59

MQ update

May 2004

In this issue

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

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.

Printed in England.

Interpreting WebSphere MQ messages

INTRODUCTION

WMQ allows two applications to communicate using messages. This is also possible in a network of computers with different architectures and operating systems. The application sending the message doesn't have a problem, it just formats a message and sends the data, and doesn't need to worry about its content. If the sending application is building the message as expected by the receiving application there are no communication problems. But in some cases the receiving application can have problems, for example it might be that:

- The format is not as expected. This may be because the sending application made a mistake when creating the message, or the messages of another application are routed by mistake to the input queue of the receiving application.
- The format of the message is correct but the content is not as expected. This can happen when messages are sent with unexpected codepages or encoding.
- The message type is not as expected. There are parameters in the messages that control the kinds of message that should be returned. If these settings are not correct, the application may not work.
- The message never reaches the input queue of the application. It may be that a message is sent to the wrong queue or that it ends up in a dead letter queue.

In some cases one just needs to verify the messages that are created by an application.

ApAart from all these reasons, it is important anyway to be able to analyse and understand the messages in a WMQ message queue. How to analyse the content of a message, what problems can occur, and how to handle them are demonstrated using a tool

program that is able to display messages in any queue and thereby analyse any WMQ-defined formats and interpret their content.

This article reviews first the general structure of WMQ messages, followed by the handling required to receive a message, and how to parse the message. Then we describe some special considerations for dynamic data structures.

A REMINDER

To demonstrate the kinds of problem that can occur if the content of a message is not as expected by the receiving application, let's just remind ourselves what WMQ messages look like.

A WMQ message consists of two parts: the WMQ message descriptor (MQMD) and the message data. The MQMD, besides some context information for the message data, describes the message data in terms of message data format name, message encoding, and coded character set ID (CCSID). The message data format name, or just format, indicates how the following data is structured. The encoding tells the receiver of the message how numbers in the message data are encoded. This encoding is different for different processor architectures. The encoding field describes the actual format of integer numbers, floating-point numbers, and decimals.

The CCSID describes how characters in the message data should be interpreted. In general there are two main character encodings, ASCII (American Standard Code for Information Interchange) and EBCDIC (Extended Binary Coded Data Interchange Code), but also within these encoding schemes there are different CCSIDs for different languages, eg German or English. In addition to ASCII and EBCDIC characters, WMQ also supports Unicode CCSIDs within some WMQ headers.

The message data itself can be completely described by the MQMD if the data consists of only one data structure. In general the message data can consist of different parts, intermediate headers for intermediate applications (eg middleware), and a

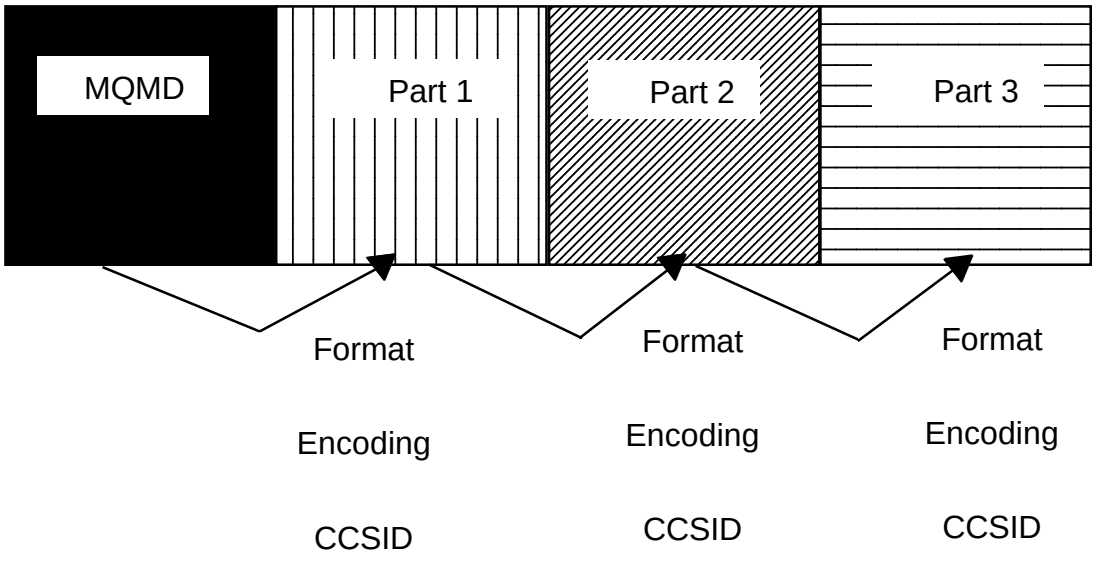

Figure 1: WMQ message structure

final part containing the information necessary to run your business. To be able to understand the following parts in the message, each part must describe the following part in the same terms, format, encoding, and CCSID. Each part in the message data therefore can have a different encoding and CCSID. Some parts are already predefined WMQ data structures. These predefined formats are described in *WMQ Application Programming,* reference SC33-1673.

Even if handled as different parameters in the WMQ API, the MQMD and the message data can be seen as one stream of bytes. This eases the description. This structure is shown in Figure 1 for a message that consists of an MQMD, two headers and the final application data.

RECEIVING THE MESSAGE

The tool program contains a main function. This is called with a queue name and, optionally, the name of a queue manager. It does the usual:

```
MQCONN ()
MQOPEN ()
```

This is followed by a loop of MQGET() calls. The following special characteristic is interesting. The open is done only for browsing. This is because the program should not destroy the message. The message may be reprocessed later when the problem is identified and the program is corrected to handle the message. The MQGET could be issued with parameters that force WMQ to convert the message, but this is not done because problems could be caused by the conversion process, or the conversion itself could fail.

The complete message is received into one buffer according to Figure 1. Afterwards the parser for the MQMD is retrieved from a parser factory. Then the parser is invoked for the first part of the message. Each parser combines with the data to parse the information about encoding and the CCSID of the message part to parse. Generally this is not known, but for the MQMD returned by the MQGET it is assured that the CCSID is in the encoding standard of the platform the program is running on, and the character data in the MQMD is either in ASCII or EBCDIC depending on the operating system the receiving platform is running on.

PARSING THE MESSAGE

A parser's function is to analyse and understand a part of a message. Each parser parses and displays the message part for which it is responsible. To do this, the parser first checks whether it can understand the part by looking to see whether the data length is sufficient. Most WMQ-supported formats start with an eye-catcher and a version number. These can also validate that the message can be handled by the program. As an example, an extract from the MQMD parser is shown below:

```
if ( length < sizeof ( MQMD )
 ...
  printf ( "Incorrect version or size.\n"
           "Size is %u, expected is %u, version is %u\n",
           length, sizeof ( MQMD ),
           local_enc ( mqmd->Version, encoding ) );
  return (MGERR_INVALID_FORMAT);
} // endif
```

```
  if ( !is_correct_StrucId ( mqmd->StrucId, MQMD_STRUC_ID, &converter )
) {
    printf ( "Incorrect Structure ID: " );
    display_data ( mqmd->StrucId, sizeof ( mqmd->StrucId ), 24 );
    return (MGERR_INVALID_FORMAT);
  }/* endif */
  // Convert if necessary
  if ( (encoding     & MQENC_INTEGER_MASK) !=
       (MQENC_NATIVE & MQENC_INTEGER_MASK) ) {
    /* integer conversion required */
    mqmd->Version        = conv_int ( mqmd->Version );
    ...
  }/* endif */
  if ( converter != NULL ) {
    /* character conversion required */
    converter ( mqmd->Format, sizeof ( mqmd->Format ));
    ...
  }/* endif */
  /* Output MQMD content */
  printf ( "\tVersion: %u\n", mqmd->Version );
  ...
  if ( ( mqmd->Version == MQMD_VERSION_1 && length == sizeof (MQMD) ) )
{
     // No data with MQMD available
     return ( Ø );
  }/* endif */
  /* Pass on to next formatter if necessary */
  formatter = format_factory->getFormatter ( mqmd->Format );
  if ( formatter == NULL ) {
    printf ( "No formatter for format !!!!!!\n" );
  } else {
    return ( formatter->format ( mqmd + 1,
                                 length - sizeof ( MQMD ),
                                 mqmd->CodedCharSetId,
                                 mqmd->Encoding ) );
  }/* endif */
  return ( Ø );
```

The length check shown here is simplified because different versions of the MQMD structure are already available from different versions of the WMQ product.

The eye-catcher's check is not that simple because the character field is in the CCSID passed to the current formatter. In general, from the CCSID it's not obvious whether the character data is in ASCII or EBCDIC. One trick that helps here is based on the fact that the characters used for the eye-catcher are restricted to being in the common part of either all ASCII or EBCDIC codepages.

Therefore the eye-catcher is first checked against the expected value for the platform the program is running on.

If this fails, the function converts the actual value from ASCII to EBCDIC on an EBCDIC platform or *vice versa* on an ASCII platform. Then the comparison for a correct eye-catcher is repeated. If this also fails, then it is assumed that the eye-catcher doesn't match. In the case that the actual data is converted, the function *is_correct_StructId()*, which carries out the checks for eye-catchers, returns the converter used to convert the data. This converter is used later by the formatter to convert all other character data in the actual header.

A similar problem also arises for the integer fields in each header. The encoding itself is encoded within the encoding field. WMQ headers contain only an integer number – no floating-point values or decimal numbers. The extracted encoding for integers is compared with the encoding of the local platform. This local encoding for integers can be retrieved as shown in the code example from the constant MQENC_NATIVE. For each platform this is correctly defined in the WMQ header file. For integers there are only two valid forms of encoding – normal (as on most processor architectures) and reversed (as, for example, for the processors in a PC). So in the case of the integer encodings not matching, all integer fields within the actual header will be converted.

