global Services Division, provided additional examples of mainframe integration. These are summarized below.

Arkansas Blue Cross and Blue Shield and its affiliates generate 12 million manual contacts (phone, fax, e-mail) annually. Arkansas Blue Cross wants to replace the inefficient tangle of phone and paper-based transactions with direct, secure, and HIPAA-compliant processes. Web services applications will put billing and claims processes online, bringing an anticipated 50%

© 2004. Reproduction prohibited. Please inform Xephon of any infringement.

reduction in its 12 million manual contacts, as well as a 20% increase in efficiency in filing and processing claims.

Miami Dade County is already using Web services projects to improve government operations and enhance the services it offers to its citizens. Web services will allow 40 departments to reuse existing mainframe functionality as they build new e-government applications. For example, Web services allow contractors to apply for and pay for building licences and permits on-line, and the information is automatically directed to the appropriate building inspectors. Web services also power its new 311 non-emergency information phone number. It will be the key in providing operator access to all county information – regardless of what system it is sitting on – to answer callers' questions.

Visa's network includes more than 21,000 members, 396 million cardholders, and millions of merchants in the USA. It's no surprise that the company must deal with charge-backs and dispute-resolution as part of the normal course of business. The company's most recent Web services project, Resolve Online, lets banks resolve charge-backs over the Internet and automates the dispute process over its network. Now, most cardholder disputes will be resolved within just one billing cycle.

Huntington Bank provides innovative retail and commercial financial products and services to more than 300 regional banking offices, and has been serving the financial needs of its customers for more than 137 years. Over the past decade, Huntington Bank experienced significant growth through acquisitions and increased cross sales, which led to a more complex IT infrastructure to support a diverse mix of products, services, and delivery channels. Implementing advanced Web services technologies, Huntington was able to drive customer information across lines of business at every delivery channel and reduce redundant software by 60%. By streamlining their application infrastructure, the bank has reduced maintenance costs and can now take a more focused and strategic approach to future software development. Web services also allow the bank to have instant access to all its customer data, which

© 2004. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

reduces the amount of time customers spend on the phone with customer service representatives. Other Web services related savings come from ease in deploying, maintaining, and upgrading systems.

WHAT ARE WEB SERVICES?

A business-centric viewpoint

Picture Web services as applications exposed as standards-based services. More than 100 open source and commercial toolkits are available, allowing applications developed under any environment to be easily (somewhat automatically) exposed as Web services. These services can be used by other applications bridging internal applications or across trading partner supply and demand chains. Using intranets and the Internet, any system can easily connect to any other system. Web services allow Business Process Management (BPM), permitting different platforms, such as SAP and Oracle, to exchange information. Moreover, Web services provide real-time visibility to all aspects of the business value chain, demand chain, and supply chain, enabling Business Activity Monitoring (BAM) and alerting users to imminent problems. Web services enable the collection and computation of analytic information and optimization through the evaluation of complex ‘what if’ scenarios.

A technological perspective

This time, picture Web services as an Application Program Interface (API) based on XML and open standards – a WS-API. Web services can be built from an existing application simply and easily by using one of the many toolkits available. Since it’s standards-based, with all major vendors in agreement, a WS-API supports Application-To-Application Communication (APPC) from any system to any other system across an internal intranet or across the Internet. This allows loosely-coupled connections, services reuse, and combining services into composite services.

WS-API loosely-coupled connections are flexible, not brittle, and

© 2004. Reproduction prohibited. Please inform Xephon of any infringement.

do not break when applications are modified. Changing one application doesn't require rewriting connected applications, much like in the AT&T TIRKS example. No expensive integration middleware, integration-bus technology, or adaptors are required to connect different systems.

In addition, once a Web service is created, it can be reused. For example, a travel reservation service can be reused in the following ways:

1 To provide Business Intelligence (BI) by analysing reservation trends in a BI application.

2 To provide customer service representatives with customer travel reservation history in a CRM application.

3 To allow customers to make travel reservations and obtain reservation status information through a self-service travel portal.

4 To allow management reporting on reservations.

Composite services are formed by combining services. This permits business agility or the quick reaction to new trends and business needs. In addition, end-to-end business processes can be modelled and automated.

Because Web services are based on XML, they allow universal connectivity and interoperability between applications of any type. And this is across platforms, operating systems, and programming languages. The use of XML provides a standard way of representing data, understandable by all systems and vendors, and is human/machine readable as illustrated in this XML example:

```
<order>
  <quantity>5</quantity>
  <price>4.50</price>
</order>
```

Web services are founded on universally-recognized standards. The primary ones are HTTP, SOAP, WSDL, and UDDI. HTTP and SOAP are used to communicate with a Web service. WSDL

© 2004. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

is used to describe a service, can be auto-generated from service code, and can be used to auto-connect with a service. UDDI is used to register Web services so they can be found and then used, even automatically. A more in-depth description follows.

