48

MQ update

June 2003

In this issue

3 Writing a plug-in output node (WMQI V2.1)

9 WMQI: retrieving warehoused messages

25 Preserving message order

31 MQSeries message persistence

39 The Extended Transactional Client

45 July 2002 – June 2003 index

47 MQ news

© Xephon plc 2003

MQ Update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 38126
From USA: 01144 1635 38126
Fax: 01635 38345
E-mail: info@xephon.com

North American office

Xephon/QNA
Post Office Box 350100
Westminster CO 80035-0100, USA
Telephone: (303) 410 9344
Fax: (303) 438 0290

Disclaimer

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

Editor

Madeleine Hudson
E-mail: MadeleineH@xephon.com

Subscriptions and back-issues

A year's subscription to *MQ Update*, comprising twelve monthly issues, costs £255.00 in the UK; $380.00 in the USA and Canada; £261.00 in Europe; £267.00 in Australasia and Japan; and £265.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the July 1999 issue, are available separately to subscribers for £22.50 ($33.75) each including postage.

Contributions

When Xephon is given copyright, articles published in *MQ Update* are paid for at the rate of £170 ($260) per 1000 words and £100 ($160) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £50 ($80) per 100 lines. In addition, there is a flat fee of £30 ($50) per article. For more information about contributing an article you can 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 plc 2003. 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. Copying permits are available from Xephon in the form of pressure-sensitive labels, for application to individual copies. A pack of 240 labels costs $36 (£24), giving a cost per copy of 15 cents (10 pence). To order, contact Xephon at any of the addresses above.

Printed in England.

Writing a plug-in output node (WMQI V2.1)

INTRODUCTION

This article is aimed at C programmers who need to implement a WebSphere MQ Integrator (WMQI) C plug-in node that creates a message tree using all, some, or none of the root element children of the incoming message, and creates an output message tree that can be written as a bitstream for passing to a party external to the broker. The reader should be familiar with WMQI V2.1 and the C plug-in interface to make best use of this article.

The simplest course of action to take in a broker output node is to write the contents of the incoming message into a bitstream that can be passed to a party outside the broker. This is accomplished using the **cniWriteBuffer** function call:

```
int rc;
cniWriteBuffer(&rc, inMessage); /* inMessage is passed into the
  cniEvaluate function */
```

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

However, this is only useful for a message tree that does not need to be altered before being propagated or converted to a bitstream. If the message passed into the plug-in node needs to be altered before being converted the incoming message must first be copied to a new message, altered as necessary, and finalized before it can be propagated or converted to a bitstream, as shown here.

```
int rc;
CciMessageContext* inMessageContext = cniGetMessageContext(&rc,
inMessage);
CciMessage* outMessage = cniCreateMessage(&rc, inMessageContext);
CciElement* inRootElement = cniRootElement(&rc, inMessage);
CciElement* outRootElement = cniRootElement(&rc, outMessage);
cniCopyElementTree(&rc, inRootElement, outRootElement);
cniFinalize(&rc, outMessage);
cniPropagate(&rc, outMessage); /* OR */ cniWriteBuffer(&rc, outMessage);
```

© 2003. Reproduction prohibited. Please inform Xephon of any infringement.

Code block one

(Note: C compilers will normally predicate that all declarations are finished before functions are called.)

If the programmer decides that only part of the incoming message is to be represented in the output message or output bitstream two approaches can be taken:

- The whole message can be copied (as in the previous example) and the parts of the message tree not required in the output message can be deleted.
- A new message can be created and only the required children under the root element copied from the incoming message into the output message. Once this has been done, further element trees can be added into the output message before it is finalized and then propagated or converted to a bitstream.

Instead of copying the whole message tree only to then delete parts of the new message tree, the programmer may wish to copy the Properties and MQMD folders, for example, and create a BLOB part of the message tree, as shown here:

```
int rc;
struct CciByteArray bytes;
CciMessageContext* inMessageContext = cniGetMessageContext(&rc,
inMessage);
CciMessage* outMessage = cniCreateMessage(&rc, inMessageContext);
CciElement* inRootElement = cniRootElement(&rc, inMessage);
CciElement* outRootElement = cniRootElement(&rc, outMessage);
CciElement* firstChild =
    cniSearchFirstChild(&rc, inRootElement, CCI_COMPARE_MODE_NAME,
L"Properties", Ø);
CciElement* lastChild =
    cniCreateElementAsLastChildUsingParser(&rc, outRootElement,
L"MQPROPERTYPARSER");
cniSetElementName(&rc, lastChild, L"Properties");
cniSetElementType(&rc, lastChild, CCI_ELEMENT_TYPE_NAME);
cniCopyElementTree(&rc, firstChild, lastChild);
firstChild = cniSearchFirstChild(&rc, inRootElement,
CCI_COMPARE_MODE_NAME, Ø, L"MQMD");
lastChild = cniCreateElementAsLastChildUsingParser(&rc, outRootElement,
"MQHMD");
cniSetElementName(&rc, lastChild, L"MQMD");
```


© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.

```
cniSetElementType(&rc, lastChild, CCI_ELEMENT_TYPE_NAME);
cniCopyElementTree(&rc, firstChild, lastChild);
lastChild = cniCreateElementAsLastChildUsingParser(&rc, outRootElement,
"NONE");
cniSetElementName(&rc, lastChild, L"BLOB");
cniSetElementType(&rc, lastChild, CCI_ELEMENT_TYPE_NAME);
lastChild = cniCreateElementAsLastChild(&rc, lastChild);
cniSetElementName(&rc, lastChild, L"UnknownParserName");
cniSetElementType(&rc, lastChild, CCI_ELEMENT_TYPE_NAME_VALUE);
cniSetElementValue(&rc, lastChild, L"MQSTR", 5);
lastChild = cniCreateElementAsLastChild(&rc, lastChild);
cniSetElementName(&rc, lastChild, L"BLOB");
cniSetElementType(&rc, lastChild, CCI_ELEMENT_TYPE_NAME_VALUE);
bytes.pointer = (void*) { "Hello World!" };
bytes.size = strlen("Hello World!");
cniSetElementCharacterValue(&rc, lastChild, bytes.pointer);
cniSetElementCharacterLength(&rc, lastChild, bytes.size);
cniFinalize(&rc, outMessage);
cniPropagate(&rc, outMessage); /* OR */ cniWriteBuffer(&rc, outMessage);
```

Code block two

However, if the last child of the root element in the incoming message is associated with the MRM, XML, or user-defined parser, the previous example does not solve our problem. The programmer may decide at this stage again to copy the entire message tree and delete the children under root. However, if the incoming message tree does not contain an MQMD or contains message trees with user-defined parsers, either some special processing of the incoming message is necessary to detect the message trees (and their associated parsers) that are not required in the output message or a more generic method can be applied, as in the next example.

```
int rc;
CciChar parserName[256];
CciChar elementName[256];
CciMessageContext* inMessageContext = cniGetMessageContext(&rc,
inMessage);
CciMessage* outMessage = cniCreateMessage(&rc, inMessageContext);
CciElement* inRootElement = cniRootElement(&rc, inMessage);
CciElement* outRootElement = cniRootElement(&rc, outMessage);
CciElement* firstChild =
    cniSearchFirstChild(&rc, inRootElement, CCI_COMPARE_MODE_NAME,
L"Properties", 0);
CciElement* lastChild =
    cniCreateElementAsLastChildUsingParser(&rc, outRootElement,
```

© 2003. Reproduction prohibited. Please inform Xephon of any infringement.

```
L"MQPROPERTYPARSER");
cniSetElementName(&rc, lastChild, L"Properties");
cniSetElementType(&rc, lastChild, CCI_ELEMENT_TYPE_NAME);
cniCopyElementTree(&rc, firstChild, lastChild);
firstChild = cniSearchFirstChild(&rc, inRootElement,
CCI_COMPARE_MODE_NAME, Ø, L"MQMD");
lastChild = cniCreateElementAsLastChildUsingParser(&rc, outRootElement,
"MQHMD");
cniSetElementName(&rc, lastChild, L"MQMD");
cniSetElementType(&rc, lastChild, CCI_ELEMENT_TYPE_NAME);
cniCopyElementTree(&rc, firstChild, lastChild);
CciElement* inLastChild = cniLastChild(&rc, inRootElement);
cniGetParserName(&rc, inLastChild, parserName);
cniElementName(&rc, inLastChild, elementName);
CciElementType type = cniElementType(&rc, inLastChild);
CciElement* outLastChild = cniCreateElementAsLastChildUsingParser(&rc,
outRootElement, parserName);
cniSetElementName(&rc, outLastChild, elementName);
cniSetElementType(&rc, outLastChild, elementType);
cniCopyElementTree(&rc, inLastChild, outLastChild);
cniFinalize(&rc, outMessage);
cniPropagate(&rc, outMessage); /* OR */ cniWriteBuffer(&rc, outMessage);
```

Code block three

An alternative method (illustrated in *Code block four*) would be to iterate through a list of root element children names to copy from the incoming message root to the new output message. While this is possibly more complex it is nevertheless more flexible and provides a fluent implementation. The complexity arises when the programmer has the additional responsibility of creating the root element children in the output message with the correct parser, name (eg BLOB), and type (eg CCI_ELEMENT_TYPE_NAME), if any. This is because **cniCopyElementTree** only copies the children under the source element to the target element. The target element must, therefore, be created correctly if that part of the message tree is going to be parsed correctly when it is propagated or converted to a bitstream. The additional effort of code development is necessary when the programmer needs to avoid taking the easy (but inefficient) option of copying the entire message only to then delete root element children not required in the output message.