INTERPRETING THE MESSAGE

Most tools that are available to analyse messages display just the values of each field in the WMQ headers. For a 'normal' WMQ programmer these values are not meaningful because he is usually familiar with the constants as described in the *WMQ Application Programming Guide*. The tool program converts the values into the WMQ constants as described for the header. To allow for this, there is a set of helper functions for each type of value:

- Completion codes.

- Reason codes.
- Persistent values.
- Feedback codes.
- Encoding values.
- Application types.
- Message flags.
- Integer attributes.
- Character attributes.
- Command values.

The helper functions can be found in the file *mgmq.cpp*. Usually, each version of WMQ extends the list of supported values for these fields. That's why the helper functions need to be checked whenever a new version of WMQ becomes available or when compiling the program on another platform.

The program is capable of analysing and interpreting the following WMQ headers and messages:

- Message descriptor.
- Transmission header.
- Dead letter queue header.
- Work information header.
- Trigger messages.
- Strings and command formats.
- Reference message header.
- Request and format header.
- Programmable command format.
- Event messages.
- Administration messages.

- CICS and IMS information headers.
- Distribution header.
- Binary messages.

Details of all the WMQ-supported formats can be found in the *WMQ Application Programming Guide,* reference SC33-1673.

PARSERS WITH SPECIAL HANDLING

Most WMQ-supported headers have a fixed structure with a defined number of elements, but there are some headers that require special handling. These structures are listed below:

- Unknown – this is the simplest parser. An unknown format means that the structure cannot be interpreted. In this case the data is handled as a binary large object (BLOB). The BLOB parser is used for the defined format MQFMT_UNKNOWN and it is the parser that is used as the default for any format that is not in the list of known and supported message elements of the program.

  The handling of the BLOB parser is very simple. It just formats and prints out the data in hex and character format. The parser doesn't care about the CCSID and encoding of the data. This way the data could be interpreted by a human reader.

- String and command formats – WMQ supports various formats that contain only character strings. MQFMT_STRING is the format that should be used for user applications, but there are other formats, eg MQFMT_CMD, that are used for commands to a command server running within the queue manager that contain only characters. While processing the data, the parser itself doesn't take the CCSID into account. The main reason for this is that there is no rule to identify whether a CCSID is EBCDIC, ASCII, or Unicode. The same method as used in the 'standard' headers is not working because there is no structure identifier that can be used as an indication for the character encoding.

- CICS and IMS information headers – these are ‘normal’ headers with a fixed structure and a defined number of elements. But the WMQ documentation states that the Encoding and CCSID fields are reserved and have no meaning. Their values are initialized with zero. For these formats the appropriate parsers assume that the following structure is in the same Encoding and CCSID as the one for the current header.
- Programmable Command Format (PCF) – this is a WMQ internal, self-definable, dynamic data structure. Examples of where PCF format is used in WMQ include command messages to define and delete WMQ objects, such as queues or name lists, and also for events.

  The PCF structure consists of a short fixed header and a variable number of structures for each element. The structures are different for each supported type of element. Each structure starts with an identification about what type of element it is.

  The use of PCF character strings, arrays of character strings, integer numbers, and arrays of integer numbers is supported. Floating point numbers and decimals are not supported. The structure that describes each element in the PCF header contains detailed information about the element, eg the length, number of array elements, the encoding, or the CCSID.

  For details about the PCF structure and how to use it refer to the *WMQ Programmable System Management Guide*, reference SC33-1482.
- Request and Format headers – WMQ supports two versions of the Request and Format Header (MQRFH): Version 1 and Version 2 (MQRFH2). Both headers start with a fixed structure. This is followed by a variable length structure. In Version 1 the variable part consists of a number of name-value pairs. The MQRFH parser of the tool program tries to parse and analyse this variable part.

With MQRFH2, the variable part of the header is an XML-like data structure. This structure can only be a set of predefined Unicode CCSIDs. The MQRFH parser prints this part in hex and converts the data from Unicode to character format. The RFH parser does not try to analyse the XML-like structure. This would require integrating an XML parser.

PARSING THE REMAINDER OF THE MESSAGE

Once a parser has interpreted the content of its part of the message, it has to allow the remaining part to be parsed. Because all parser functions should be independent of available parsers and supported message formats, the knowledge about all existing parsers and the formats that they are responsible for is concentrated in a parser factory.

The parser factory is the central point where the tool program can be extended. The parser factory can be found in the file *mgFormat.cpp* amongst the source code provided with this article, available from www.xephon.com/extras/mqmay04.zip.

The factory maps the format name to a parser. Each parser is a function that gets a buffer, the length of the buffer, plus the encoding and CCSID values for the buffer. If the parser detects that the buffer is longer than the data it can interpret, it retrieves the name of the following format from the current message part. This format name is then used to ask the factory for the parser that is responsible for the remaining part of the buffer. The actual parser hands the remaining buffer, with the remaining length of the buffer, to the parser returned by the factory. In addition, the new parser gets the encoding and CCSID that the actual parser gets from its part of the buffer.

BUILDING THE TOOL PROGRAM

The tool program supplied with the source code is written in C/C++ and should compile on all WMQ-supported platforms. Nevertheless, the code is not completely platform-neutral. The main reason is that there are slight differences in the supported

constants and encodings. Because each new version of WMQ usually introduces new constants, the versions on different platforms will not always match, and it may be that some constants are missing or new ones need to be added.

The tool program was compiled and tested on z/OS and Linux. A makefile for the z/OS platform is part of the code. On z/OS the distribution header is not supported.

POSSIBLE ENHANCEMENTS

The tool program is usable as it is. Nevertheless, it may be enhanced with some additional options by any developer, depending on where and how the program will be used. Such enhancements may include:

- An option to convert the message. At the moment, the program uses the MQGMO_CONVERT option when retrieving messages from a queue. An option can be added to the command line parameters that allows switching of this option.
- Parsing of MQRFH2 NameValueData. Currently the program simply prints out the XML-like structure without analysing or interpreting it. The program could be enhanced by integrating an XML parser to check whether the data is well-formed and can show the structure.
- Change the output format. The program prints the messages in its own format. There are SupportPacs available that can construct a WMQ message using a text description as the input. The output format could be changed to match the input format of that particular SupportPac. This would allow messages to be reconstructed for test purposes, for example.

SUMMARY

Understanding WMQ messages by a program is very complex as any part of the message can have different encodings and CCSIDs. The tool program, located at www.xephon.com/extras/

mqmay04.zip, demonstrates how to analyse messages from any WMQ message queue. It shows each header, interprets each field, and shows its field with the corresponding WMQ constant, all of which improves our understanding of the messages.

The program is delivered as source code, which allows developers to extend the program with message formats that are used by their applications.

The program has already been proved to be useful when messages are not handled correctly, or just for looking at the messages that are generated by different programs.

Michael Groetzner
IBM (Germany)

Configuring a Web client to show a list of authorized users while transferring work items

The general interface provided out-of-the-box with MQSeries Workflow allows the transfer of work items. It lets logged-on users transfer work items to other users to whom they are authorized. Suppose the logged-on user is TESTMGR, and TESTMGR would like to transfer a work item to, say, ADHOCTEST01. The user needs to click on the transfer button shown in Figure 1.

This will bring up a Java pop-up as shown in Figure 2 that prompts the user to enter the name of the person to whom the item needs to be transferred.

User (TESTMGR) needs to remember the ID of the person to whom the work is being transferred and has to type in the text box provided and click *OK*.

The whole process is shown in Figures 2 and 3.

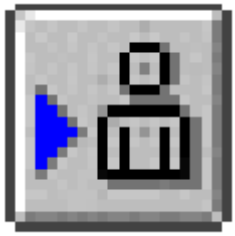

Figure 1: Transfer button

This is quite manageable if the number of users reporting to TESTMGR is low and TESTMGR can remember each individual's ID. But what if there are hundreds of workers reporting to TESTMGR? What if the workers are on a continuous job rotation and the group reporting to TESTMGR is continually changing? I am not sure if anybody else has faced this situation, but we have. So, in this article, I would like to present the solution we found to solve this problem.