HTTP

HTTP (HypertText Transfer Protocol) is used as a transport for message exchanges with a Web service. Protocols such as SMTP (Simple Mail Transport Protocol), FTP (File Transfer Protocol), and others can be used; however, HTTP has the advantage that it's the same protocol for seeing Web pages, it goes through firewalls, and is widely supported and pervasive.

SOAP

SOAP (formerly Simple Object Access Protocol) is now just a name as of Version 1.2. It is the XML-based envelope used to hold message exchanges with a Web service. The messages are typically XML documents containing business information or method (service) calls. SOAP is valuable in its own right since it can be used as a messaging system for generic Enterprise Application Integration (EAI), replacing the need for expensive messaging software or middleware. An added advantage of SOAP is that it can be generated automatically by the Web services toolkits. Moreover, SOAP can be extended for added capabilities such as security and management and is supported by all systems and vendors.

WSDL

WSDL (Web Services Description Language – pronounced Wiz Del) is used in WSDL documents. WSDL, which is automatically generated by Web services toolkits from application code, describes in a human/machine readable form what a service does, what operations it can perform, how you communicate with it, and where it's located. WSDL describes the Web services' interface in a standardized way that can be understood and used by anyone from any development environment (Java, C#, VB, COBOL, etc) without requiring knowledge of the internal workings

© 2004. Reproduction prohibited. Please inform Xephon of any infringement.

of the software component. You can think of it as a generic API design document allowing code reuse – a COBOL programmer can easily use a Java component for example. You can also use an application's associated WSDL document to auto-generate the Web services connection code for the client software.

UDDI

UDDI (Universal Description Discovery and Integration) offers the ability to submit and register Web services in a private or public registry so they can be manually or programmatically/ dynamically (automatically) found and then used. UDDI provides the ability to describe the owner of the software, contact information, the services offered by the software, information on how to use it, and where it's located. Furthermore, UDDI provides the ability to do searches based on categories. Using a phone book analogy, you can think of a UDDI registry consisting of searchable white pages listing business information, yellow pages categorizing services, and green pages containing the technical details about what the service does, how you communicate with it, and where it's located. Public UDDI registries are offered by companies such as Microsoft, IBM, and SAP. A private UDDI registry allows a company or trading partner network to list available Web services. This provides an added function as a software catalogue to allow reuse and prevent duplication – to manage your software assets.

A ROADMAP FOR IMPLEMENTING WEB SERVICES

Here are some guidelines if you wish to implement Web services:

1 Gain knowledge on Web services technologies and educate at all levels to realize the strategic and operational opportunities that result in a reduction in TCO and provide a high ROI.

2 Create a centre for excellence that promotes best practices, technology reuse, and development patterns.

3 Look at what you are doing, what platforms you are using, and conduct an application inventory.

© 2004. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

4 Look to see what others are doing.

5 Define standards/policies/guidelines on how you will conduct your Web services implementation.

6 Look for service candidates with the greatest impact; for example, recurring processes involving many users, areas where there's dynamic or continual change, business processes involving many manual steps or manual intervention, or areas where you can consolidate views to increase employee productivity.

7 Pilot an internal integration project.

8 Fine-tune your standards/policies.

9 Repeat steps 1 to 7 using Web services to integrate your internal systems as much as possible.

10 Try a few of your closest trading partners – prioritize to the top 20% who affect 80% of your revenues.

11 Expand to a broader market only when the standards are mature.

Stephen Ibaraki
Board of Directors, http://www.hr-online.com/
www.StephenIbaraki.com (Canada)
© Stephen Ibaraki 2004

Application Serialization with WMQ

Starting with Version 5.1 WebSphere, MQ introduced a not-widely-recognized feature called Application Serialization. It was introduced as part of the support for shared queues on z/OS. This article describes the problem that is solved with this feature, the different possibilities it offers, and how to use it.

PROBLEM DESCRIPTION

For some applications it is necessary to process messages in the

© 2004. Reproduction prohibited. Please inform Xephon of any infringement.

same sequence as they arrive on the queue. On all platforms supported by WebSphere MQ, it is possible to open message queues in exclusive mode. This assures that only one application can get messages from that message queue. But there are other scenarios that cannot be resolved with exclusive access to one queue only. Examples are partitioned queues that are accessed by indexes, for example when there is key information in the message id (MsgId), or correlation id (CorrelId), or both fields of the message descriptor (MQMD). One application is processing messages with key value A while others process messages with key value B.