The previous example (*Code block three*) can be rewritten to be

© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.

completely generic (without hard-coding for Properties or MQMD) by providing a list of root element children and their parsers, as shown in the next example. For a real implementation all calls should be checked for bad return codes.

```
CciMessageContext* inMessageContext = cniGetMessageContext(&rc,
inMessage);
CciMessage* outMessage = cniCreateMessage(&rc, inMessageContext);
CciElement* inRootElement = cniRootElement(&rc, inMessage);
CciElement* outRootElement = cniRootElement(&rc, outMessage);
CciChar* list[4] = { L"Properties", L"0", L"MQMD", L"MQHMD" };
unsigned int i;
for (i = 0; i < 4; i += 2) {
  copyChild(inRootElement, outRootElement, list[i*2], list[i*2+1]);
}
copyLastChildUsingParser(inRootElement, outRootElement);
cniFinalize(&rc, outMessage);
cniPropagate(&rc, outMessage); /* OR */ cniWriteBuffer(&rc, outMessage);
void copyChild(
  CciElement* inRootElement,
  CciElement* outRootElement,
  CciChar* elementName,
  CciChar* parserName;
)
{
  int rc;
  CciElement* firstChild = cniSearchFirstChild(
    &rc, inRootElement, CCI_COMPARE_MODE_NAME, elementName, 0
  );
  if ((CciCharCmp(parserName, L"0") == 0) {
    CciElement* lastChild = cniCreateElementAsLastChild(&rc,
outRootElement);
    cniSetElementName(&rc, lastChild, elementName);
    cniSetElementType(&rc, lastChild, CCI_ELEMENT_TYPE_NAME);
    cniCopyElementTree(&rc, firstChild, lastChild);
  } else {
    cniElementName(&rc, firstChild, elementName);
    CciElementType type = cniElementType(&rc, firstChild);
    CciElement* lastChild = cniCreateElementAsLastChildUsingParser(&rc,
outRootElement, parserName);
    cniSetElementName(&rc, lastChild, elementName);
    cniSetElementType(&rc, lastChild, elementType);
    cniCopyElementTree(&rc, firstChild, lastChild);
  }
}
void copyLastChildUsingParser(
  CciElement* inRootElement,
  CciElement* outRootElement,
)
```

```
{
  int rc;
  CciElement* inLastChild = cniLastChild(&rc, inRootElement);
  CciChar parserName[256];
  cniParserName(&rc, inLastChild, parserName);
  CciElement* outLastChild = cniCreateElementAsLastChildUsingParser(&rc,
outRootElement, parserName);
  CciChar inElementName[256];
  cniElementName(&rc, inLastChild, inElementName);
  CciElementType inElementType;
  inElementType = cniElementType(&rc, inLastChild);
  cniSetElementName(&rc, outLastChild, inElementName);
  cniSetElementType(&rc, outLastChild, inElementType);
  cniCopyElementTree(&rc, inLastChild, outLastChild);
}
```

Code block four

(Note: if the last child of root is known to be BLOB, for example, the values L"BLOB", L"0" can be added to the CciChar* list, the index '4' changed to '6' and then the call to **copyLastChildUsingParser** can be omitted.)

SUMMARY

The important points to note from this article are:

- An incoming message cannot be altered in any way.
- All new messages must be finalized (**cniFinalize**) before issuing a call to **cniPropagate** or **cniWriteBuffer**.
- In other words the broker does not under any circumstances finalize a message for you.

Note that, with reference to the third point above, it is useful to know that, while the message buffer can be accessed at any time for an incoming message, the message buffer for a new message is not accessible until **cniFinalize** has been called. If this fails (CCI_EXCEPTION == rc, for example) and the program does not check for this return code a subsequent call to **cniWriteBuffer** may not return a bad return code itself but the buffer pointer will certainly not be valid. In the author's experience the buffer pointer is often null and the buffer size is often zero.

© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.

This article has explained how to copy only the required root element children from an incoming message into a new output message, and the steps required to construct correctly the new message for propagating or converting to a bitstream. This is a general method for copying only the parts of the incoming message required in the output message.

Alexander Russell, M Eng, IBM Certified. With thanks to John Hosie (IBM).
IBM Hursley (UK) © IBM 2003

WMQI: retrieving warehoused messages

INTRODUCTION

WMQI provides a warehouse node to save whole messages, including header information, as BLOB into a relational table. Saving messages into the table with a warehouse node is easy; retrieving them is a little more complicated.

Since most people don't know how to retrieve the whole message from the warehouse table they usually take the shortcut of using a database node, storing just the message body or part of the message body with the message-ID as the key for retrieval. This requires extra work, defining the table with a schema that matches the content of the message to be saved.

With the warehouse node you don't need to predefine a schema.

The biggest reward gained from using the warehouse node to store messages is the consistency of the message header, which includes the message context information. This may be a particularly significant requirement for organizations with strict auditing criteria.

The warehouse node provides a useful way of 'parking' messages so that if a failure should occur, for example, the message can

© 2003. Reproduction prohibited. Please inform Xephon of any infringement.

be reprocessed in its original message context when the system is running again.

WAREHOUSE/RETRIEVE SCENARIO

There are various reasons why users might like to warehouse or park messages for subsequent reprocessing. In my particular situation I have two messages; the business logic is written in such a way that the second message depends on the successful processing of the first message.

- The first message from the front end arrives at the hub. The hub will add the front-end system key to the cross-reference table.
- The hub forwards the message to the back end.
- Once the back end has finished processing the message it sends a message to the hub to add the back-end key to the cross-reference table.
- A UUID on the cross-reference table links together the keys of the two systems.
- The second message from the front end arrives at the hub at a later time. The front-end key has to be cross-referenced to that of the back end or the destination system key by the hub before the hub forwards the message to the destination system.

Normally the time it takes for the front-end user to gather information and submit the second message is long enough for the first message to complete the trip of adding two system keys to the hub. Yet there may still be occurrences (perhaps because of network congestion) when the first message does not make it back to the hub in time with the new key and so the second message fails because the cross-referencing is not found. We know that the first message will come back eventually though, with the new key, so instead of 'error out' the message is warehoused. Once the message from the back end arrives at the hub it updates the cross-reference table; it also releases and

© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.

reprocesses the warehoused messages so this time the cross-reference will be successful.

This method saves a lot of human intervention in resubmitting messages.

SAVING TO THE WAREHOUSE TABLE

In simple terms, when the second message arrives at the hub the message flow will use the front-end (source) system key and the front-end (source) system-ID with the destination (target) system-ID to look up the cross-reference table for the system key of the destination (target) system, which is tied by the UUID. If the target system key is not found the message will be warehoused, using the UUID, the target system-ID, and also the put time, as a composite key to the table.

THE WAREHOUSE TABLE

The warehouse table HUB_WH was defined as follows:

```
-- DDL Statements for table "ALEXAU  "."HUB_WH"
  CREATE TABLE "ALEXAU  "."HUB_WH"  (
        "WH_UUID" INTEGER NOT NULL ,
        "WH_TARG_SYSID" CHAR(3) NOT NULL ,
        "WH_TS" TIMESTAMP NOT NULL ,
        "WH_BLOB" BLOB(32000) NOT LOGGED NOT COMPACT )
          IN "USERSPACE1" ;
```

WAREHOUSE THE MESSAGE

Figure 1 shows a sample message flow (no cross-reference logic), which simulates a cross-reference failure; the message will be warehoused with the UUID, target system-ID, and the put timestamp.

The Compute1 node consists of ESQL that simulates, in case of a cross-reference failure, the information required to warehouse the message.

```
-- SET the failing UUID and SYSID, this should be done from the
  cross-reference in the environment
```

© 2003. Reproduction prohibited. Please inform Xephon of any infringement.

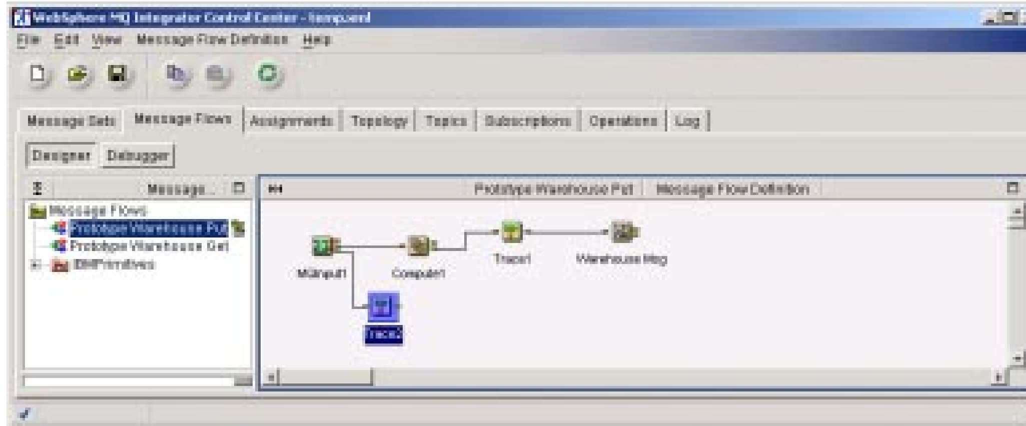

Figure 1: Saving the message with the warehouse node

```
SET Environment.Variables.WH.UUID =
  CAST(InputBody.Message.CrfActionGroup.KeyGroup.UUID AS INTEGER);
SET Environment.Variables.WH.TARG_SYSID =
  InputBody.Message.CrfActionGroup.(XML.attr)destinationLogicalId;
```

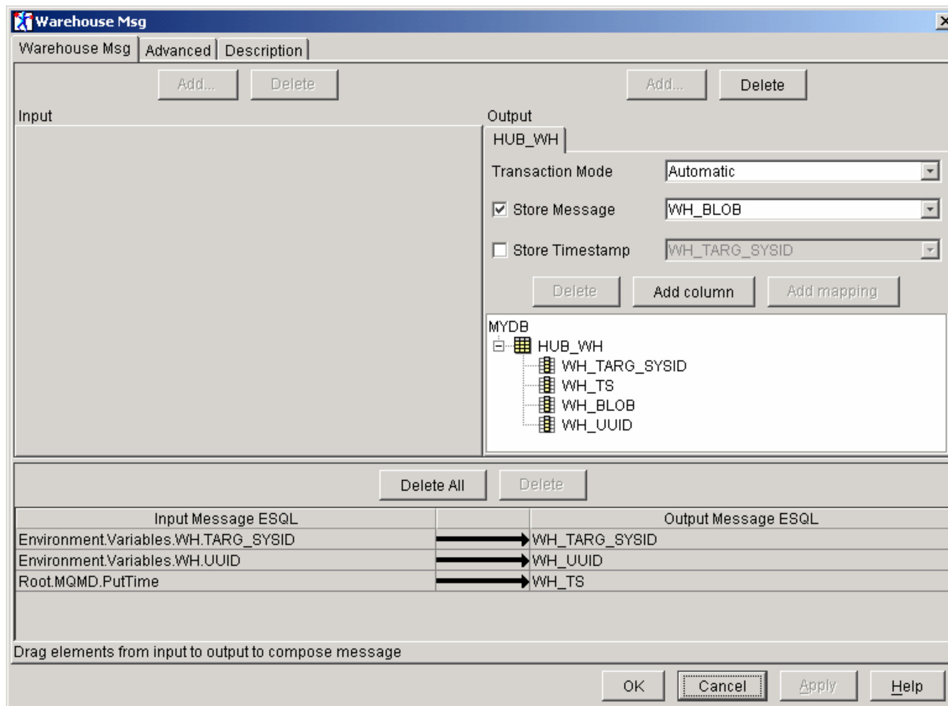

Figure 2: The warehouse node

© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.