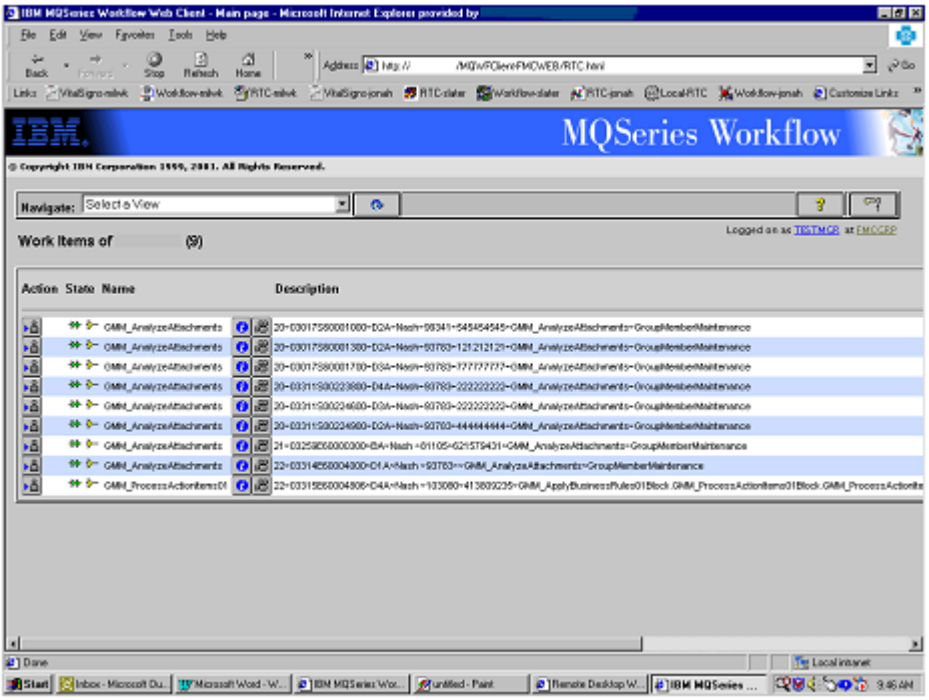

Figure 2: Work list view without code changes

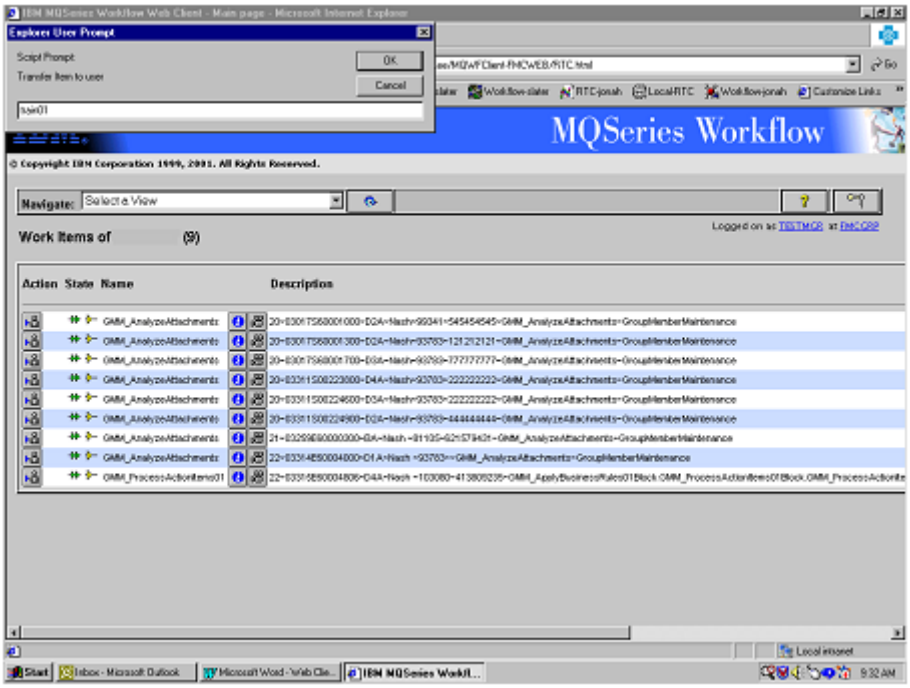

Figure 3: Transferring works item to another user

I would like to thank the IBM MQ Workflow team for providing useful tips. The modifications I have made to ListViewer.jsp are intended to help the MQ Workflow development community and it's not my intention to incur any copyright infringement issues.

SOLUTION

When a user logs on as TESTMGR, we present the same interface as given in the out-of-the-box client. But when the user (TESTMGR) clicks on the *Transfer Item* button, as in Figure1, we show a pop-up window containing a list of users to whom TESTMGR could transfer the work item. The list comprises user ID, first name, middle initial, and last name. TESTMGR can select the ID/name to whom the work item needs to be transferred and click on the *Transfer Item* button on the pop-up window. Alternatively the user (TESTMGR) may choose to cancel the transfer, in which case the *Cancel* button needs to be clicked or

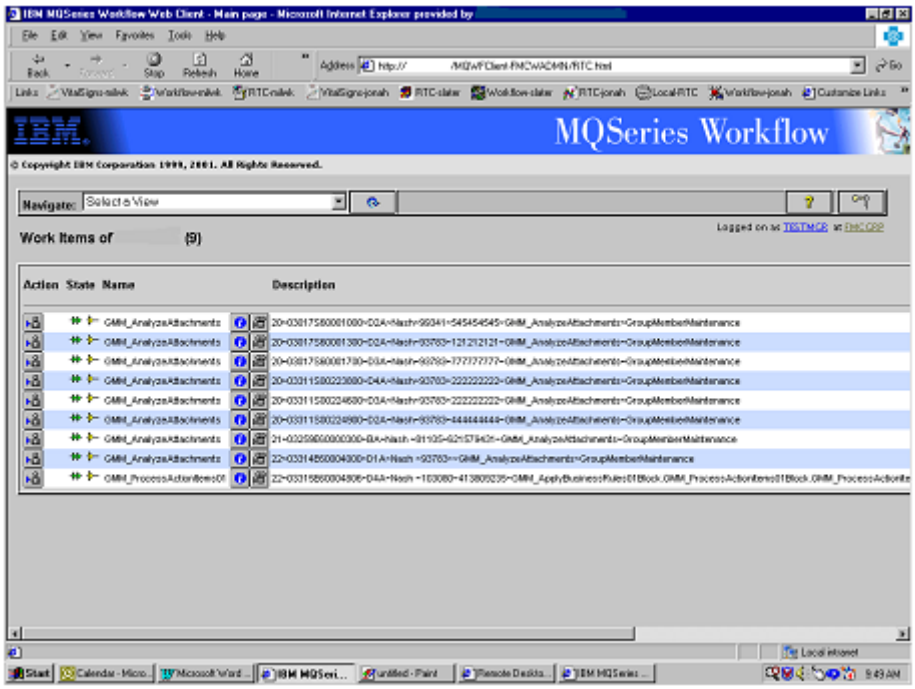

Figure 4: Work list view with code changes

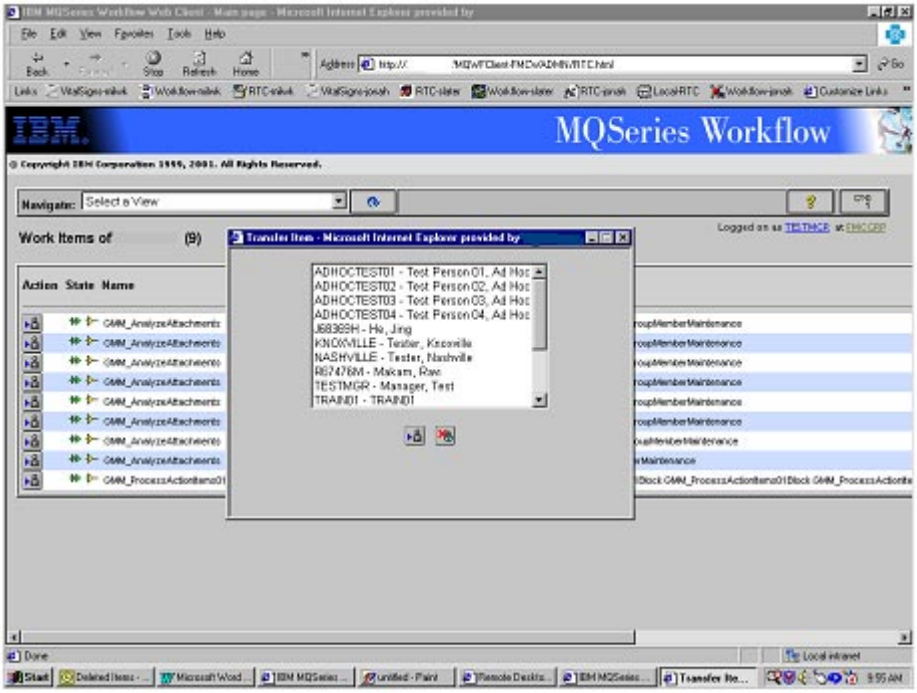

Figure 5: Transferring work item with code changes

the pop-up window closed.

We, in fact, modified our custom Web client interface to have the new functionality. However, to explain it to the user community I have modified ListViewer.jsp to demonstrate how the functionality can be added.

The modifications are shown in Figures 4 and 5.

CODE

ListViewer.jsp

We have to modify ListViewer.jsp, which can be found at *\Program Files\MQSeries Workflow\cfgs\(Your webclient configuration)\WebClient\webpages\forms*. Add the code exactly as shown. To make it clear, I have started each code modification with 'code change # begins' and ended each one with 'code change # ends'. I have included the IBM-supplied code to make it clear where the changes are to be made.

In ListViewer.jsp we added a function *getUsersAuthorizedFor*. This function performs the following:

- Checks the authorizations for user (TESTMGR).
- Then gets personsAuthorizedFor: for (TESTMGR).
- For each personId in the *personsAuthorizedFor* list, gets the firstname, lastname, middlename using *persistentPerson(personID)*.