When using exclusive access to a message queue, the first application may terminate abnormally. The second application can then open the queue and start processing messages. This processing can start before the recovery of the messages being processed by the first application has completed. This could result in processing messages out of sequence.

Another scenario is serialization across different queue managers. Exclusive access to a message queue works only when accessing the same queue in the same queue manager.

For some application scenarios it is necessary to process and correlate messages from multiple message queues. Exclusive access covers processing messages for only one queue.

A further, more theoretical, problem is where applications exclusively access multiple message queues in different orders. Such a scenario can occur for WMQ-based application platforms like message brokers.

APPLICATION SERIALIZATION

To solve these problems, WebSphere MQ offers Application Serialization. In contrast to exclusive access on a message queue, Application Serialization works at the connection level. Because there is no message queue object involved at connection time, another identification item is needed. This item is the so-called queue manager connection tag (connection tag). The

© 2004. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

connection tag is defined by an application programmer. It should be unique for the applications that need to be serialized. The connection tag is a character field that allows for 128 bytes. This is sufficient to find a tag that is unique for a type of application. If you're using the connection tag within a product, you could use a similar naming schema as for Java packages, for example com.ibm.dni.myapplication. In this example dni is a product identifier for the IBM product WebSphere Business Integration for Financial Networks (for details see http://www.ibm.com/software/integration/wbifn).

The connection tag is defined in the connection options (MQCNO). It requires connection options Version 3, which has been supported since WebSphere MQ Version 5.1 on z/OS. A example showing how to specify the connection tag is given below:

```
#include <cmqc.h>
...
MQCNO    connection_options = { MQCNO_DEFAULT };
MQLONG   rc, cc;
MQCHAR48 queue_manager;
MQHCONN  connection_handle;

/* make sure to use connection options version 3 */
connection_options.Version = MQCNO_VERSION_3;
connection_options.Options = MQCNO_SERIALIZE_CONN_TAG_QSG;
strcpy ( connection_options.ConnTag, "com.ibm.dni.myapplication" );
strcpy ( queue_manager,              "MQØ1" );

MQCONNX ( queue_manager, &connection_options, &connection_handle, &cc,
&rc );
...
```

The version in the connection options structure must be explicitly set to MQCNO_VERSION_3 because the default version is Version 1.

Once the MQCONNX call is successful, any other application that tries to connect using the same connection tag gets an unsuccessful return code MQRC_CONN_TAG_IN_USE (2271). This return code occurs until the first application disconnects using the MQDISC function. If the application holding the connection tag ends abnormally, there is also a difference in the situation where the serialization is done using exclusive access

© 2004. Reproduction prohibited. Please inform Xephon of any infringement.

to a queue. A second application cannot connect using the connection tag until recovery for the crashed application is completed. This assures that messages can be processed only if there is no recovery outstanding and therefore the sequential processing of all messages can be assured.

SERIALIZATION SCOPE

Application Serialization can be assured in different situations. First of all, the scope for the serialization can be a single WebSphere MQ queue manager. If a connection tag is in use by the first application for one queue manager, other applications that want to connect to the same queue manager using the same connection tag are rejected. Applications connecting to another queue manager are not affected by the first application even if they use the same connection tag.

The second choice for the serialization scope is the scope of a WebSphere MQ Queue Sharing Group (QSG). All queue managers within the QSG can be either on one z/OS logical partition (LPAR) or on different LPARs. An application can request that serialization should occur for the whole QSG. If an application is connected to a queue manager using a connection tag and that queue manager is part of a QSG, other applications trying to connect to any queue manager in the same QSG using the same connection tag is rejected. Applications connecting to a queue manager that are not part of the QSG are not affected by the first application even if they use the same connection tag.

SERIALIZATION TYPES

Besides the serialization scope, there is another behaviour that an application can control. This is the type of serialization that is required for the connection tag. WebSphere MQ supports two types of serialization:

1 The connection tag is used to serialize applications. This type allows just one application to connect within the serialization scope. Even the same application can connect only once using the same connection tag.

© 2004. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

2 The connection tag is used to restrict applications. This type allows multiple connections within the same serialization scope to be successful. The second MQCONNX succeeds if it was issued from within the same processing scope as the first MQCONNX with the same connection token. The processing scope for z/OS is defined to be the same MVS address space. Within one MVS address space, you can run one process with multiple threads (or TCBs in MVS terms), multiple processes with a single thread, or a mix of both. All of them can connect using the same connection tag.

Applications running outside the processing scope and trying to connect using the same connection tag will get a return code MQRC_CONN_TAG_IN_USE.