The warehouse message is illustrated in Figure 2.

A SAMPLE FAILED MESSAGE

I use the *RFHUtil* utility to put a message to the message flow to which I can add an MQHRF2 header with the *Mcd* and *usr* folders populated with information.

Below is the output of the trace1 node.

```
*** trace generate after the compute node @ 2003-02-07 14:49:11.291
*** Environment is
(
  (0x1000000)Variables = (
    (0x1000000)WH = (
      (0x3000000)UUID       = 1
      (0x3000000)TARG_SYSID = 'AAU'
    )
  )
)
*** Root is
(
  (0x1000000)Properties = (
    (0x3000000)MessageSet      = 'Alex Au'
    (0x3000000)MessageType     = ''
    (0x3000000)MessageFormat   = 'XML'
    (0x3000000)Encoding        = 546
    (0x3000000)CodedCharSetId  = 1208
    (0x3000000)Transactional   = TRUE
    (0x3000000)Persistence     = TRUE
    (0x3000000)CreationTime    = GMTTIMESTAMP '2003-02-07 19:49:10.760'
    (0x3000000)ExpirationTime  = -1
    (0x3000000)Priority        = 0
    (0x3000000)ReplyIdentifier =
X'000000000000000000000000000000000000000000000000'
    (0x3000000)ReplyProtocol   = 'MQ'
    (0x3000000)Topic           = NULL
  )
  (0x1000000)MQMD       = (
    (0x3000000)SourceQueue      = 'ALEX_TEST_IN'
    (0x3000000)Transactional    = TRUE
    (0x3000000)Encoding         = 546
    (0x3000000)CodedCharSetId   = 1208
    (0x3000000)Format           = 'MQHRF2  '
    (0x3000000)Version          = 2
    (0x3000000)Report           = 0
    (0x3000000)MsgType          = 8
    (0x3000000)Expiry           = -1
```

© 2003. Reproduction prohibited. Please inform Xephon of any infringement.

```
    (0x3000000)Feedback          = 0
    (0x3000000)Priority          = 0
    (0x3000000)Persistence       = 1
    (0x3000000)MsgId             =
X'414d51204d5153492020202020202020201908443e12c00000'
    (0x3000000)CorrelId          =
X'000000000000000000000000000000000000000000000000'
    (0x3000000)BackoutCount      = 0
    (0x3000000)ReplyToQ          = 'ALEX_TEST_OUT                               '
    (0x3000000)ReplyToQMgr       = 'MQSI                                        '
    (0x3000000)UserIdentifier    = 'alexau      '
    (0x3000000)AccountingToken   =
X'1601051500000495e9d7af700ba481f11393fef030000000000000000000000b'
    (0x3000000)ApplIdentityData = '                                '
    (0x3000000)PutApplType       = 11
    (0x3000000)PutApplName       = 'QSI IH03\IH03 v3\rfhutil.exe'
    (0x3000000)PutDate           = DATE '2003-02-07'
    (0x3000000)PutTime           = GMTTIME '19:49:10.760'
    (0x3000000)ApplOriginData    = '    '
    (0x3000000)GroupId           =
X'000000000000000000000000000000000000000000000000'
    (0x3000000)MsgSeqNumber      = 1
    (0x3000000)Offset            = 0
    (0x3000000)MsgFlags          = 0
    (0x3000000)OriginalLength    = -1
  )
  (0x1000000)MQRFH2      = (
    (0x3000000)Version         = 2
    (0x3000000)Format          = 'MQSTR   '
    (0x3000000)Encoding        = 546
    (0x3000000)CodedCharSetId = 1208
    (0x3000000)Flags           = 0
    (0x3000000)NameValueCCSID = 1208
    (0x1000000)mcd             = (
      (0x1000000)Msd = (
        (0x2000000) = 'XML'
      )
      (0x1000000)Set = (
        (0x2000000) = 'Alex Au'
      )
      (0x1000000)Fmt = (
        (0x2000000) = 'XML'
      )
    )
    (0x1000000)usr             = (
      (0x1000000)ReplyInfo = (
        (0x2000000)             = '
'
        (0x1000000)ReplyToQ     = (
          (0x2000000) = 'ALEX_TEST_OUT1'
```


© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.

```
          )
          (0x1000000)ReplyToQMgr = (
            (0x2000000) = 'MQSI'
          )
        )
      )
    )
    (0x1000010)XML        = (
      (0x5000018)XML     = (
        (0x6000011) = '1.0'
      )
      (0x6000002)         = '
'
      (0x1000000)Message = (
        (0x3000000)id                       = 'SIEBEL01A2001-10-
31T11:00:00:000000'
        (0x3000000)version                  = '1.4'
        (0x3000000)bodyType                 = 'IAA-XML'
        (0x3000000)timeStampCreated         = '2001-10-31T11:00:00:000000'
        (0x3000000)sourceLogicalId          = 'SBL'
        (0x3000000)destinationLogicalId     = 'HUB'
        (0x3000000)authenticationId         = 'PADM039'
        (0x3000000)crfPublish               = 'true'
        (0x2000000)                         = '
   '
        (0x1000000)CrfActionGroup           = (
          (0x3000000)crfPublish               = 'true'
          (0x3000000)destinationLogicalId     = 'AAU'
          (0x2000000)                         = '
      '
          (0x1000000)CommandReference         = (
            (0x3000000)refid = 'cmd1'
          )
          (0x2000000)                         = '
      '
          (0x1000000)KeyGroup                 = (
            (0x3000000)id             = 'K2'
            (0x3000000)keyGroupType = 'PARTY'
            (0x2000000)               = '
         '
            (0x1000000)AlternateId   = (
              (0x3000000)value              = 'A23456789'
              (0x3000000)sourceLogicalId = 'AAU'
              (0x3000000)state              = 'exists'
            )
            (0x2000000)               = '
         '
            (0x1000000)UUID           = (
              (0x2000000) = '1'
            )
```

© 2003. Reproduction prohibited. Please inform Xephon of any infringement.

```
        (Ø×2ØØØØØØ)             = '
      '
      )
      (Ø×2ØØØØØØ)                     = '
    '
    )
    (Ø×2ØØØØØØ)                     = '
    '
    (Ø×1ØØØØØØ)COMMAND              = (
      (Ø×2ØØØØØØ)                    = '
      '
      (Ø×1ØØØØØØ)AddHouseholdRequest = (
        (Ø×3ØØØØØØ)id         = 'CMD1'
        (Ø×3ØØØØØØ)cmdType    = 'request'
        (Ø×3ØØØØØØ)cmdMode    = 'alwaysRespond'
        (Ø×3ØØØØØØ)echoBack   = 'false'
        (Ø×2ØØØØØØ)           = '
        '
        (Ø×1ØØØØØØ)Household = (
          (Ø×2ØØØØØØ)         = '
          '
          (Ø×1ØØØØØØ)KeyGroup = (
            (Ø×3ØØØØØØ)refid = 'K1'
          )
          (Ø×2ØØØØØØ)         = '
        '
        )
        (Ø×2ØØØØØØ)           = '
        '
        (Ø×1ØØØØØØ)Party      = (
          (Ø×2ØØØØØØ)         = '
          '
          (Ø×1ØØØØØØ)KeyGroup = (
            (Ø×3ØØØØØØ)refid = 'K2'
          )
          (Ø×2ØØØØØØ)         = '
        '
        )
        (Ø×2ØØØØØØ)           = '
      '
      )
      (Ø×2ØØØØØØ)                    = '
    '
    )
    (Ø×2ØØØØØØ)                     = '
'
  )
 )
)
```


© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.

I also put a second message with just the MQMD header, not the MQHRF2 header, to the warehouse table.

THE TABLE CONTENT

Figure 3 shows the screen shot that results from querying the

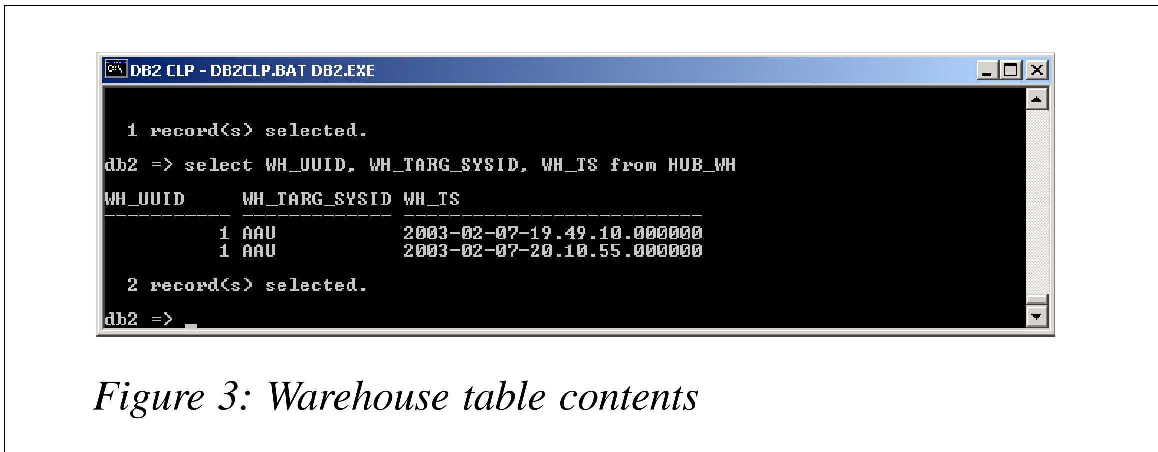

Figure 3: Warehouse table contents

contents of the warehouse table after two messages have been warehoused. Note, I do not query on the BLOB field since it will be too big to fit inside the window.