  Puts each ID in Treemap

  end for.

```
<%@ page language="java" contentType="text/html"
errorPage="ViewError.jsp"
         import="java.util.*,com.ibm.workflow.api.*,
                        com.ibm.workflow.api.ProcessInstancePackage.*,c
om.ibm.workflow.api.ItemPackage.*,com.ibm.workflow.servlet.client.*"
%><jsp:useBean id="context" scope="request"
type="com.ibm.workflow.servlet.client.RequestContext"
/><% context.setLocale(response); %>
```

```
<html>
<!-- Create the page title ------------------------------------>
<% String user = context.getUserID();
   String distype = "", refreshCommand = "";
   int count = Ø, type = SessionContext.NOT_SET;
   PersistentList list = null;
   //code change #Ø1 begins
   TreeMap authorizedUsers = null;
   String userId = context.getUserID();;
  authorizedUsers =
getUsersAuthorizedFor(context.getExecutionService().persistentPerson(userId),
context.getExecutionService());
%><%! private TreeMap getUsersAuthorizedFor(Person workflowUser,
ExecutionService wfService) {
      TreeMap userNameReference = new TreeMap();
                                              //TreeMap to be returned.
      String[] usersAuthorizedFor = null;
                     //List of userIds current user is authorized for.
      String currentId = null; //UserId of current logged on user
      String firstName = null; //User First Name
      String middleName = null; //User Middle Name
      String lastName = null; //User Last Name
      String fullName = null; //Formatted Full Name of user
      Person thisPerson = null; //Workflow Person object
      //Get list of users for whom this person is authorized.
      try {
            currentId = workflowUser.userID();
// **** Important ****
//  PersonsAuthorizedFor()
//  Returns the persons for whom this person is authorized either
//   explicitly or by being a
//  substitute. If the person is authorized for all other persons,
//  then no person is returned here. For example if "ADMIN" is
//   authorized for all users
//  then usersAuthorizedFor  will contain null list.
//  Please see MQ Workflow programming guide for further information.
            usersAuthorizedFor = workflowUser.personsAuthorizedFor();
      } catch (FmcException xcpt) {
            return null;
      }
      //Retrieve information about the current logged on user
      //  and store in the TreeMap.
      try {
            //Get info about the current logged on user.
            thisPerson = workflowUser;
            firstName = thisPerson.firstName();
            middleName = thisPerson.middleName();
            lastName = thisPerson.lastName();
            //Format the Full Name text for the current logged on user.
```

```
            if ((lastName == null) || (firstName == null)) {
                  fullName = currentId;
            } else {
                  if (middleName == null) {
                        middleName = "";
                  }
            fullName = lastName + ", " + firstName + " " + middleName;
            }
  //Store username info about the current logged on user Id in TreeMap.
            userNameReference.put(currentId, fullName);
      } catch (FmcException xcpt) {
      }
      // For each user ID for which the current Workflow User is
      // authorized, find the first name, middle name and last name.
      for (int i = 0; i < usersAuthorizedFor.length; i++) {
            try {
                  //Retrieve UserName Info for authorized user.
                  // *** Important ****
  // The user who is trying to get the list of users for whom he/she is
         // authorized  needs to have "AUTHORIZED_FOR STAFF" authority.
 // This authorization can be set in Buildtime.  When defining the
 // person under Buildtime see to that the check box "Staff definition"
 // under "Authorizations" tab is checked. Make sure the exported FDL
                  // contains "AUTHORIZED_FOR STAFF".
 // Please see MQ Workflow programming guide & Getting started with
                  // Buildtime for further information.
                  thisPerson =
wfService.persistentPerson(usersAuthorizedFor[i]);
                  currentId = thisPerson.userID();
                  firstName = thisPerson.firstName();
                  middleName = thisPerson.middleName();
                  lastName = thisPerson.lastName();
                  //Format the Full Name of the authorized user.
                  if ((lastName == null) || (firstName == null)) {
                        fullName = currentId;
                  } else {
                        if (middleName == null) {
                              middleName = "";
                        }
            fullName = lastName + ", " + firstName + " " + middleName;
                  }
           //Store username info for the authorized user Id in TreeMap.
              userNameReference.put(usersAuthorizedFor[i], fullName);
            } catch (FmcException xcpt) {
            }
      }
      /*try {
      } catch (FmcException xcpt) {
            //Do nothing.
      }
```

```
*/
      return userNameReference;
}
//code change #01 ends
%><%
   if (context.getInstances() != null)
   {
     distype = context.get("ListViewer.ProcessInstances");
     type = SessionContext.INSTANCELIST;
     refreshCommand = "queryProcessInstances";
     count = context.getInstances().length;
     list = context.getInstanceList(context.getInstanceListOid());
   }
   *** *** ***
   *** *** ***
   *** *** ***
  <script language='JavaScript'>
  function fillForm(doc, name, owner, user, cmd, canEmail)
  {
    doc.open("text/html");
 doc.writeln("<html><head><title><%=context.get("ListViewer.TransferWorkItem")%></
title>");
    doc.writeln("<link rel='stylesheet' type='text/css' href='../
webclientstyle.css'>");
    doc.writeln("</head><body><form name='data'><div width='100%'
class='title'>");
    doc.writeln("<%=context.get("ListViewer.TransferWorkItemToPrefix")%>
<i>" + name + "</i>
<%=context.get("ListViewer.TransferWorkItemToPostfix")%><br><br></
div>");
 doc.writeln("<center><table><tr><label><td><b><%=context.get("ListViewer.UserID")%></
b></td><td><input type='text' name='userID' value='" + user + "'
size='30'></td></label></tr>");
    if (canEmail == 'true')
doc.writeln("<tr><label><td><b><%=context.get("ListViewer.EmailAddress")%></
b></td><td><input type='text' name='email' size='30'></td></label></
tr>");
    doc.writeln("</table><p><input type='button'
value='<%=context.get("ListViewer.Transfer")%>' " +
                "onClick='javascript:if (data.userID.value ==
&quot;&quot;) alert(&quot;You must provide a User ID&quot;); else { var
sub=&quot;" + cmd +
                "&quot; + &quot;&userID=&quot; + data.userID.value" +
                (canEmail == 'true' ? "+ &quot;&email=&quot; +
data.email.value;" : ";"));
    doc.writeln("window.opener.location.href = sub;
window.close();}'>");
    doc.writeln("<input type='button'
value='<%=context.get("Command.Cancel")%>'
```

```
onClick='javascript:window.close();'></center></p></form></body></
html>");
    doc.close();
    doc.data.userID.focus();
  }
//code change #Ø2 begins
// Insert include file transferInclude that conatins necessary code
// segement for popping up of window with names of the users populated
            <%@ include file="../forms/transferInclude.inc" %>

            function newWindow() {
                  if (window.screen) {
                        window.moveTo(Ø, Ø);
                        window.resizeTo(1Ø24, 74Ø);
                  }
                  if (window.opener && !window.opener.closed) {
                        window.opener.close();
                  }
            }
//code change #Ø2 ends
  </script>
</head>
<body>
<!-- code change #Ø3 begins
-->
<FORM NAME="TransferForm" method="post"
action='<%=context.getServletName()%>'>
      <INPUT type="hidden" name="command" value="transferItem">
      <INPUT type="hidden" name="id" value="">
      <INPUT type="hidden" name="userID" value="">
</FORM>
<!-- code change #Ø3 ends
-->
<%=context.openForm(refreshCommand, list == null ? null :
list.persistentOid())%>
<table width="1ØØ%" border="2" bordercolordark="#ØØØØØØ"
bordercolorlight="#FFFFFF" cellpadding="Ø" cellspacing="Ø"
class="navigator">
  <tr>
  <td align="left" nowrap>
    <!-- Create the navigation combobox ------------------------------>
   *** *** ***
   *** *** ***
   *** *** ***
  <%} /* End of for ---------- Process Templates ---------- */
  else if (type == SessionContext.WORKLIST)
  {
    Config  cfg      = context.getConfig();
    boolean viaEmail = cfg.getParameter("SMTP", "Host") != null;     //
```

```
A configured SMTP host triggers email enablement
    int cpn = context.getProcessInstanceNotifications().length;
    int can = context.getActivityInstanceNotifications().length;
    int cwi = context.getWorkItems().length;
    for (int i = 0; i < count; i++)
    {
      try
      {
        ProcessInstanceNotification processNotification = null;
        ActivityInstanceNotification activityNotification = null;
        WorkItem workItem = null;

        Item item = null;
        Command[] cmds = null;

        if (i < cpn)
        {
          item = processNotification =
             context.getProcessInstanceNotifications()[i];
          cmds = Command.getActions(processNotification);
        }
        else if (i < cpn + can)
        {
          item = activityNotification =
             context.getActivityInstanceNotifications()[i - cpn];
          cmds = Command.getActions(activityNotification);
        }
        else
        {
          item = workItem = context.getWorkItems()[i - cpn - can];
          cmds = Command.getActions(user, workItem);
        }

        String oid = item.persistentOid();
        String name = item.name();
      %>

      <tr class="<%=row[i % row.length]%>">

      <td nowrap><%
      if (cmds.length == 0)
        {%> <%}
      else for (int j = 0; j < cmds.length; ++j)
      {
             //code04
             %>
        <%
        if (!"transferItem".equals(cmds[j].getCommand())) {
          %><%=cmds[j].getTriggerTag(context, oid, name)%><%
```

```
        }
              //code change #Ø4 begins
              else if ("transferItem".equals(cmds[j].getCommand())) {
          %><%
              String url1= "javascript:{}\"
onclick=\"javascript:TransferWorkitemWindow('" + oid + "')\"";
        %><a href="<%=url1%>"><%=cmds[j].getActionIcon(context,
imageAttr)%></a><%
          //code change #Ø4 ends
              } else if (!viaEmail) {
          %><%=cmds[j].getTriggerTag(context, oid, user)%><%
        }
        else
        {
//          String url = "javascript:{}\" onClick=\"javascript: var xfer
= window.open('','Transfer_Item','width=4ØØ,height=19Ø,screenX=1ØØ,
screenY=1ØØ,dependent=yes,resizable=yes, menubar=yes, status=yes');" +
          String url = "javascript:{}\" onClick=\"javascript: var xfer =
window.open('','Transfer_Item','width=4ØØ,height=19Ø,screenX=1ØØ,screenY=1ØØ,
dependent=yes,resizable=yes');" +
                       " fillForm(xfer.document, '" + name + "', '" +
item.owner() +
                                  "', '" + user + "', '" +
                                  context.getCommand("x-transferItem",
oid) +
                                  "', '" + viaEmail + "');";
          %><a href="<%=url%>"><%=cmds[j].getActionIcon(context,
imageAttr)%></a><%
        }
      }%></td>
   *** *** ***
   *** *** ***
   *** *** ***

    } /* End of for ---------- List of Lists ---------- */

  } %>
</table>

</td></tr></table>

</body></html>
```

transferInclude.inc

Transferinclude.inc contains basic JavaScript that takes the contents of TreeMap built in the *getUsersAuthorizedFor* function and populates it in the form of a scrollable list. The functionality

for the *Transfer* button on the pop-up window is similar to the out-of-the-box client.