SERIALIZATION OPTIONS

The serialization scope and the serialization type that an application requests is controlled using the *options* field in the MQCNO structure used for the MQCONNX call. Any combination of serialization scope and type is allowed so that there are four different values that can be used:

- MQCNO_SERIALIZE_CONN_TAG_Q_MGR – this option serializes the usage of the connection tag in the scope of a queue manager.
- MQCNO_SERIALIZE_CON_TAG_QSG – this option serializes the usage of the connection tag in the scope of a QSG.
- MQCNO_RESTRICT_CONN_TAG_Q_MGR – this option restricts the usage to the connection tag in the scope of a queue manager.
- MQCNO_RESTRICT_CONN_TAG_QSG – this option restricts the usage to the connection tag in the scope of a QSG.

These values have to be used exclusively. How to set this is shown in the example above with the value

© 2004. Reproduction prohibited. Please inform Xephon of any infringement.

MQCNO_SERIALIZE_CONN_TAG_QSG. As mentioned above, the options take effect only if the version in the MQCNO structure is set to 3.

If you use the QSG options for the MQCONNX and the queue manager you are connecting to is not part of a QSG, then the behaviour falls back to its corresponding options for the queue manager; for example, MQCNO_SERIALIZE_CON_TAG_QSG falls back to MQCNO_SERIALIZE_CONN_TAG_Q_MGR.

USAGE SCENARIOS

There are several scenarios for which Application Serialization can be used. The main one has already been described. Application Serialization can be used to assure that messages in a WebSphere MQ message queue can be guaranteed to be processed in the sequence in which they are stored in the queue. Be aware that WebSphere MQ does not guarantee that messages will always arrive in sequence when putting them onto a queue.

Application Serialization can also be applied to a set of applications that need to work together. An example might be where a program implements a protocol conversion, eg between a program that is processing WebSphere MQ messages and another program that is processing, say, TCP/IP requests, where messages are retrieved from a message queue as well as others having to be written to a message queue . Because protocols are often asynchronous for a connection, a preferred implementation is by using different threads. Because each thread needs to connect to a queue manager, Application Serialization with the restrict option can be used to protect a set of queues at once. If the protocol conversion needs to implement a kind of session and the program is able to support multiple sessions, the connection tag could be used to represent the session.

Another main use for Application Serialization is for allowing the implementation of a hot stand-by system. A hot stand-by system is where one application is actively processing messages and another application is already active with all the resources, but is

© 2004. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

not processing any of the messages. The hot stand-by application becomes active only if the primary application fails. This kind of system is especially interesting in a parallel sysplex environment with shared queues.

Hot stand-by can be used for applications that process messages needing to be processed in sequence and a response must be assured within a predefined time. In such situations, a long start-up time for a second instance of the application is not allowed.

There is one drawback when using Application Serialization for hot stand-by systems. The problem is that the hot stand-by application gets just the reason code MQRC_CONN_TAG_IN_USE and there is no way of knowing when a connection tag is no longer in use. This requires the stand-by application to be regularly polling using the MQCONNX call.

SUMMARY

With Application Serialization, WebSphere MQ offers capabilities to serialize or restrict access to applications. This can be used to guarantee that messages on a message queue are processed in sequence and to implement hot stand-by environments. Application Serialization can be applied for a single queue manager but also across a queue sharing group.

Michael Groetzner
IBM (Germany)
© IBM 2004

WebSphere Integrator, writing a plug-in input node

TARGET AUDIENCE

This article is aimed at C programmers who need to implement a WebSphere MQ Integrator C plug-in node that creates an entirely new message tree with a *Properties* folder and an *MQMD*

© 2004. Reproduction prohibited. Please inform Xephon of any infringement.

folder and is then propagated to and correctly parsed by the IBM primitive nodes and written to a bitstream. The reader must be familiar with WebSphere MQ Integrator and the C plug-in interface to make best use of this article.

WRITING A PLUG-IN INPUT NODE (WMQI)