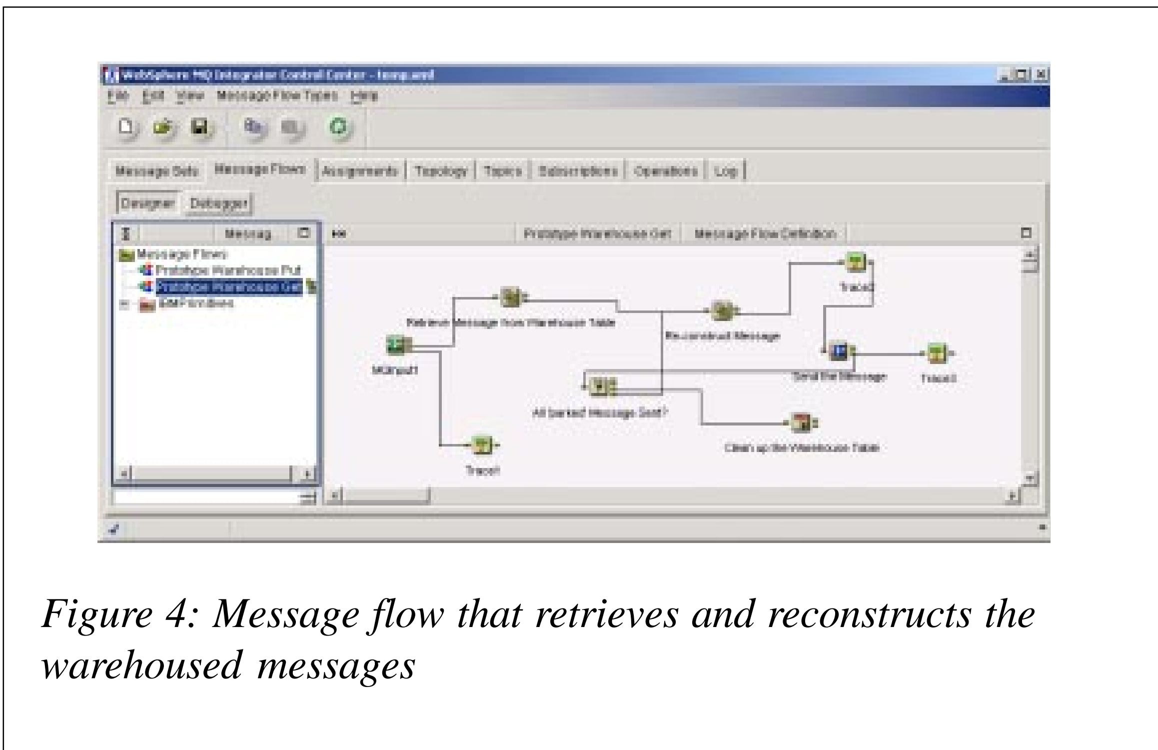

Figure 4: Message flow that retrieves and reconstructs the warehoused messages

© 2003. Reproduction prohibited. Please inform Xephon of any infringement.

FLOW TO RETRIEVE MESSAGES FROM THE WAREHOUSE TABLE

The message flow to retrieve the message from the warehouse is shown in Figure 4.

In this message flow I simulate a message from the back end that will add to the cross-reference table with its new key, according to the UUID received earlier. With this UUID and target system-ID the message flow will query the warehouse table and retrieve all the messages parked according to these two entries.

SAMPLE OF THE RELEASE MESSAGE

The release message in my simulated case is as simple as that shown below.

```
<Message>
<UUID>1</UUID>
<TID>AAU</TID>
</Message>
```

RETRIEVE MESSAGE FROM WAREHOUSE TABLE

The ESQL for the 'Retrieve Message from Warehouse Table' instruction is shown below. You can tailor the select statement to suit your own requirements. The messages are retrieved in ascending order so that once put to the input queue they will be reprocessed according to their chronological sequence.

```
SET OutputRoot = InputRoot;
-- Enter SQL below this line.  SQL above this line might be regenerated,
causing any modifications to be lost.
DECLARE selectStatment CHAR;
SET selectStatment = 'SELECT D.* FROM ALEXAU.HUB_WH AS D WHERE D.WH_UUID
= ? AND D.WH_TARG_SYSID = ? ORDER BY WH_TS ASC';
SET Environment.Variables.WH.Results[] = PASSTHRU(selectStatment,
InputBody.Message.UUID, InputBody.Message.TID);
SET Environment.Variables.Select.WH.SQLState1 = SQLSTATE;
SET Environment.Variables.Select.WH.SQLErrorText1 = SQLERRORTEXT;
SET Environment.Variables.Select.WH.SQLCode1 = SQLCODE;
SET Environment.Variables.Select.WH.SQLNativeError1 = SQLNATIVEERROR;
SET Environment.Variables.WH.maxCnt =
CARDINALITY(Environment.Variables.WH.Results[]);
SET Environment.Variables.WH.cnt = 1;
```


© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.

RECONSTRUCT MESSAGE

This is the key logic in the flow to reconstruct the parked message. I do not check the box to Copy message header nor Copy entire message because we want to reconstruct the original message from the table read into the environment variables.

```
-- Enter SQL below this line.  SQL above this line might be regenerated,
causing any modifications to be lost.
DECLARE msgCnt INT;
DECLARE beginMsg INT;
DECLARE MQHRF2HeaderLength INT;
DECLARE bitHolder BLOB;
DECLARE BigEn BLOB;
DECLARE TempBLOB BLOB;
SET bitHolder = X'00000000';
SET msgCnt = CAST(Environment.Variables.WH.cnt AS INT);
-- Set up the new message
-- note: we do not copy the message header, we want to restore the
message header stored
SET OutputRoot.Properties.MessageDomain = 'XML' ;
-- Clean up for multiple messages written out
SET Environment.Variables.MQMDOut = NULL;
SET Environment.Variables.MQRFH2Out = NULL;
SET Environment.Variables.XMLOut = NULL;
-- Get the MQMD portion, a fix 364 bytes long
SET TempBLOB =
SUBSTRING(Environment.Variables.WH.Results[msgCnt].WH_BLOB FROM 1 FOR
364);
SET Environment.Variables.TempBLOB1 = TempBLOB;
-- use the MQMD parser to format these fields into the a new field in
the environment tree
CREATE LASTCHILD OF Environment.Variables.MQMDOut DOMAIN('MQMD')
PARSE(TempBLOB, 546, 437);
-- In order for the above restored message attributes to be effective,
including all the context information, the subsequent MQ output
-- node will have select 'Set All' on the message context box on the
advanced tab
SET beginMsg = 365;
-- check for MQHRF2 header presence, if yes, get the length of the
header, add to get the new beginMsg
SET MQHRF2HeaderLength = 0;
-- Byte 32 onward is the MQMD_FORMAT field, this example only consider
the MQHRF2 header, not other
IF CAST(SUBSTRING(Environment.Variables.WH.Results[msgCnt].WH_BLOB FROM
33 FOR 6) AS CHAR CCSID 437) = 'MQHRF2' THEN
     -- Set up the MQHRF2 header
     -- Total length of the MQHRf2 header is from the 8th bytes after
```

© 2003. Reproduction prohibited. Please inform Xephon of any infringement.

```
the beginning of the MQHRF2 header, a 4 to 8 bytes fields
     -- here we only consider 4 bytes length, our message now is not
that long
     -- because this is a little endian deal, need to set the most
significant bytes to the least
     SET BigEn =
SUBSTRING(Environment.Variables.WH.Results[msgCnt].WH_BLOB FROM 373 FOR
4) || bitHolder;
     SET MQHRF2HeaderLength = CAST(ConvertEndian(BigEn) AS INT CCSID
437);
     -- with this length, we can parse the MQHRF2 header to the
environment tree
     SET TempBLOB =
SUBSTRING(Environment.Variables.WH.Results[msgCnt].WH_BLOB FROM 365 FOR
MQHRF2HeaderLength);
     CREATE LASTCHILD OF Environment.Variables.MQRFH2Out
DOMAIN('MQRFH2') PARSE(TempBLOB, 546, 437);
END IF;
-- so message body begins either from 365 if no MQHRF2 header, or begin
from where MQHRF2 header ends
SET beginMsg = beginMsg + MQHRF2HeaderLength;
SET TempBLOB =
SUBSTRING(Environment.Variables.WH.Results[msgCnt].WH_BLOB FROM
beginMsg) ;
CREATE LASTCHILD OF Environment.Variables.XMLOut DOMAIN('XML')
PARSE(TempBLOB, 546, 437);
-- Putting them back together
SET OutputRoot.MQMD = Environment.Variables.MQMDOut.MQMD;
SET OutputRoot.MQRFH2 = Environment.Variables.MQRFH2Out.MQRFH2;
SET OutputRoot.XML = Environment.Variables.XMLOut.XML;
-- Increment for next message
SET Environment.Variables.WH.cnt = Environment.Variables.WH.cnt + 1;
-- Function to convert the big endian
CREATE FUNCTION ConvertEndian (Value BLOB)
RETURNS BLOB BEGIN
     DECLARE LittleEn BLOB;
     DECLARE FieldLength INT;
     SET FieldLength = LENGTH(Value);
     SET LittleEn = substring(Value from FieldLength for 1);
     SET FieldLength = FieldLength - 1;
     WHILE FieldLength >= 1  DO
          SET LittleEn = LittleEn || substring(Value from FieldLength
for 1);
          SET FieldLength = FieldLength - 1;
     END WHILE;
     SET Value = LittleEn;
RETURN Value;
END;
```


© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.

SEND THE MESSAGE

This is where you will put the reconstructed message. I put it into a local queue so that I can examine the header information.

One important point to make clear is that in the Advanced tab of the output node you should be sure to select 'Set All' for the message context as Figure 5 illustrates, otherwise the message context of the original message will be overridden by the message flow.

Trace3

Trace3 shows the actual message put into the input queue of the message flow that failed at the first attempt. Check the trace to ensure that the content of the MQ header, MQHRF2 header, and the message body are the same as in the original message.

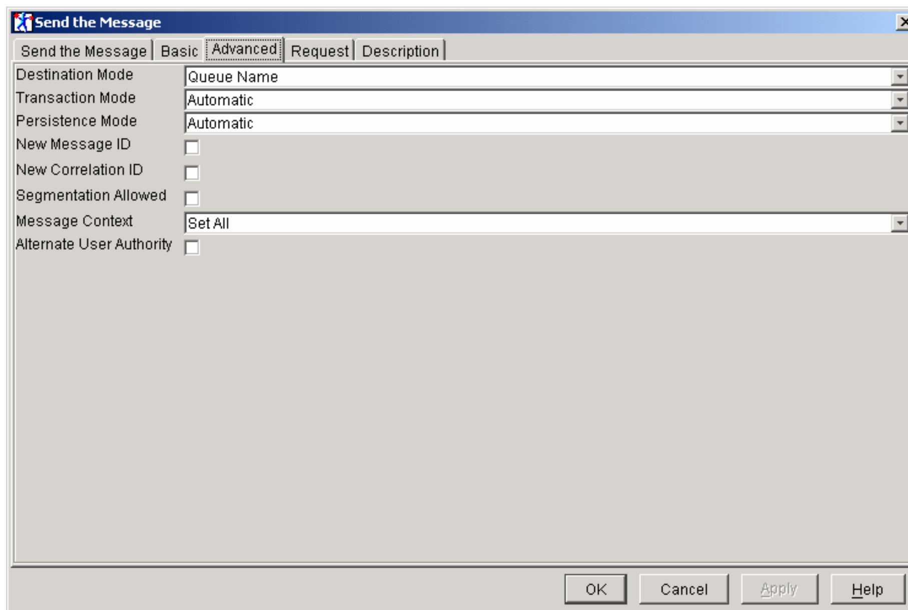

Figure 5: Output node

© 2003. Reproduction prohibited. Please inform Xephon of any infringement.