Please keep transferInclude.inc under *\Program Files\MQSeries Workflow\cfgs\(Your webclientconfiguration)\WebClient\webpages\forms.*

```
-  function TransferWorkitemWindow(WorkItemOid)
  {
      var w = window.open("", "TransferWindow", "width=450, height=300,
resizable=no", false);
      var d = w.document;
      var transferString = "<%= context.getCommand("transferItem", "")
%>" + WorkItemOid;

      TransferForm.id.value=WorkItemOid;

      window.name="WorkListPage";
      d.open();
      d.writeln('<HTML><HEAD><TITLE>Transfer Item</TITLE>');
      d.writeln('<link rel="stylesheet" type="text/css" href="../
webclientstyle.css">');
      d.writeln('<META HTTP-EQUIV="Pragma" CONTENT="no-cache"><META
HTTP-EQUIV="Expires" CONTENT="-1">');
      d.writeln('</HEAD><BODY onblur="window.focus();" text="#FFFFFF"
onLoad="window.moveTo(250, 250);">');
<%if (authorizedUsers != null){  /* ---- If Authorized Users is null ---
--- */%>
      d.writeln('<SCRIPT language="JavaScript">');
      d.writeln('function SendTransfer()');
      d.writeln('{');
      d.writeln('  if(TransferUser.value == ""){');
      d.writeln('      alert("Please Select the User to Transfer");');
      d.writeln('      return false;');
      d.writeln('}');
      d.writeln('   var itemTransferString = "' + transferString + '" +
"&userID=" + TransferUser.value;');
      //d.writeln('   window.open(itemTransferString,
"WorkListPage");');
      d.writeln('
window.opener.TransferForm.userID.value=TransferUser.value;');
      d.writeln('   window.opener.TransferForm.submit();');
      d.writeln('   window.close();');
      d.writeln('}');
      d.write('</SCRIPT');
      d.writeln('>');
      d.writeln('<CENTER><SELECT NAME="TransferUser" id="TransferUser"
SIZE="10">');
```

```
<%Iterator userIdList = authorizedUsers.keySet().iterator(); /* */
  while (userIdList.hasNext()) {
      Object thisUser = userIdList.next();%>

      d.writeln('<OPTION VALUE="<%=(String) thisUser%>"><%=(String)
thisUser %> - <%=authorizedUsers.get(thisUser)%>');
  <%}/* End While */%>
      d.writeln('</SELECT><BR><BR>');
      d.writeln('<a href="javascript:{}"
onClick="javascript:{SendTransfer()}"><img src="/MQWFClient-FMCWADMN/
images/action/transfer.gif" border="0" alt="Transfer Item"></
a>  ');
      d.writeln('<a href="javascript:{}"
onClick="javascript:{window.close();}"><img src="/MQWFClient-FMCWADMN/
images/action/cancel_checkout.gif" border="0" alt="Cancel"></a></
CENTER>');
<%}else{%>
      d.writeln('<P>There are no users to transfer to.');
<%}/* End If of Authorized Users is null */%>
      d.writeln('</BODY>');
      d.writeln('<HEAD>');
      d.writeln('<META HTTP-EQUIV="Pragma" CONTENT="no-cache"><META
HTTP-EQUIV="Expires" CONTENT="-1">');
      d.writeln('</HEAD>');
      d.writeln('</HTML>');
      d.close();
  }
  function truncateName(name)
  {
      document.write(name.substr(0,10));
  }


  function transferListItem(itemOid)
  {
      TransferForm.id.value = itemOid;
  }
```

TROUBLESHOOTING

The easiest way to implement the changes is to modify ListViewer.jsp and restart the Web client Web application, if you are using WebSphere. Taking a back-up of the original ListViewer.jsp is strongly recommended.

The modifications to ListViewer.jsp confuse normal users who are not used to the way the workflow Web client functions. So I

suggest you create an alias Web client configuration and modify the ListViewer.jsp under the alias configuration. The steps to create an alias Web client configuration FMCADMIN under WebSphere using FMCZUTIL utility are described below.

CREATING AN ALIAS WEB CLIENT CONFIGURATION

```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\Documents and Settings\db2admin>cd \

C:\>fmczutil
   FMC33201I Configuration Commands Menu:
     l ... List
     s ... Select
     c ... Create
     d ... Change default configuration
     x ... Exit Configuration Commands Menu
c
     Configuration identifier : [FMC] FMCWADMN

   FMC33210I Select Category Menu:
     s ... ( ) Server
     i ... ( ) Runtime Database Utilities
     b ... ( ) Buildtime
     c ... ( ) Client with queue manager
     j ... ( ) Java Agent
     w ... ( ) Web Client
     a ... all
     n ... none
     x ... Exit Select Category Menu
w
   FMC33210I Select Category Menu:
     s ... ( ) Server
     i ... ( ) Runtime Database Utilities
     b ... ( ) Buildtime
     c ... ( ) Client with queue manager
     j ... ( ) Java Agent
     w ... (X) Web Client
     a ... all
     n ... none
     x ... Exit Select Category Menu
j
   FMC33210I Select Category Menu:
     s ... ( ) Server
```

```
     i ... ( ) Runtime Database Utilities
     b ... ( ) Buildtime
     c ... (A) Client with queue manager
     j ... (X) Java Agent
     w ... (X) Web Client
     a ... all
     n ... none
     x ... Exit Select Category Menu
x


-  Configuration of queue manager ...

     System group name        : [FMCGRP1] FMCGRP
     System name              : [FMCSYS1] FMCSYS
     Queue manager name       : [FMCCONQM] FMCCONQM
     Queue prefix             : [FMC] FMC


-  Configuration of client ...

     Channel definition table file        : [d:\program files\mqseries
workflow
\chltabs\mqwfchl.tab]

-  Configuration of Java Agent ...

-  FMC33749I Selected Locator Policy : Local bindings

   FMC336Ø6I Specify information about garbage collection (reaper) ...:
     Agent cycle (in seconds)             : [3ØØ]
     Client threshold (number of objects) : [1ØØØ]
     Client cycle (in % of agent cycle)   : [9Ø]
-  Configuration of Web Client ...

   FMC33942I Specify the root URI of the Web Client :
     Root URI : [MQWFClient-FMCWADMN]

   FMC33777I Select application server ...:
     w ... ( ) WebSphere 3.x
     f ... (X) WebSphere 4.Ø (EAR)
     o ... ( ) Other
     j ... ( ) Other (Servlet 2.2 / J2EE 1.2)
w
     Code Version                         : [33Ø3]

FMC336Ø7I Specify information about the WebSphere Application Server..:
     Installation directory          : [d:\WebSphere\AppServer]
     TCP/IP address of administration node : [slater]
```

```
     TCP/IP address of name service host   : [slater]
     TCP/IP port number of name service    : [900]
     XML configuration skeleton file name  : [fmcoh354.skel]

     c ... Create configuration profile for 'FMCWADMN' now
     s ... Save input to file
     r ... Review/change input
     x ... Exit (input for configuration 'FMCWADMN' will be lost)
-  FMC3368ØI The profile for the configuration 'FMCWADMN' was updated
successfully.

-  Do you want to configure the Web Client within the WebSphere
Application Server now?
     y ... Yes
     n ... No
y
[Ø2.12.1Ø 17:41:14:5ØØ EST] ce421533 NodeConfig    A XMLC0053I:
Importing Node :
 slater
[Ø2.12.1Ø 17:41:14:671 EST] ce421533 ApplicationSe A XMLC0053I:
Importing Applic
ationServer : MQWF Web Client - FMCWADMN
[Ø2.12.1Ø 17:41:15:671 EST] ce421533 ServletEngine A XMLC0053I:
Importing Servle
tEngine : Servlet Container
[Ø2.12.1Ø 17:41:17:Ø62 EST] ce421533 WebApplicatio A XMLC0053I:
Importing WebApp
lication : MQWFClient-FMCWADMN
[Ø2.12.1Ø 17:41:17:562 EST] ce421533 ServletConfig A XMLC0053I:
Importing Servle
t : ErrorReporter
[Ø2.12.1Ø 17:41:17:875 EST] ce421533 ServletConfig A XMLC0053I:
Importing Servle
t : file
[Ø2.12.1Ø 17:41:18:281 EST] ce421533 ServletConfig A XMLC0053I:
Importing Servle
t : jsp11
[Ø2.12.1Ø 17:41:18:546 EST] ce421533 ServletConfig A XMLC0053I:
Importing Servle
t : Main
[Ø2.12.1Ø 17:41:18:89Ø EST] ce421533 SessionManage A XMLC0053I:
Importing Sessio
nManager : Session Manager
[Ø2.12.1Ø 17:41:19:281 EST] ce421533 ApplicationSe A XMLC0053I:
Importing Applic
ationServer : MQWF Web Client - FMCWADMN
[Ø2.12.1Ø 17:41:19:687 EST] ce421533 ContainerConf A XMLC0053I:
Importing Contai
ner : Default Container
   FMC3320ØI Configuration Commands Menu:
```