The simplest course of action to take in an input node is to create a new message with a *Properties* and an *MQMD* folder that can be passed to, for example, the IBM primitive MQOutputNode. This is accomplished by calling two CNI functions, cniCreateMessage and cniCreateElementAsFirstChildUsingParser (or suitable alternatives), to create the default *Properties* and *MQMD* folders. Uninitialized fields are populated by the relevant parser as and when necessary through the message flow:

```
int rc;
CciChar* constMQHMD = (CciChar*)CciString("MQHMD", BIP_DEF_COMP_CCSID);
CciChar* constMQMD = (CciChar*)CciString("MQMD", BIP_DEF_COMP_CCSID);
CciChar* constMqPropertyParser = (CciChar*)CciString("MQPROPERTYPARSER",
BIP_DEF_COMP_CCSID);
CciChar* constProperties = (CciChar*)CciString("Properties",
BIP_DEF_COMP_CCSID);
CciElement* element;
/* inMessage is passed by the broker but for all intent purposes is
blank */
CciMessageContext* inMessageContext = cniGetMessageContext(&rc,
inMessage);
CciMessage* outMessage = cniCreateMessage(&rc, inMessageContext);
element = cniCreateElementAsFirstChildUsingParser(&rc, outMesage,
constMQHMD);
cniSetElementName(&rc, element, constMQMD);
cniSetElementType(&rc, element, CCI_ELEMENT_TYPE_NAME);
element = cniCreateElementAsFirstChildUsingParser(&rc, outMesage,
constMqPropertyParser);
cniSetElementName(&rc, element, constProperties);
cniSetElementType(&rc, element, CCI_ELEMENT_TYPE_NAME);
cniFinalize(&rc, outMessage);
cniPropagate(&rc, outMessage);
```

Note:

- Broker plug-in nodes should always check for non-zero (0 != rc) return codes.

© 2004. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

- The C compiler may predicate that all declarations are finished before functions are called.
- The *MQMD* folder is created first so that when cniCreateElementAsFirstChildUsingParser is called for the *Properties* folder, the *Properties* folder becomes the first folder in the message tree.

PROPAGATING MESSAGES FROM A USER-WRITTEN PLUG-IN INPUT NODE

In most broker plug-in nodes, messages are propagated from the cniEvaluate function. For a plug-in input node, messages are propagated using cniRun; they can also be propagated from the cniEvaluate function if the node defines this function, and if another node is plumbed to an input terminal of the plug-in input node. This behaviour is not well documented and can be confusing: suffice it to say that a plug-in input node has all the same functionality as a normal plug-in node and can also initiate a message flow by creating and propagating a new message without needing to receive a message on an input terminal. In fact, a plug-in input node does not normally define any input terminals since it is normally only ever dispatched directly by the broker. The user-written code programmatically sets a pointer to cniRun inside the bipGetMessageflowNodeFactory function of the loadable implementation library, in an identical way to setting a pointer for cniEvaluate in a standard plug-in node. When a message flow is deployed, the broker calls the cniRun function of each plug-in input node. The first noteworthy difference between the cniEvaluate and cniRun functions concerns message context: cniEvaluate typically creates messages based on the context of the message passed to it; whereas cniRun creates messages based instead on the context of a blank message passed as a parameter of the cniRun function, which has the context of the broker. If a different context is required it can either be set programatically in the user code, set by a compute node or database node plumbed to the output terminal of the plug-in node, or by a mechanism described below.

© 2004. Reproduction prohibited. Please inform Xephon of any infringement.

COMMON MISTAKES WITH PROPERTIES AND MQMD FOLDERS

New messages created by a plug-in input node have no *Properties* folder and no *MQMD* folder. The presence of an *MQMD* folder is not mandatory since the message may never end up being propagated to an MQOutputNode. The *Properties* folder has more uses, especially in the ResetContentDescriptor node. The omission of these two folders can create problems that can be difficult to remedy or understand, so the following information will be useful.

If a new message is passed to an MQOutputNode, the message tree must contain an *MQMD* folder, and the MQOutputNode property 'Message Context' cannot be left as the default or set to either 'Pass All' or 'Pass Identity' if there is no message context set in the *MQMD* folder – these problems are made apparent by finding a related reason code in the system log. The property value of 'Set All', 'Set Identity', and 'Default' can be used, but the context of the message on the queue will be set to that of the broker (unless it is changed *en route*). The property value of 'None' also works, but applications that rely on context will not find this useful. If context is required to differentiate messages on the output queue, and the broker context is not practical, a compute node, database node, or alternative means should be used to set the context before propagating the message.

CNIRUN COMPLETION CODES: TRANSACTION RESULT, STATE INFORMATION

The second noteworthy difference between cniRun and cniEvaluate is the dispatching mechanism: cniRun is dispatched directly by the broker, and cniEvaluate from a message propagated to an input terminal of the node. A third difference is that while cniEvaluate returns a single-meaning completion code to the broker, cniRun returns a double-meaning completion code to reflect the message flow transaction result and state information. The completion codes are: CCI_TIMEOUT, CCI_SUCCESS_CONTINUE, CCI_SUCCESS_RETURN, CCI_FAILURE_CONTINUE, and CCI_FAILURE_RETURN.