ALL 'PARKED' MESSAGES SENT?

This filter node controls the loop until all messages read from the table that satisfied the selection criteria have been sent. The ESQL statement is:

```
CAST(Environment.Variables.WH.cnt AS INT)  <=
 CAST(Environment.Variables.WH.maxCnt AS INT)
```

EXAMINING THE OUTPUT

Once the messages' BLOB are read from the database they are stored in *Environment.Variables.WH.Results* for parsing (see Figure 6).

The BLOB was cut into two or three pieces, depending on the presence of the MQHRF2 header. Each BLOB piece was then parsed according to its DOMAIN by the CREATE statement.

The MQMD portion

```
-- Get the MQMD portion, a fix 364 bytes long
SET TempBLOB =
  SUBSTRING(Environment.Variables.WH.Results[msgCnt].WH_BLOB FROM 1
  FOR 364);
SET Environment.Variables.TempBLOB1 = TempBLOB;
```

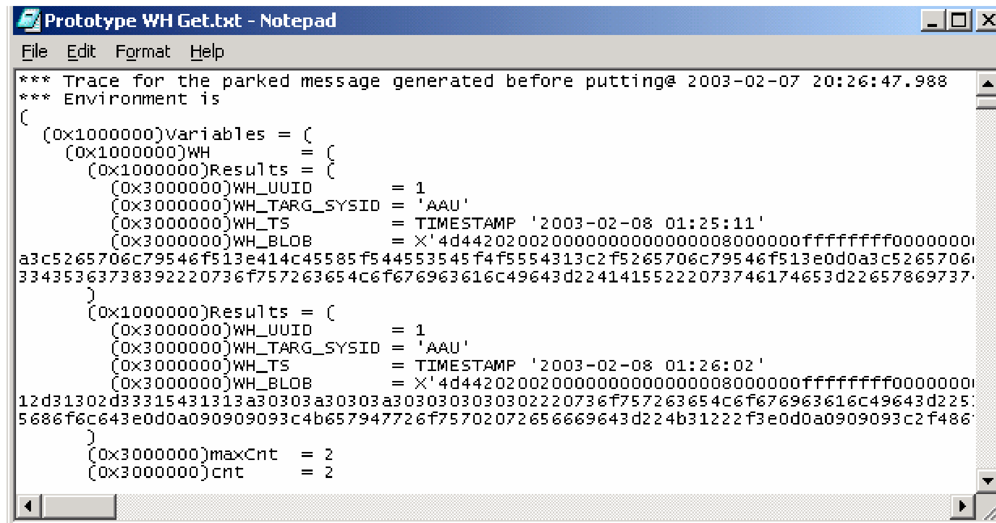

Figure 6: Trace showing that the message BLOB has been read

© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.

```
-- use the MQMD parser to format these fields into the a new field
  in the environment tree
CREATE LASTCHILD OF Environment.Variables.MQMDOut DOMAIN('MQMD')
  PARSE(TempBLOB, 546, 437);
```

The trace in Figure 7 shows the MQMD reconstructed.

The MQHRF2 portion

```
-- with this length, we can parse the MQHRF2 header to the
environment tree
    SET TempBLOB =
  SUBSTRING(Environment.Variables.WH.Results[msgCnt].WH_BLOB FROM
  365 FOR MQHRF2HeaderLength);
     CREATE LASTCHILD OF Environment.Variables.MQRFH2Out
  DOMAIN('MQRFH2') PARSE(TempBLOB, 546, 437);
```

The trace in Figure 8 shows the reconstructed MQHRF2 header.

The message body

```
-- so message body begins either from 365 if no MQHRF2 header, or
  begin from where MQHRF2 header ends
SET beginMsg = beginMsg + MQHRF2HeaderLength;
```

Prototype WH Get.txt - Notepad

File Edit Format Help

```
    (0x3000000)TempBLOB1 = X'4d44202002000000000000000008000000ffffffff000000002202000
    (0x1000000)MQMDOut    = (
      (0x1000000)MQMD = (
        (0x3000000)SourceQueue        = 'ALEX.TEST.IN'
        (0x3000000)Transactional      = TRUE
        (0x3000000)Encoding           = 546
        (0x3000000)CodedCharSetId     = 1208
        (0x3000000)Format             = 'MQHRF2   '
        (0x3000000)Version            = 2
        (0x3000000)Report             = 0
        (0x3000000)MsgType            = 8
        (0x3000000)Expiry             = -1
        (0x3000000)Feedback           = 0
        (0x3000000)Priority           = 0
        (0x3000000)Persistence        = 1
        (0x3000000)MsgId              = X'414d51204d5153492020202020202020201908443e42c0
        (0x3000000)CorrelId           = X'000000000000000000000000000000000000000000000
        (0x3000000)BackoutCount       = 0
        (0x3000000)ReplyToQ           = 'ALEX_TEST_OUT
        (0x3000000)ReplyToQMgr        = 'MQSI
        (0x3000000)UserIdentifier     = 'alexau      '
        (0x3000000)AccountingToken    = X'1601051500000049 5e9d7af700ba481f11393fef0300
        (0x3000000)ApplIdentityData = '                                '
        (0x3000000)PutApplType        = 11
        (0x3000000)PutApplName        = 'QSI IH03\IH03 v3\rfhutil.exe'
        (0x3000000)PutDate            = DATE '2003-02-08'
        (0x3000000)PutTime            = GMTTIME '01:25:11.410'
        (0x3000000)ApplOriginData     = '    '
        (0x3000000)GroupId            = X'000000000000000000000000000000000000000000000
        (0x3000000)MsgSeqNumber       = 1
        (0x3000000)Offset             = 0
        (0x3000000)MsgFlags           = 0
        (0x3000000)OriginalLength     = -1
      )
    )
```

Figure 7: Trace shows the MQMD reconstructed

© 2003. Reproduction prohibited. Please inform Xephon of any infringement.

Prototype WH Get.txt - Notepad

File Edit Format Help

```
    )
    (0x1000000)MQRFH2Out = (
      (0x1000000)MQRFH2 = (
        (0x3000000)Version        = 2
        (0x3000000)Format         = 'MQSTR   '
        (0x3000000)Encoding       = 546
        (0x3000000)CodedCharSetId = 1208
        (0x3000000)Flags          = 0
        (0x3000000)NameValueCCSID = 1208
        (0x1000000)mcd            = (
          (0x1000000)Msd = (
            (0x2000000) = 'XML'
          )
          (0x1000000)Set = (
            (0x2000000) = 'Alex Au Mark Up'
          )
          (0x1000000)Fmt = (
            (0x2000000) = 'XML'
          )
        )
        (0x1000000)usr            = (
          (0x1000000)ReplyInfo = (
            (0x2000000)           = '
'
            (0x1000000)ReplyToQ     = (
              (0x2000000) = 'ALEX_TEST_OUT1'
            )
            (0x1000000)ReplyToQMgr = (
              (0x2000000) = 'MQSI'
            )
          )
        )
      )
    )
```

Figure 8: trace shows the MQHRF2 header reconstructed

```
    SET TempBLOB =
      SUBSTRING(Environment.Variables.WH.Results[msgCnt].WH_BLOB FROM
      beginMsg) ;
    CREATE LASTCHILD OF Environment.Variables.XMLOut DOMAIN('XML')
      PARSE(TempBLOB, 546, 437);
```

The trace in Figure 9 shows the reconstructed message body.

Finally we put the two or three pieces of BLOB together to get the output message.

```
-- Putting them back together
SET OutputRoot.MQMD = Environment.Variables.MQMDOut.MQMD;
SET OutputRoot.MQRFH2 = Environment.Variables.MQRFH2Out.MQRFH2;
```

CONCLUSION

As we can see, the CREATE statement with DOMAIN PARSE can easily reconstruct the MQ headers and its message body.

© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.

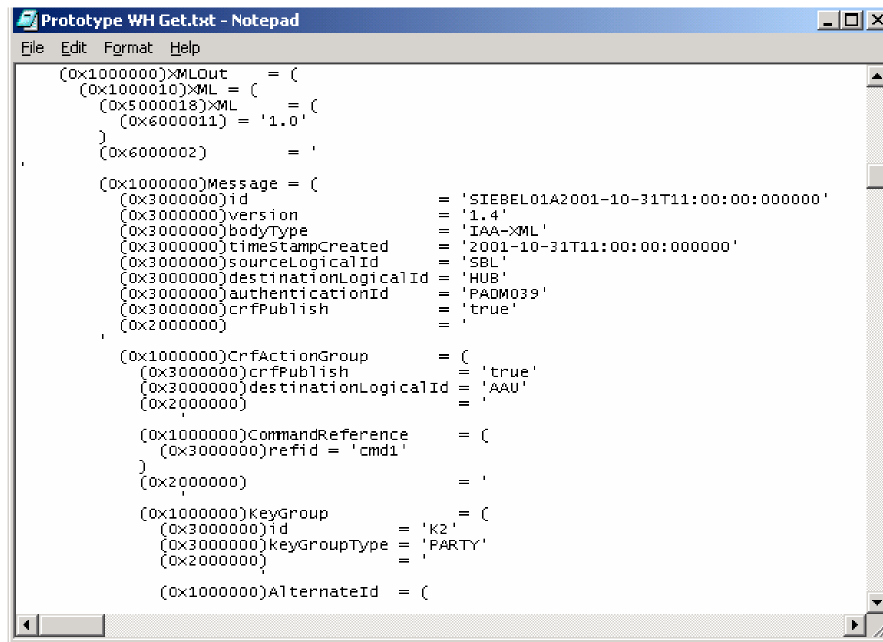

Figure 9: The trace shows the reconstructed message body

This is especially useful for reconstructing the MQHRF2 header, which consists of a variable portion of name value pair. Unlike the fixed length MQMD header there is just no other way to reconstruct the MQHRF2 header easily.

Alex Au
IT Architect
IBM Global Services (Australia)
© IBM 2003

Preserving message order

The WebSphere MQ products provide a function called message grouping, which can be used by a getting application to retrieve a set of related messages in a specified logical order. This article looks at the benefits that message grouping provides and

© 2003. Reproduction prohibited. Please inform Xephon of any infringement.

contains code extracts showing how a group of messages can be constructed at MQPUT time and read back in logical order at MQGET time.