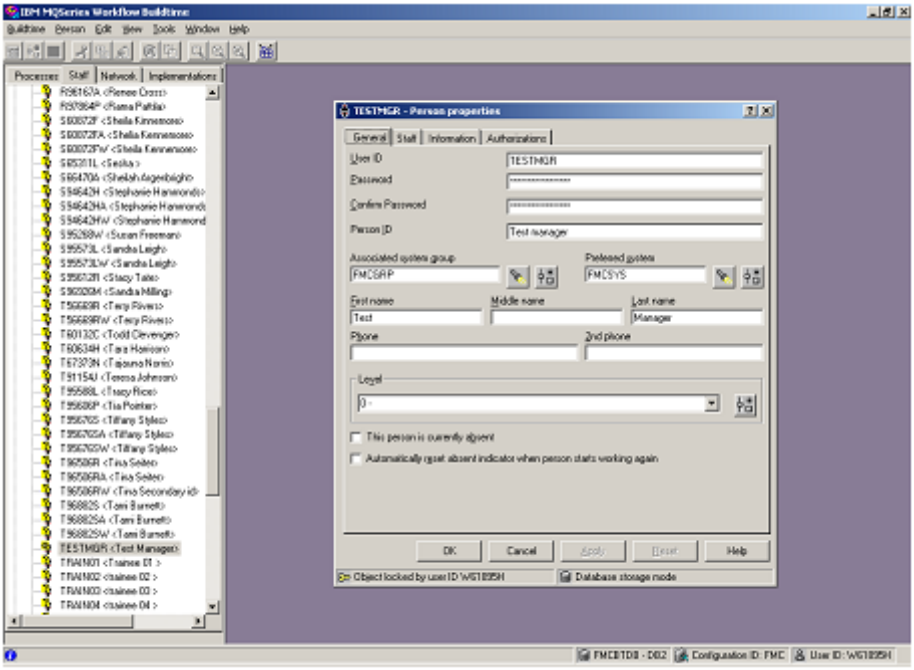

Figure 6: Buildtime person set up step 1

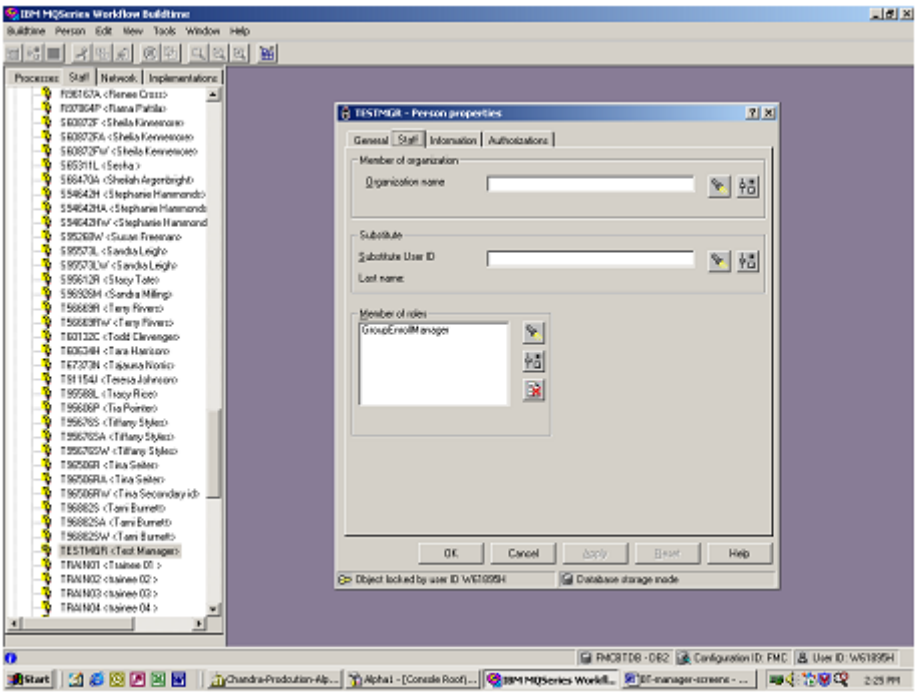

Figure 7: Buildtime person set up step 2

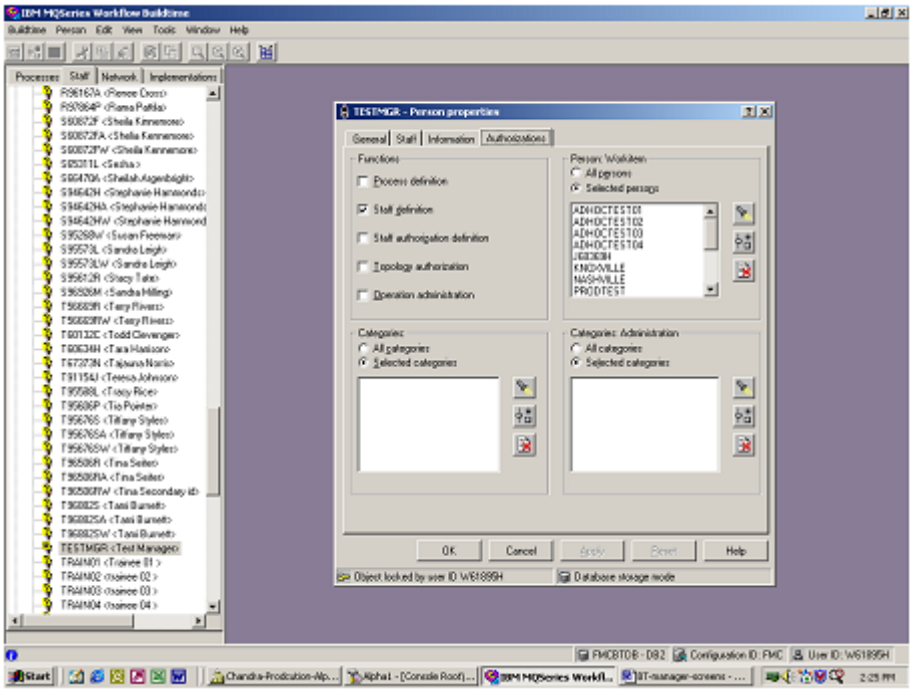

Figure 8: Buildtime person set up step 3

```
     l ... List
     s ... Select
     c ... Create
     d ... Change default configuration
     x ... Exit Configuration Commands Menu
x

C:\>
```

CAVEATS

1 The code is tested on the environment comprising IE, IIS, and WebSphere 3.5 with MQSeries Web Client V3.3.0.3.

2 The solution works for users of type TESTMGR who have authorization for a partial list of users. In the case of ADMIN (system administrator) type users who are authorized for ALL staff the solution does not give the list of ALL users. This is not the limitation of the solution but this is how the *personsAuthorizedFor* function in MQWorkflow works. Please see the workflow programming guide.

PersonsAuthorizedFor() returns the people for whom this person is authorized either explicitly or by being a substitute. If the person is authorized for all other people, then no person is returned here.

3 The user TESTMGR needs to have *AUTHORIZED_FOR STAFF* authority.

This authorization can be set at build time. When defining the person under build time see that the check box *Staff definition* under the *Authorizations* tab is ticked. Make sure the exported FDL contains *AUTHORIZED_FOR STAFF*.

SETTING AUTHORIZED_FOR STAFF AUTHORITY

To set *AUTHORIZED_FOR STAFF* authority for a person at Buildtime, follow the three steps illustrated in Figures 6, 7, and 8.

Please see the checkbox in Figure 8.

Chandra Upadhyayula
Programmer/Analyst (USA)

Customizing WebSphere MQ Integrator Broker message flows

WebSphere MQ Integrator Broker is a development platform and a run-time environment for message flows. Once the development of a message flow is done, the flow is usually transferred from the development environment to a production environment. In simple cases, just the message flow needs to be transferred or the flow is transferred for only one customer. In these situations only the flow needs to be adapted to the target environment. This can be done using the Control Center, the development tool for message flows.

In general, transferring the flow is not that simple because the message flow may be installed for multiple customers and the

message flow is usually referencing external resources, for example WebSphere MQ message queues or DB2 database tables. Together with the message flow, the resources that are referenced by the message flow must be created in the target environment. While this is done, the resources must be defined and their names adapted to the convention of the target system. Adapting the names of resources is necessary because customers usually have different naming conventions for test and production system or they use a different resource manager, for example different WebSphere MQ queue managers. These changes must be consistent with the changes in the message flows. This article describes how WebSphere Business Integration for Financial Networks (abbreviated to WebSphere BI for FN) solves this problem. (For details visit http://www.ibm.com/software/integration/wbifn/.)

CUSTOMIZATION

The process to adapt a message flow and the resources needed by the message flows to a target environment in WebSphere BI

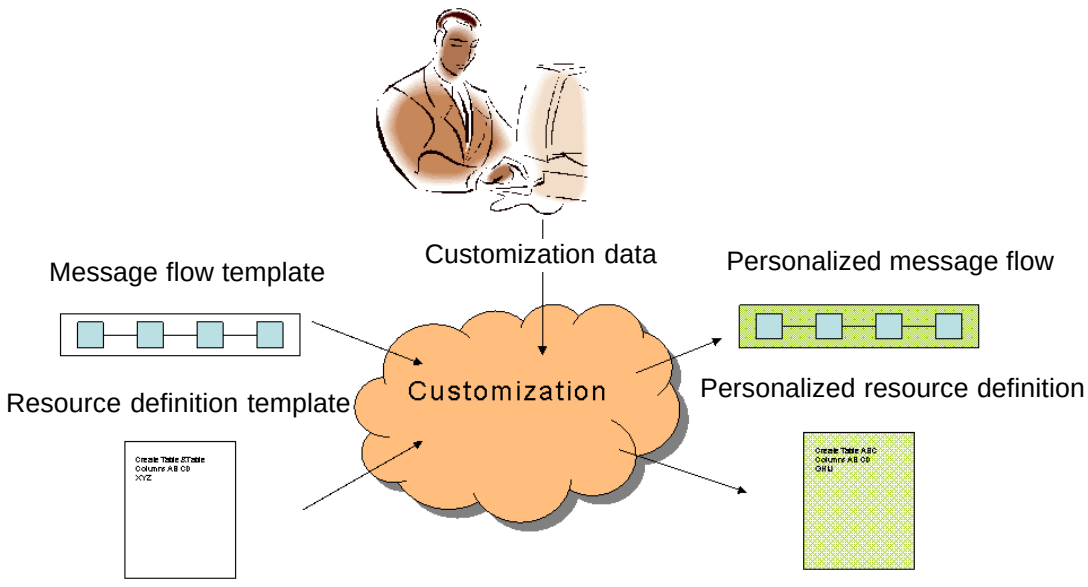

Figure 1: Customization overview

for FN is called customization. The overall structure of the process is shown in Figure 1.