© 2004. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

The following notes give general guidelines on how the completion codes are interpreted when returned to the broker:

- CCI_TIMEOUT is used to tell the broker 'nothing happened' – no messages have been propagated out of the node and the node should be dispatched again.
- CCI_SUCCESS_CONTINUE and CCI_FAILURE_CONTINUE also indicate that the node should be re-dispatched, whereas CCI_SUCCESS_RETURN and CCI_FAILURE_RETURN indicate that the node should not be re-dispatched – the broker will not invoke the node until after the redeployment of message flows incorporating the node.
- The use of the 'SUCCESS' completion codes gets the broker to commit transactions because of messages being propagated out of the plug-in node for this invocation of cniRun – ie this time the plug-in node was dispatched by the broker. 'FAILURE' completion codes get the broker to back out transactions for this invocation of cniRun.

Messages propagated from a single call to cniRun or cniEvaluate are encapsulated in a unit of work managed by the broker. (An example of an exception is when an MQInputNode or MQOutputNode is configured to handle messages not in the transaction. These are handled by that node.)

A well-behaved plug-in node frequently returns control to the broker, especially in cases where many messages are propagated in an iterative fashion – for example processing records in a file. If the node does not yield in this way, the broker is prevented from stopping (or restarting) the data flow engine, publishing configuration changes, or redeploying message flows within the same execution group.

If it is critical that the broker dispatches the plug-in node on the very same thread, the cniRun function must not use either of the 'RETURN' completion codes since this indicates that the node has finished processing and does need to be re-dispatched. If the node is subsequently dispatched again (for example, a message

© 2004. Reproduction prohibited. Please inform Xephon of any infringement.

flow containing the node is redeployed) the broker may use any of its available threads with no guarantee of ever using the same thread as before. To ensure the cniRun function is dispatched on the same thread, CCI_TIMEOUT or either of the 'CONTINUE' completion codes should be used.

Note: if a 'RETURN' completion code is returned by cniRun and the plug-in node also declares cniEvaluate, cniEvaluate can continue to be called if the plug-in node has its input terminal connected (directly or indirectly) to an input node that has at least one thread dispatched and is propagating messages.

THREAD AFFINITY

The message context passed to either the cniRun or cniEvaluate functions is not valid across future calls to the plug-in node. This limits the scope of any one message context to the function call in-progress. A handle on the message tree (or part of it) is immediately invalid when control returns to the broker. However, the message tree can be written to a bitstream and reconstructed in a different scope, since this constitutes taking a copy of the message tree rather than passing it as a reference. For example the *MQMD* folder of a message tree passed to cniEvaluate can be stored for use later in the cniRun function for creating new messages using the context of the message passed to cniEvaluate (recall that the message context passed to the cniRun function is that of the broker).

Delegating to cniRun is particularly useful for adhering to POSIX restrictions such as thread affinity on file descriptors, where a file must be opened and closed on the same thread. The open/close call can be deferred for processing in the cniRun function, providing a 'RETURN' completion code was not used last time cniRun returned control to the broker. Although this problem does not often arise, it is one example of where thread affinity must be addressed when architecting a plug-in node to be used in a multi-threaded environment. Careful consideration is required to avoid imposing a scalability restriction (ie a point of synchronization/performance bottleneck) when the message

© 2004. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

flow is configured for additional instances. It is also difficult to correctly handle exceptions that may need to be passed from cniRun back to the cniEvaluate function.

In a real life scenario, additional instances of a message flow write data into the same file. A message that arrives when a file is not open causes a new file to be opened. A message that arrives and results in certain conditions being met closes the file. The net result may effectively require the file to be 'opened' on one thread and 'closed' on another. This violates POSIX rules on file descriptor thread affinity, so the cniEvaluate defers the open/close operation to the cniRun function. This is not a standard use of cniRun – a typical application is MQInputNode, where messages are read from a queue and propagated out to the message flow.

MEMORY UTILIZATION

Consider implementing a C plug-in input node that processes the records of a file, creating and propagating a new message for each record in the file. For a small file with few records there is little adverse impact on the broker. As the number of records in the file increases, it become obvious that memory utilization and processing time per record also increase. For a file with hundreds of thousands of records, this side effect is severe. The broker cannot allocate sufficient memory to process all the records and causes an abnormal termination of the execution group in which the message flow is running. Memory is required to create a new message tree and syntax elements attached to it. This memory is available for re-use when the node that initiated the message flow returns control to the broker with a completion code other than CCI_TIMEOUT.

This effect can be addressed in one of two ways. The first is the preferred method and the more reliable of the two. It requires saving state between consecutive calls to the plug-in node (either the broker dispatching cniRun, or a message propagated to an input terminal). The second approach is both undocumented and unsupported. Nevertheless, it is presented here as a tried-and-tested solution architected by the author for propagating

© 2004. Reproduction prohibited. Please inform Xephon of any infringement.