WHAT FUNCTIONS DOES MESSAGE GROUPING PROVIDE?

Logical ordering

Consider an application that puts a set of three related messages to a transmission queue destined for a target queue on another queue manager. The messages are ordered and it is important that the order in which they are read by the getting application from the target queue is the same order as that in which they were put (see Figure 1).

When the messages arrive at the target queue manager it is possible that they cannot be put immediately to the target queue, perhaps because it's full or put-inhibited. The messages would then be put to the dead letter queue if one was available. If some of the messages were put to the target queue and some were dead letter-queued it can affect the order in which the messages are seen by a getting application. If, for example, message one could not be put to the target queue because it was full but messages two and three could be put to the target queue, when message one is later processed by a dead letter queue handler and put to the target queue it will be behind messages two and three, as Figure 2 illustrates. The application getting the messages

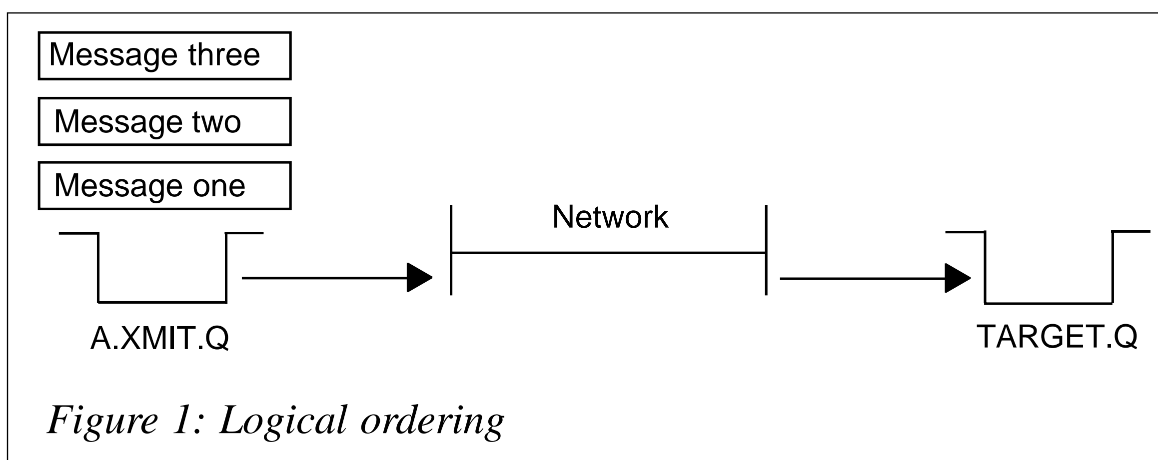

Figure 1: Logical ordering

© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.

from the target queue needs to get them in the order message one, message two, message three.

This is one example of where message grouping can be a benefit. The group of messages on the target queue can be viewed in two ways: the physical order of the messages in the group is now message two, message three, message one; however, the logical order of the messages in the group is message one, message two, message three. By requesting the messages in logical order at MQGET time they will be returned in the order message one, message two, message three, regardless of their physical order on the queue.

Complete groups of messages

Another useful option is the ability to get only complete groups of messages. Consider an application that needs to process groups of three messages. The application issues an MQGET request for the first message in the group, processes it successfully, then goes to get the second message in the group and finds that it is not yet available on the queue, perhaps because of a communications problem with the channel receiving the messages. The application now has two choices: it can either back out the first message in the group or it can wait for the second message to arrive. If the application backs out it has

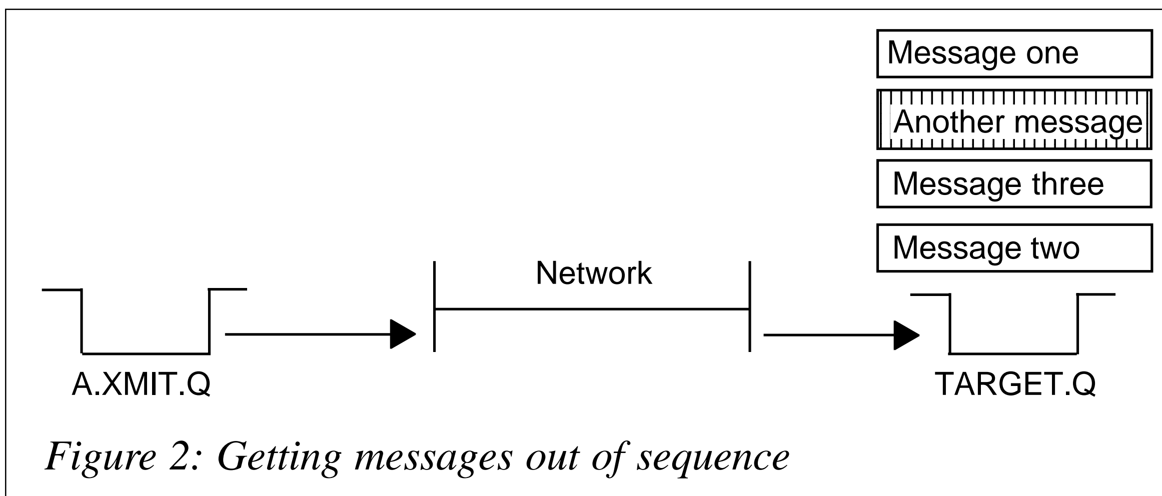

Figure 2: Getting messages out of sequence

© 2003. Reproduction prohibited. Please inform Xephon of any infringement.

wasted resources by unnecessarily processing the first message, whilst if it enters an MQGET wait it is held up when it could be processing other groups that have all three messages available. What the application would like to be able to do is to MQGET the first message of a group only if all the messages in that group are available on the queue.

USING MESSAGE GROUPING

MQPUT time

To benefit from message grouping the messages must be put as part of a group at MQPUT time. This is done by using a version two message descriptor and setting one of the MsgFlags on, as shown here:

```
/* Indicate that this message is part of a group */
MQMD myMqmd;
MyMqmd.Version = MQMD_VERSION_2;
MyMqmd.MsgFlags = MQMF_MSG_IN_GROUP;
```

There are two different approaches you can use when putting a group of messages. These are explained below.

Using 'logical mode'

In the earlier example we looked at an application that put three messages in the same order that the getting application required them. If this is the case we can inform the queue manager at MQPUT time that we are putting the messages in the logical order that we want to retrieve them. The benefit of this is that the queue manager will take care of assigning a unique group identifier to the group and will also assign each message in the group a message sequence number starting from one. This information is stored with the messages as part of their message descriptors so that at MQGET time the queue manager can return the messages in their logical order.

When we put the last message in the group we must tell the queue manager that it is the last message. This indicator informs the queue manager that the group is complete and is used at

© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.

MQGET time when the getting application requests complete groups of messages only. The example code below shows a loop that puts a group of three messages in logical order and tells the queue manager when it is putting the last message.

```
MQMD myMqmd;
MQPMO PutOptions;
/* Indicate that the MQPUT is in logical order */
PutOptions.Options = MQPMO_LOGICAL_ORDER;
i=1;
while(i <= 3)
{
  /* Indicate that this message is part of a group */
  MyMqmd.Version = MQMD_VERSION_2;
  MyMqmd.MsgFlags = MQMF_MSG_IN_GROUP;
  if (i==3)
  {
    /* Indicate that this is the last message in the group */
    MyMqmd.MsgFlags = MyMQMD.MsgFlags | MQMF_LAST_MSG_IN_GROUP;
  }
  /* Issue the actual MQPUT request */    MQPUT(...);
  i++;
}
```

Using 'physical mode'

There are times when the order in which the messages are MQPUT may not match the logical order in which they are to be retrieved. One example could be three different applications A, B, and C, each of which puts one message to make up a group of three messages for the getting application. If these applications are not linked in any way the order in which they put the messages to the queue will be random, but the getting application may always need to process them in the order A, B, C.

In this case the putting applications cannot use **MQPMO_LOGICAL_ORDER** when putting the messages but they can fill in the required fields in the message descriptors so that the getting application can use logical order to retrieve the messages correctly. Each putting application would have to assign the group identifier of the message and the message sequence number. Group identifiers are required to be unique and of course each putting application needs to know the group identifier. (Perhaps if a request message was driving the three

© 2003. Reproduction prohibited. Please inform Xephon of any infringement.

applications the Correl-ID of the request message could be set and used as the Group-ID.)

For the getting application to retrieve the messages in the correct order we would assign message sequence numbers such that application A uses one, B uses two, and C uses three. Application C also specifies the last-message-in-group flag. Given that this is the case, here is the complete set of fields from the message descriptor that would need to be assigned for each of the three messages.

```
MQMD myMqmd;
MyMqmd.Version = MQMD_VERSION_2;
MyMqmd.MsgFlags = MQMF_MSG_IN_GROUP (+ MQMF_LAST_MSG_IN_GROUP for
                                     Application C);
MyMqmd.GroupId = <24 byte unique group identifier>;
MyMqmd.MsgSeqNumber = <message sequence number (in this example,
  either 1, 2 or 3);
```

It is important to note that when an application issues an MQGET in logical order the queue manager will always be looking for a message with sequence number one. If none is found the application will receive MQRC_NO_MSG_AVAILABLE even if there are other messages on the queue.

MQGET time

At MQGET time we must tell the queue manager that we want to get the messages in logical order. As with MQPUT, we should use a version two message descriptor to get the message.

```
/* Indicate that the MQGET is in logical order */
MQGMO GetOptions;
GetOptions.Options = MQGMO_LOGICAL_ORDER;
MQMD myMqmd;
MyMqmd.Version = MQMD_VERSION_2;
```

If we want to get only complete groups of messages this must also be specified:

```
GetOptions.Options |= MQGMO_COMPLETE_GROUP;
```

This will get the first message of a group of messages in logical order. Once the application has retrieved the first message in a group it can only get the next message in the group whilst it is

© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.

getting in logical order. It will continue to get messages from the same group until it gets the message with the last in group indicator, at which point the next get will look for the first message in a new group.

SUMMARY

Message grouping provides a way of ensuring that a group of messages is available for processing and that individual messages are processed in the correct order, regardless of their position on the queue. It does this by providing a logical ordering for the messages within a group and allowing them to be retrieved in that logical order.

Dan Millwood
IBM Hursley (UK)
© IBM 2003

MQSeries message persistence

Do you ever wonder whether the way in which the message persistence property is set makes a difference to MQ message throughput performance?