Inputs to the customization process are message flow templates, resource definition templates, and customization data. A message flow template is a message flow that contains placeholders in the properties that need to be substituted. This is similar for the resource definition templates. These templates contain all the required definitions, for example the column names for a DB2 database table or required attributes of WebSphere MQ message queues. For attributes that need to be adapted to a target environment, like the name of a WebSphere MQ queue or the schema name for a DB table, the same placeholders as in the message flow are used.

The customization program is substituting the placeholders in the template files with actual values for the target environment. The values are defined by a WebSphere BI for FN customizer as customization data. This customization data describes the target environment. Substituting the placeholders with actual values in the message flow templates and resource definition templates is done by a customization program. The program generates message flows and resource definitions that are personalized for the target environment. These can then be deployed to the appropriate resource managers. The personalized message flows can be imported into the Control Center for the target broker domain, assigned to the target broker, and deployed.

The personalized resource definitions can be deployed to the resource managers to which the target broker is connected. How to handle the resource definitions depends on the type of resources that need to be defined. For example, definitions of WebSphere MQ queues can be processed with the program **runmqsc** on distributed platforms or **db2cmd** can be used for definitions of DB2 tables.

WebSphere BI for FN currently is processing the following kinds of resource definition file:

- WebSphere MQ resources.

- WebSphere MQ Integrator Broker message flows.
- DB2 resources.
- Security definitions, for example for RACF on z/OS or OAM on distributed platforms.
- WebSphere BI for FN configuration objects.

Since the customization program is doing simple text substitution, any other kind of resource definition could also be handled in the customization process, if needed.

After all the resources and the message flows are deployed, messages can be processed.

CUSTOMIZATION VARIABLES

The placeholders that need to be substituted in the template files are called customization variables. In principle any kind of information could be substituted in the message flow, but, if the variables are not carefully selected, the process can result in some overhead. Using the customization process, WebSphere BI for FN has restricted the customization variable to describe static properties of the run-time environment only. Static properties are variables for values of a target environment that are expected to stay constant for the lifetime of the message flow.

If other values are needed that could change often, it would be necessary to repeat the customization process. You would then, possibly, need to re-deploy all message flows and all resource definitions. This must be done carefully because if the resources already exist they may already contain data, for example messages in a WebSphere MQ message queue or entries in a DB2 table. This data would then be migrated. To prevent this, WebSphere BI for FN provides other means, so called configuration objects, which can be changed within the lifetime of the message flow.

Another restriction by WebSphere BI for FN for customization variables that can be substituted in message flows is that they

should not describe broker-specific values. If this were to be allowed, the result of the customization process would be different message flows for different brokers in a WebSphere MQ Integrator Broker domain. Since both would have the same template as origin, both message flows would have the same UUID. This would lead to the situation in which the second message flow would override the one that was imported first.

WebSphere BI for FN has chosen a very limited set consisting of the DB2 parameters data source name (DSN), the schema name for WebSphere BI for FN tables, and a high-level qualifier for the names of WebSphere MQ queue names. Also, there are only a few additional WebSphere BI for FN internal values, for example the name of a WebSphere BI for FN instance. Such an instance is a subset of the broker domain that is used by WebSphere BI for FN message flows.

The customization program is a text substitution program. Therefore the customization values could be anywhere in the message flow XML file, including in the ESQL for compute nodes. But WebSphere BI for FN is just using the customization variables in node properties. These are promoted to a high-level main flow. Just this main flow is processed through the customization process. This has the advantage that, for service purposes, sub-flows can be altered, imported into the Control Center, and deployed, without the need to process the main flow again through the customization process.

Most customization variables like a DSN or a queue name can be inserted directly in standard properties of WebSphere MQ Integrator Broker nodes. But for others, for example the schema name for DB2 tables, there are no such properties. To feed such values into the message flow processing, WebSphere BI for FN provides different plug-in nodes that can be used to receive the values and place them into the messages that are processed by the message flows.

Template files and template message flows are in different codepages. Message flow files are XML files in codepage 1208 (UTF-8), while other resource template files are usually in a text

format where the encoding depends on the platform, either ASCII or EBCDIC. The customization program is able to handle these differences in such a way that all resource template files, including the template message flows, are first converted to Unicode UCS-2 (codepage 1200) before the customization variables are replaced with actual values. After the replacements takes place, the customization program converts the files back to their original codepage to be able to process them with the appropriate import utilities.

CUSTOMIZATION DATA DEPLOYMENT

As described above, the personalized message flows and the personalized resource definitions can be deployed to the corresponding resource managers. In general the resource definitions describe only the mandatory parameters that are required for processing. For DB2 this is, for example, the name of the table columns. But such resources usually have a lot of other parameters that can be used to tune the application at run-time. Such parameters are for example DB2 buffer pools, WebSphere MQ page sets, or WebSphere MQ Integrator Broker message flow attributes like *Additional Instances*. Such parameters are tuning parameters that are usually adjusted by an administrator or the corresponding resource manager. That's why WebSphere BI for FN has decided that these kinds of parameters are best suited to being set by the administrator for the resource managers when the resources are defined. So before actually deploying the personalized resource definitions, the administrator should review the definitions and optionally set such tuning parameters. If at run-time an administrator decides that other values for the resources are better suited, the administrator can adapt these parameters without reprocessing the resource definition files or the message flow again through the customization process.

EXPLOITING CUSTOMIZATION

The customization as described in the previous sections is part

of the WebSphere BI for FN base product. The product uses the customization mechanism for its own resources. The base product is used as an infrastructure to deliver network-specific extension. Such extension can be developed by IBM, Independent Software Vendors (ISVs), or any customer. All WebSphere BI for FN extensions can exploit the customization program.

To participate in the customization process the message flows and resource definitions that make up a solution have to be collected after the flows are developed. Afterwards the values that need to be adapted are replaced with the customization variables. These can then be processed using the customization program when the extension should run at a customer site.

This kind of processing is especially useful if it is planned to use the message flows at multiple sites, for example when the message flows are delivered as part of a product or a solution.

SUMMARY

WebSphere BI for FN provides a powerful customization program that can be used to adapt message flows and the resources that are needed to run the message flow to a target environment. This program is used for product and solutions that need to be deployed to different target environments, either within an organization or if it is planned to sell a WebSphere MQ Integrator Broker application to multiple customers.

Michael Groetzner
IBM (Germany)

Sizing WebSphere MQ for z/OS CF structures

This article will explain a method used to calculate the sizes of the application Coupling Facility structures; it is valid for both MQ V5.2 with non-persistent messages and V5.3 with persistent messages on shared queues.

The various books available from IBM refer to a logarithmic chart one could use to estimate roughly the size of the structures, but I haven't found it very usable.

There is also a Web site that is dedicated to displaying the CF structure sizes given the size and number of messages – www-1.ibm.com/servers/eserver/zseries/cfsizer/mq.html. Unfortunately, it does not give the correct figures!

I have written to IBM and they acknowledge the fact that the calculations behind the scene have not been updated for some time. A PMR has been raised for this (January 2004).

The only source of useful information was the IBM SupportPac MP16 entitled *Capacity Planning and Tuning for WebSphere MQ for z/OS* and MP1D entitled *WebSphere MQ for zOS V5.3 and V5.3.1 Performance Report*, which gave this formula for calculating application structure sizes:

- Estimate the size of the message(s) including all headers (I believe excluding the MQMD).
- Round up to the 256 byte boundary (subject to a minimum size of 1,536).
- Multiply by the maximum number of messages.
- Add an overhead dependent on the CFCC level.

The following variables will therefore need to be obtained:

- A list of all the shared queues.
- For each queue the size of the messages.
- For each queue the highest queue depth reached.
- The CFCC level.

A LIST OF ALL THE SHARED QUEUES

A list of all the shared queues could be found by running the CSQUTIL utility with the following command:

```
DIS QL(*) QSGDISP(SHARED)
```

The only problem with this method is that the system might contain a number of shared queues that have been defined but are not, nor will ever be, actually used. So a more accurate method is to use the SMF statistics because only queues that have been used are listed.

Because SMF reports on all types of queues, the list will need to be filtered.

Warning: I discovered the following issue with the SMF data. I used the MXG product to report on all queues and summarized them on an hourly basis; in addition, I wanted to separate shared from non-shared queues. The reason for this is that the non-shared queues were being looked at from a storage class and pageset point of view, and as we know, shared queues' messages are stored in the CF.

In V5.2 a fix was applied (and carried forward into V5.3) by IBM to resolve alias queues into their base name (APAR PQ49374).

Unfortunately, in doing so, the fix did not copy the other details of the base queue like QSGDISP, CFSTRUCT, and STGCLASS. This means that if the report summarizes on queue name, it will appear twice – once as a shared queue and once as a non-shared queue. This, too, has been raised as a PMR (January 2004).