300,000 messages without memory heap expansion and associated performance degradation. A message tree and syntax element are created into which the user data is copied, the handle to the syntax element is retained and used to copy new data over old. The updated message tree is propagated each time round the loop. The first method is obviously more difficult to implement and manage state between invocations. The second method has obvious limitations in that the message tree structure must essentially remain unchanged, otherwise memory utilization will increase as before for each structural change to the tree.

The actual implementation of propagating every record without returning control to the broker contradicts best practice, but was judged to be an acceptable disadvantage in the rare event of an extraordinarily large number of records. It was deemed necessary for the broker to wait for all the records to complete before the execution group could be interrupted, and it was also decided that the performance benefits of processing all records using a single message tree outweighed the disadvantage of holding up configuration changes or redeployment of other message flows, which could always be allocated in an alternative execution group.

SUMMARY

The important points to note from this article are:

- A new message created by a plug-in node must have a *Properties* folder in order to be correctly processed by many of the IBM primitive nodes, especially the MQOutputNode, and for configurations of the ResetContentDescriptor node.
- A new message created by a plug-in node must have an *MQMD* folder that does not specify the default message context of ‘Pass All’ or ‘Pass Identity’ if the message is propagated to an MQOutputNode, unless it is set by a compute node or database node, or by some other means.
- Using the C programming interface, syntax elements are

© 2004. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

retained by the broker and consume memory until control is returned to the broker.

This article has explained how to create a new message with the necessary parser folders that can be propagated without causing parsing errors. It has also discussed considerations for propagating an essentially unlimited number of messages without running into memory problems.

Efficient use of memory is one of two reasons for a plug-in input node to frequently return control to the broker: it is also desired to allow the data flow engine to update configuration information, and redeploy some or all of the message flows in an execution group. A node that returns control frequently to the broker will fit in better with the architecture of the Integrator product.

A FINAL WORD OF CAUTION

The reader must note that re-using syntax elements is an approach that has been used successfully by the author in a production environment with no discernible side effects, but may not necessarily be the preferred design for versions of WebSphere MQ Integrator other than V2.1 or when using the Java programming interface. It is the reader's responsibility to implement a plug-in node with careful attention to data integrity.

With thanks to John Hosie (IBM) and Vicente Suarez (IBM).

Alexander Russell
IT Specialist
IBM Hursley (UK)

© IBM 2004

Using WebSphere MQ as a JNDI repository for JMS administrable objects

JMS applications are designed to be 'provider' independent, that is, they should have no code that pertains only to WebSphere

© 2004. Reproduction prohibited. Please inform Xephon of any infringement.

MQ. This is so that they could easily be taken and used with another JMS provider without changing the code. JMS applications would generally get a connection to the messaging transport by obtaining a connection factory object and calling a method createConnection() on it. A connection factory provides a connection to the messaging transport, and destination objects provide a connection to a specific destination on that transport (such as a queue). These objects are called 'administrable objects'. Obtaining administrable objects would normally be done by a look-up from a repository using JNDI (Java Naming and Directory Interface). A JNDI repository can be anything from an LDAP server to a file system, and an administrator would have previously defined these connection factories and bound them into the repository. Using this system a JMS application has no repository-specific or messaging transport-specific code because it is written using these standard interfaces.

This can add some complexity when getting your first JMS application up and running. You would have to create your messaging transport environment, create a repository, define your administrable objects, and then write your JMS application. However, by using WebSphere MQ and Support Pac ME01, you can combine the jobs of messaging transport and JNDI repository into one, and thus set-up can be greatly reduced. Consequently, a JMS application can be up and running in much less time.

JMS IN WEBSPHERE MQ

The WebSphere MQ classes for Java provide a JMS implementation and have been part of the full WMQ product since MQ Version 5.3 (previously it was a downloadable Support Pac). Provided with these classes is a JMS administration tool (JMSAdmin), which will create JMS administrable objects that work with WebSphere MQ and stash them in a repository that provides a JNDI implementation. Administrable objects are the only JMS objects that can contain transport-specific information, and, as such, need to be created with a provider-written tool. For example, JMSAdmin will allow you to create a connection factory

© 2004. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

that defines the name of the queue manager or the name of the SVRCONN channel to use – a property which may mean nothing to another messaging transport.

As someone who has written many JMS applications for WebSphere MQ, I have often found that a large amount of information is repeated in defining the queue manager and the queues, and the administrable objects. For example, if my JMS application wishes to put a message on a queue located on a queue manager I would need to:

- Define a queue manager called QM1.
- Define a queue called QUEUE1.

Similarly I would have to create a JNDI repository and use JMSAdmin to perform the following operations:

- Define a QueueConnectionFactory that points to queue manager QM1.
- Define a Queue type destination that points to queue QUEUE1.