In a nutshell, MQSeries message persistence defines whether or not a message will survive a restart of the queue manager. Message persistence is defined in one of two ways:

- Explicitly, by setting the Persistence value in the MQMD structure to MQPER_NOT_PERSISTENT or MQPER_PERSISTENT.
- Implicitly, by setting MQMD Persistence to MQPER_PERSISTENCE_AS_Q_DEF. When set implicitly the DEFINE QUEUE attribute DEFPSIST = [YES | NO] determines message persistence. The default is DEFPSIST=NO.

© 2003. Reproduction prohibited. Please inform Xephon of any infringement.

One key point to begin with is that message persistence is a property of the message itself. In fact MQSeries has a terrific feature that permits persistent and non-persistent messages to share the same queue. In my experience I've observed that many people mistakenly associate the persistence property with the queue. While it is true that the queue DEFPSIST parameter will influence message persistence when the MQMD Persistence setting equals MQPER_PERSISTENCE_AS_Q_DEF, the fact remains that persistence is a property of the message.

It is a common practice for developers to simplify programming by always setting the MD persistence value to MQPER_PERSISTENCE_AS_Q_DEF and then leaving it up to the MQ administrators to set the queue property to meet the application requirements with respect to message persistence. This is not a good practice, however, because, while you may exploit this feature with impunity, you may get more than you bargained for.

We examined six cases of message persistence using TGEN for MQSeries, a traffic generation and performance measurement solution (see Table 1). We've run this test for three versions of MQ: V2.1, V5.1, and V5.2.1 – stay tuned for version 5.3 results – where we have observed the relationship among the parameter settings; the results have changed over the releases. First we show the results in terms of transaction rates per second for persistent messages and non-persistent messages. Then we compare performance among the MQ releases.

In terms of performance it makes a difference whether DEFPSIST=YES or NO. With MQ V5.2.1 performance is significantly better when DEFPSIST=YES and the MQMD persistence value is set explicitly. This is contrary to prior releases where, with DEFPSIST=YES, performance was substantially worse compared with where DEFPSIST=NO.

To maximize performance with MQ V5.2.1 always set the DEFINE QUEUE attribute DEFPSIST = YES and set the persistence property for each message in your WMQ program.

© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.

DEFINE QUEUE attribute: MQMD Persistence value:	DEFPSIST=YES Msg/sec	DEFPSIST=NO Msg/sec
MQPER_PERSISTENCE_AS_Q_DEF	*105*	937
MQPER_PERSISTENT	*108*	*104*
MQPER_NOT_PERSISTENT	1836	1565

Note: *Persistent messages*
Non-persistent messages

Table 1: Six examples of message persistence (results for MQ V5.2.1)

The results are shown in Figures 1, 2, 3, and 4, and summarized in the tables that follow, along with extracts from the actual TGEN logs. Figures 1 and 2 show the message rates in messages per second for persistent and non-persistent messages for WMQ V5.2.1.

Turning to comparisons between WMQ releases, please refer to Figures 3 and 4. The important conclusion from these results is how the relationship among the six cases of message persistence within a given release changes from release to release rather than how a particular case has changed from release to release.

To maximize performance for each of the releases studied, note the guidance given in Table 2. Many factors contribute to performance; so your results may vary.

TGEN LOGS

DEFPSIST=NO

tgen_DP_N_NP.log: MQMD Persistence = MQPER_NOT_PERSISTENT

```
[**] Mon Mar 17 17:34:58 2003
     TGEN Response Time Report
[**]          Last Put Time:    0.000
```

© 2003. Reproduction prohibited. Please inform Xephon of any infringement.

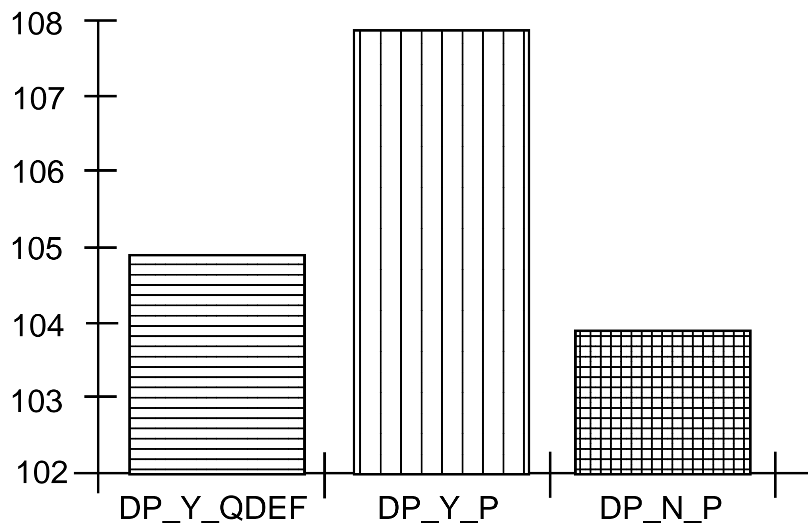

Figure 1: MQ V5.2.1 persistent messages

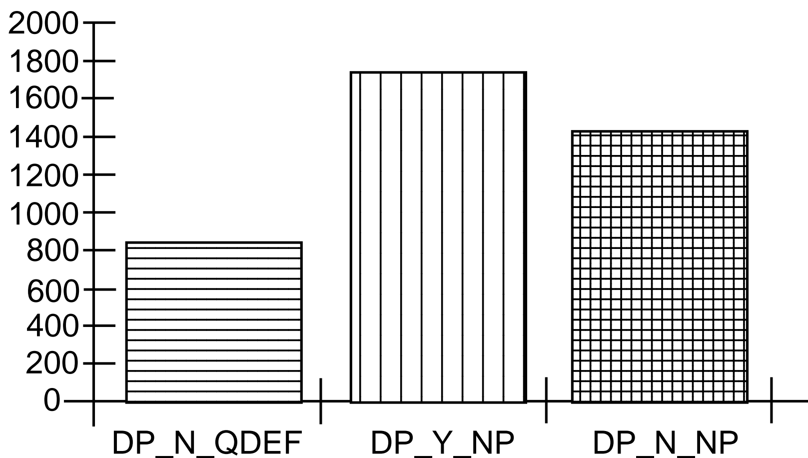

Figure 2: MQ V5.2.1 non-persistent messages

Notes for Figures 1 and 2:

Y-axis is the message rate in messages/second

DP_Y_QDEF:	DEFPSIST=YES MQMD Persistence= MQPER_PERSISTENCE_AS_Q_DEF
DP_Y_P:	DEFPSIST=YES MQMD Persistence= MQPER_PERSISTENT
DP_N_P:	DEFPSIST=NO MQMD Persistence= MQPER_PERSISTENT
DP_N_QDEF:	DEFPSIST=NO MQMD Persistence= MQPER_PERSISTENCE_AS_Q_DEF
DP_Y_NP:	DEFPSIST=YES MQMD Persistence= MQPER_NOT_PERSISTENT
DP_N_NP:	DEFPSIST=NO MQMD Persistence= MQPER_NOT_PERSISTENT

© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.