Obviously, you will need to run the SMF extract over a period of time in order to obtain a 'master' list of queues because some queues might be used one day and not the next, and *vice versa*.

SIZE OF MESSAGES

The size of messages can also be obtained from the SMF statistics. I would recommend that the queue definitions be changed to reflect the 63KB (64512 bytes) shared queue maximum message size rather than the default size of 4MB.

Also set the MAXDEPTH to the highest value it is likely to go to rather than some arbitrary value.

QUEUE DEPTH REACHED

Finding the queue depth reached is a bit trickier. Surprisingly, the SMF statistics do not hold this data, but I am hopeful that the next release of WebSphere MQ for z/OS will.

In the meantime, the only way to obtain queue depth is either via a vendor product (I did look at BMC's Mainview product but did not find the data) or via the following MQ command:

```
RESET QSTATS(*)
```

It's best to run this as part of CSQUTIL because it is then possible to store the output in a file for analysis; issuing the command at the console will only flood the system log.

Be warned, however, that the RESET command does zeroize the counters it displays.

Extract the output and format (via for example a REXX program) into a space or comma-delimited CSV file, eg store as Date, Time, Queue, Highest Queue Depth Reached.

You could also extract the MSGSIN and MSGSOUT fields (representing the MQPUTs and MQGETs in the period). This file needs to be matched with the list of shared queues.

In fact, if you are collecting a full day's worth of SMF data, and summarizing by the hour, then you could try to run the RESET command on an hourly basis as well. To reduce the amount of data, choose a typical day of the week and run it just on that day. What you're after is the highest queue depth reached.

CFCC LEVEL

The CFCC level is obtained via the following command:

```
D CF
```

CALCULATIONS

The easiest way is to use a spreadsheet and I have supplied one which can be found on the Xephon Web site, at www.xephon.com/extras/cfsizing.xls. The spreadsheet looks like this:

(A) Shared Queue Name	(B) Max Msg Size	(C) Calc Msg Size	(D) Highest Depth Reached	(E) CF o/h Factor	(F) Max Structure Size
Queue01	5434	5632	276	1.3	2020762
Queue02	5010	5120	346	1.3	2302976
Queue03	5025	5120	477	1.3	3174912
Queue04	5434	5632	2393	1.3	17520589
Queue05	5424	5632	1565	1.3	11458304
Queue06	9934	9984	471	1.3	6113203
Queue07	90	1536	15	1.3	29952
Queue08	10	1536	3	1.3	5990
Queue09	173	1536	2415	1.3	4822272
Queue10	173	1536	3332	1.3	6653338
Queue11	15	1536	3524	1.3	7036723
Queue12	600	1536	2507	1.3	5005978
Queue13	2000	2048	2135	1.3	5684224
Queue14	20	1536	493	1.3	984422
Queue15	20	1536	10621	1.3	21208013
Queue16	20	1536	10130	1.3	20227584
Queue17	20	1536	13812	1.3	27579802
Totals			54515		141829044

Column A is from the SMF statistics filtered for ‘shared’ queues only.

Column B is from the SMF statistics. Here only the maximum message size is considered.

Column C copies Column B and adjusts it to ensure that the minimum size is 1,536 and that it is divisible by 256.

Column D is from the RESET QSTATS output.

Column E represents the CF overhead, which for level 12 is 30%.

Column F is derived by multiplying columns C*D*E.

If you wanted to, you could do the above on an hourly basis, which will show how much CF storage is being used throughout the day.

DEFINING THE STRUCTURES

The z/OS systems programmer will be able to define the CFRM policy for each of the structures. Ensure that a reasonable ‘initial’

and ‘maximum’ size is provided and allow the structure to be automatically increased (ALLOWAUTOALT : YES).

In the above example set the SIZE to around 150MB, and INITSIZE to 75MB.

Although the operating system will increase the size of the structures when the threshold (FULLTHRESHOLD : 85) is reached, it is possible in a very busy system that the increase won’t happen fast enough. This will result in the application getting return code 2192.

Checking the meaning of this code (eg via the **mqrc** command) gives:

```
MQRC_PAGESET_FULL
MQRC_STORAGE_MEDIUM_FULL
```

and it’s the second description that would be applicable in this instance.

It may be found, however, that by the time the structure size is checked there is plenty of room in it; this is purely because of the time it takes to get the extra storage.

It may therefore be worthwhile setting the INITSIZE larger than the expected average size in order to avoid this situation.

In addition, if there is more than one Coupling Facility, with each one acting as a back-up for the other, ensure that the storage actually in use (check the maximum possible as well) does not exceed 50% of the available capacity (assuming the CFs are equally balanced). This is to cater for a failover scenario when one CF has to cope with the complete workload. Careful measurement is required because several of IBM’s other flagship products use the CF, including DB2, CICS, IMS, and now MQ.

MONITORING

There are several methods of monitoring available:

1 Manually by using the TSO panels. Use CFSTRUCT to list all structures and press PF11 to show the following example screen:

```
CF struct name Status Failure time Type   Size(KB) %Used Entries In use
*
STRUC1         ACTIVE              APPL    20224    74    7889     5914
CSQ_ADMIN      ACTIVE              ADMIN   10240     2    4307       75
SYSSTRUC       ACTIVE              APPL    10240     1    2025       34
STRUC2         ACTIVE              APPL   300032    99   86270    80608
TESTSTRUC1     ACTIVE              APP     13568     1    6321       35
                 ******** End of list ********
```

Note that the *In use* column is a measure of the number of messages on shared queues, but not an exact one because it depends on the size of the messages.

2 Using a vendor product.

3 SMF statistics.

4 Issuing command **/D XCF,STRUCTURE,STRNAME=xxxx** where *xxxx* is the structure name (ie QSG name plus structure name defined on the queue) and check the *ACTUAL SIZE* field. Compare it with the *SIZE* value.

5 Using the MQ command **DIS CFSTATUS(*) TYPE(SUMMARY)** to check the *SIZEMAX*, *ENTSMAX*, and *ENTSUSED* values.

6 Using an automation tool to pick up the following message:

```
IXC585E STRUCTURE QSG1STRUC2 IN COUPLING FACILITY A12CF, 641
PHYSICAL STRUCTURE VERSION BA9B11EA 1BAC5706,
IS AT OR ABOVE STRUCTURE FULL MONITORING THRESHOLD OF  85%.
```

This is followed by message IXC588I during structure increase and IXC590I when it has been completed.

CONCLUSION

This article has shown one method of calculating the CF application structures. It is, however, impossible to work it out accurately because it is dependent on the workload at any one time as well as message size.

As the available CF storage is likely to be less than the pagesets, I would recommend that a reasonable maximum queue depth be pre-assigned to each shared queue.

As queues are changed from local to shared queues, recalculate the CF structure sizes. In addition, when the CF level changes, take into account the increased system overhead.

Calculating a 'maximum size possible' gives an indication of whether the CF is able to hold that size in the case of system failures and peak periods. Be prepared!

Ruud van Zundert (ruudvz@btclick.com)
Independent Consultant (UK)

Got an idea for an article for *MQ Update*? Contact Trevor Eddolls at TrevorE@xephon.com to talk about it. Alternatively, read a copy of our *Notes for Contributors* at www.xephon.com/nfc. We are waiting to hear from you.

BMC Software has announced Version 3.2 of InTune, its MQ application tuning software.

InTune provides application-specific tuning information allowing users to improve the performance of applications in traditional and Parallel Sysplex mainframe environments. The solution allows users to view the performance metrics of their WebSphere MQ applications, enabling them to identify the source of application inefficiencies.

The new WebSphere MQ application support reports detailed performance statistics and activity associated with the WebSphere MQ queues and queue managers.

For further information contact:
BMC Software, 2101 City West Blvd, Houston, TX 77042-2827, USA.
Tel: (713) 918 8800.
URL: http://www.bmc.com/products/proddocview/0,2832,19052_19429_23398_1363,00.html.

* * *

Candle has announced two new PathWAI solutions that provide the first management support for all IBM WebSphere integration brokers, including the new WebSphere Business Integration Message Broker V5.0. PathWAI XE for WebSphere Integration Brokers V120 is a performance tuning and analytical tool that provides a high level of broker and message flow performance. Additionally, PathWAI Dashboard for WebSphere Business Integration offers a single integrated view of an entire IBM message broker and WebSphere MQ environment.

The new Candle offerings provide visibility into performance across an entire application or broker environment, enabling users to attain higher levels of process efficiencies and availability, say the company. WebSphere brokers function as communication hubs to transform and route data.

The WebSphere brokers identify new and updated events, translate the data into the appropriate format for each system and ensure that the messages are distributed immediately.

For further information contact:
Candle, 100 N Sepulveda Blvd, El Segundo, CA 90245, USA.
Tel: (310) 535 3600.
URL: http://www.candle.com/www1/cnd/portal/CNDportal_Channel_Master/0,2258,2683_2734,00.html.

* * *

Rogue Wave Software, a division of Quovadx, has announced Version 2.0 of its Lightweight Enterprise Integration Framework (LEIF). LEIF is a framework for creating service-based applications using new or existing C++ code.

LEIF eliminates barriers normally associated with transitioning to a Web services environment by allowing for complex messaging patterns and communication between business partners, including the use of WebSphere MQ connections between applications written in C/C++. This enhanced messaging capability allows developers to model virtually any communication pattern, including event notification and server-initiated messages.

For further information contact:
Rogue Wave, 5500 Flatiron Parkway, Boulder, CO 80301, USA.
Tel: (303) 473 9118.
URL: http://www.roguewave.com/products/leif.