As you can see, information has been duplicated in the administrable object definition. More work would be needed if the queue manager were on a remote machine because not only would you have to set up the SVRCONN channel and listener on the queue manager, but the QueueConnectionFactory would also need to contain exactly the same information.

In a production environment, you would want to do this. This is so that it would be possible for an administrator to change properties on your system (such as the hostname of the queue manager) while your applications continued working seamlessly without alteration. However, during development it is not always desirable to have to set up a JNDI repository as well as a queue manager.

One solution to this is to use Support Pac ME01. This Support Pac turns a WebSphere MQ queue manager into a JNDI repository for administrable objects. That sounds like a good idea

© 2004. Reproduction prohibited. Please inform Xephon of any infringement.

because it means that all definitions are in one place. However, the main advantage of this Support Pac is that it is able to dynamically generate administrable objects, based on real objects on the queue manager, with no user intervention. For example, if you use ME01 to connect to the queue manager we created earlier, you would be able to look up an object called QM1 and a QueueConnectionFactory would be returned to your application with the correct properties set on it – without needing to take the JMSAdmin tool out of the box.

USING ME01

ME01 comes shipped as a single jar that contains a set of JNDI implementation classes. There are three main operations that a JNDI implementation should support (although there are many more that it could) and these are:

- bind() – bind or store an object into the repository.
- lookup() – look up or retrieve an object from the repository.
- list() – return a list of all the objects that are stored in the repository.

ME01 can operate in two modes – advanced or basic. Let's look at the basic mode initially. In this mode, ME01 will allow you to bind and look up Queue type destinations for each WMQ queue, and also allow you to look up a single QueueConnectionFactory that will provide a connection to the queue manager. For example, if you use ME01 against a queue manager called QM1 with three queues defined on it (QUEUE1, QUEUE2, and QUEUE3) a call to lookup("QUEUE1") will return you a JMS Queue type destination that points to QUEUE1.

What this means is that, with no explicit JNDI configuration, a JMS application can be instantly used with this queue manager.

The administrable object that is returned is one with all the default properties set on it. The only property that is modified by ME01 is the queue name (for Queue type destinations) and the queue

© 2004. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

manager name, host name, port number, and SVRCONN channel name for the QueueConnectionFactory.

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

Gareth Matthews
WebSphere Platform Messaging Developer
IBM (UK)
© IBM 2004

© 2004. Reproduction prohibited. Please inform Xephon of any infringement.

MQ news

MQSoftware has announced that its flagship Q Pasa! middleware management system will provide monitoring support for WebSphere MQ for TPF (Transaction Processing Facility) used by airlines, hotels, and financial institutions worldwide. It provides a dashboard showingf MQ metrics, queues, events, and associated technologies.

For further information contact:
MQSoftware, 1660 South Highway 100, Suite 400, Minneapolis, MN 55416, USA.
Tel: (952) 345 8720.
URL: http://www.mqsoftware.com/news/newsDetail.jsp?id=60.

* * *

Sonic Software has released Sonic ESB 5.5, which makes it possible for companies to build an event-driven, Service-Oriented Architecture (SOA) that can adapt to changing business requirements.

Sonic ESB 5.5 incorporates Sonic Continuous Availability Architecture (CAA), which reduces the time required for the communications infrastructure of the ESB to resume operations after hardware, software, or network failures, and guarantees that transactions are not lost or rolled back.

For further information contact:
Sonic Software, 14 Oak Park, Bedford, MA 01730, USA.
Tel: (781) 999 7000.
URL: http://www.sonicsoftware.com/products/sonic_esb/index.ssp.

* * *

IBM has announced Versions 5.1.2 of WebSphere Studio Application Developer and WebSphere Studio Site Developer.

These are designed to simplify the development of Web user interfaces, business logic, and interactive portals. New rapid application development and code generation tools in WebSphere Studio automate many tasks and hide Java complexities, simultaneously reducing the learning curve for Java novices, while accelerating development.

For further information contact your local IBM representative.
URL: http://www-306.ibm.com/software/awdtools/studioappdev/about.

* * *

TeamQuest Software has announced TeamQuest Performance 9.1 with scalability, statistical analysis, and multi-system modelling advancements. The new version of the performance management and capacity planning software provides additional capabilities, making it easier to predict performance when servers are added to horizontally scale a tiered network of servers. The product now includes agents for WebSphere, DB2 UDB, and EMC.

For further information contact:
Teamquest, One TeamQuest Way, Clear Lake, IA 50428, USA.
Tel: (641) 357 2700.
URL: http://www.teamquest.com/newsletter/2004/1q/highlights.shtml.

* * *