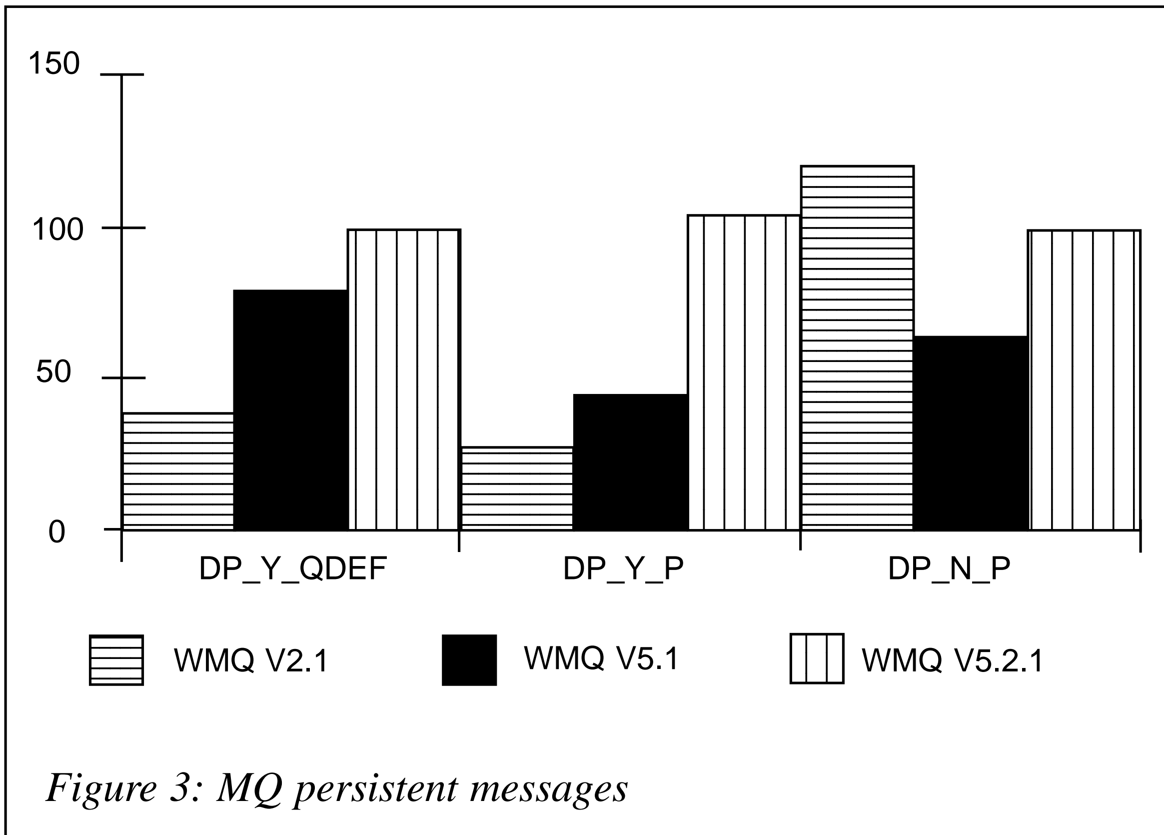

Figure 3: MQ persistent messages

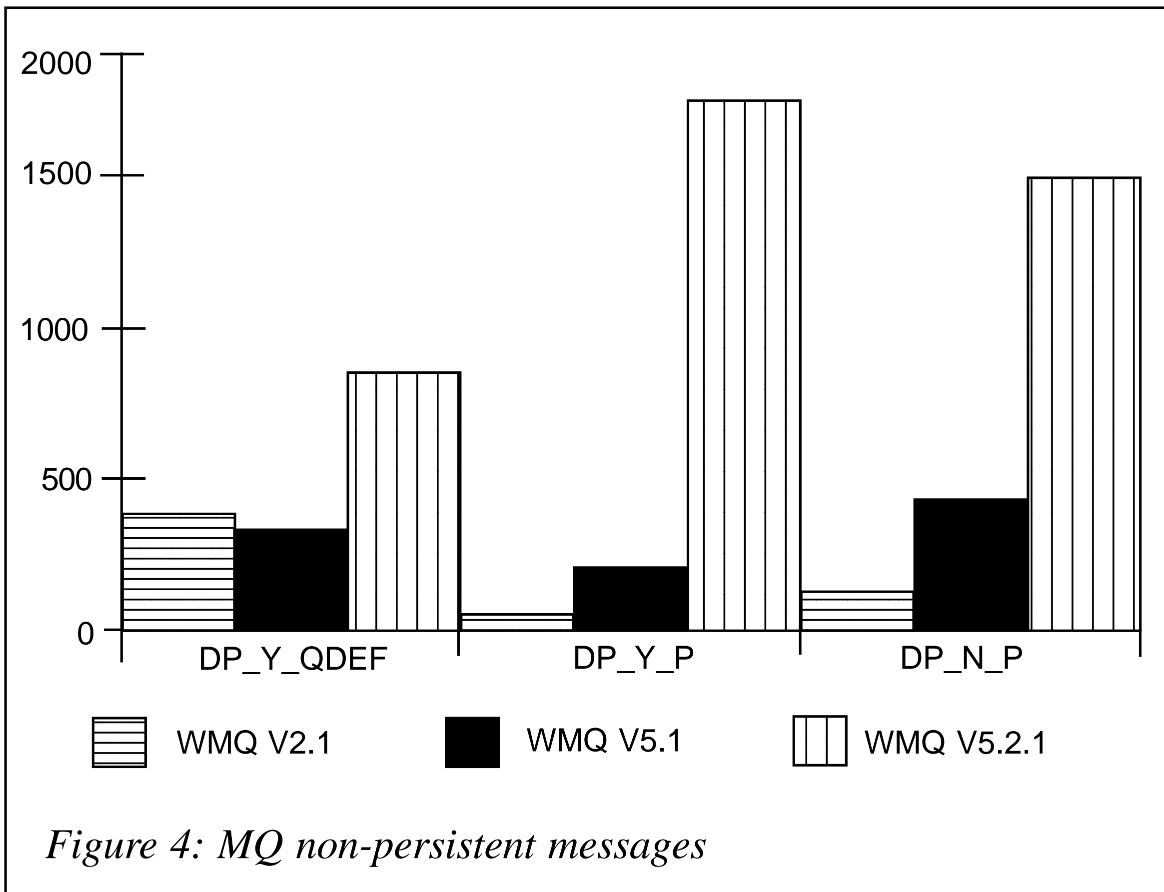

Figure 4: MQ non-persistent messages

© 2003. Reproduction prohibited. Please inform Xephon of any infringement.

```
[**]            Put Time Avg:    0.000
[**]            Put Time Min:    0.000
[**]            Put Time Max:    0.050
[**]           Last Get Time:    0.000
[**]            Get Time Avg:    0.000
[**]            Get Time Min:    0.000
[**]            Get Time Max:    0.020
[**]      Last Response Time:    0.000
[**]       Response Time Avg:    0.001
[**]       Response Time Min:    0.000
[**]       Response Time Max:    0.050
[**] Mon Mar 17 17:34:58 2003
     TGEN Throughput Report
[**] Interval Traffic Summary:   500 messages (150000 bytes) in 0.36
secs (msg/sec: 1388.89, bytes/sec: 416666.66)
[**]        Traffic Totals:5000 messages (1500000 bytes) in 3.19 secs
(msg/sec: 1565.44, bytes/sec: 469630.53)
```

tgen_DP_N_P.log: MQMD Persistence = MQPER_PERSISTENT

```
[**] Mon Mar 17 17:39:36 2003
     TGEN Response Time Report
[**]           Last Put Time:    0.000
[**]            Put Time Avg:    0.004
[**]            Put Time Min:    0.000
[**]            Put Time Max:    0.410
[**]           Last Get Time:    0.010
[**]            Get Time Avg:    0.004
[**]            Get Time Min:    0.000
[**]            Get Time Max:    0.170
[**]      Last Response Time:    0.010
[**]       Response Time Avg:    0.009
[**]       Response Time Min:    0.000
[**]       Response Time Max:    0.410
[**] Mon Mar 17 17:39:36 2003
     TGEN Throughput Report
```

	Persistent messages		Non-persistent messages	
	DEFPSIST	MQMD Persistence	DEFPSIST	MQMD Persistence
V2.1	NO	MQPER_PERSISTENT	NO	MQPER_PERSISTENCE_AS_Q_DEF
V5.1	NO	MQPER_PERSISTENT	NO	MQPER_NOT_PERSISTENT
V5.2.1	YES	MQPER_PERSISTENT	YES	MQPER_NOT_PERSISTENT

Table 2: Performance guidelines

© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.

```
[**] Interval Traffic Summary:   500 messages (150000 bytes) in 4.43
secs (msg/sec: 112.97, bytes/sec: 33890.64)
[**]         Traffic Totals:5000 messages (1500000 bytes) in 48.21 secs
(msg/sec: 103.72, bytes/sec: 31114.52)
```

tgen_DP_N_QDEF.log: MQMD Persistence = MQPER_PERSISTENCE_AS_Q_DEF

```
[**] Mon Mar 17 17:40:32 2003
     TGEN Response Time Report
[**]           Last Put Time:    0.000
[**]            Put Time Avg:    0.000
[**]            Put Time Min:    0.000
[**]            Put Time Max:    0.080
[**]           Last Get Time:    0.000
[**]            Get Time Avg:    0.000
[**]            Get Time Min:    0.000
[**]            Get Time Max:    0.070
[**]      Last Response Time:    0.000
[**]       Response Time Avg:    0.001
[**]       Response Time Min:    0.000
[**]       Response Time Max:    0.080
[**] Mon Mar 17 17:40:32 2003
     TGEN Throughput Report
[**] Interval Traffic Summary:   500 messages (150000 bytes) in 0.45
secs (msg/sec: 1108.65, bytes/sec: 332594.22)
[**]         Traffic Totals:5000 messages (1500000 bytes) in 5.34 secs
(msg/sec: 936.68, bytes/sec: 281004.09)
```

DEFPSIST=YES

tgen_DP_Y_NP.log: MQMD Persistence = MQPER_NOT_PERSISTENT

```
[**] Mon Mar 17 17:47:09 2003
     TGEN Response Time Report
[**]           Last Put Time:    0.000
[**]            Put Time Avg:    0.000
[**]            Put Time Min:    0.000
[**]            Put Time Max:    0.020
[**]           Last Get Time:    0.010
[**]            Get Time Avg:    0.000
[**]            Get Time Min:    0.000
[**]            Get Time Max:    0.020
[**]      Last Response Time:    0.010
[**]       Response Time Avg:    0.000
[**]       Response Time Min:    0.000
[**]       Response Time Max:    0.020
[**] Mon Mar 17 17:47:09 2003
     TGEN Throughput Report
```

© 2003. Reproduction prohibited. Please inform Xephon of any infringement.

```
[**] Interval Traffic Summary:   500 messages (150000 bytes) in 0.27
secs (msg/sec: 1845.02, bytes/sec: 553505.50)
[**]         Traffic Totals:5000 messages (1500000 bytes) in 2.72 secs
(msg/sec: 1835.54, bytes/sec: 550660.75)
```

tgen_DP_Y_P.log: MQMD Persistence = MQPER_PERSISTENT

```
[**] Mon Mar 17 17:49:43 2003
     TGEN Response Time Report
[**]            Last Put Time:    0.010
[**]             Put Time Avg:    0.005
[**]             Put Time Min:    0.000
[**]             Put Time Max:    0.120
[**]            Last Get Time:    0.000
[**]             Get Time Avg:    0.004
[**]             Get Time Min:    0.000
[**]             Get Time Max:    0.191
[**]       Last Response Time:    0.010
[**]        Response Time Avg:    0.008
[**]        Response Time Min:    0.000
[**]        Response Time Max:    0.201
[**] Mon Mar 17 17:49:43 2003
     TGEN Throughput Report
[**] Interval Traffic Summary:   500 messages (150000 bytes) in 4.27
secs (msg/sec: 117.21, bytes/sec: 35161.74)
[**]         Traffic Totals:5000 messages (1500000 bytes) in 46.10 secs
(msg/sec: 108.47, bytes/sec: 32540.78)
```

tgen_DP_Y_QDEF.log: MQMD Persistence = MQPER_PERSISTENCE_AS_Q_DEF

```
[**] Mon Mar 17 17:52:09 2003
     TGEN Response Time Report
[**]            Last Put Time:    0.010
[**]             Put Time Avg:    0.005
[**]             Put Time Min:    0.000
[**]             Put Time Max:    0.350
[**]            Last Get Time:    0.000
[**]             Get Time Avg:    0.004
[**]             Get Time Min:    0.000
[**]             Get Time Max:    0.040
[**]       Last Response Time:    0.010
[**]        Response Time Avg:    0.009
[**]        Response Time Min:    0.000
[**]        Response Time Max:    0.370
[**] Mon Mar 17 17:52:09 2003
     TGEN Throughput Report
[**] Interval Traffic Summary:   500 messages (150000 bytes) in 4.78
secs (msg/sec: 104.67, bytes/sec: 31400.46)
```


© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.

```
[**]        Traffic Totals: 5000 messages (1500000 bytes) in 47.77 secs
(msg/sec: 104.67, bytes/sec: 31401.77)
```

Tom Krpta
Founder, President
Stellar Software (USA)
© Stellar Software 2003

The Extended Transactional Client

A new option has recently been added to WebSphere MQ (WMQ), called the Extended Transactional Client. This article describes the function that has been added and explains why and where you would use it.

Applications have always had a number of ways in which they can use transactions with WMQ. The simplest mechanism involves only messages on queues, which can all be committed simultaneously to their queues with the MQCMIT verb. This is available with both bindings mode applications (where the application is on the same machine as the queue manager) and with client applications (using CLNTCONN and SVRCONN channels).

Many applications, however, want to update database tables as part of the same transaction and this can best be done with a transaction manager coordinating the updates to both WMQ and the database with a two-phase commit protocol.

WMQ can be used this way under the control of an external product such as CICS, IMS, or Tuxedo. On distributed platforms (eg Windows and Unix) WMQ can also operate as a transaction manager, where the application uses the MQBEGIN verb. Until now all of these functions have been available only when the application is running on the same machine as the queue manager.

The Extended Transactional Client (ETC) is a new licensed feature, which adds a two-phase commit protocol to the MQI

© 2003. Reproduction prohibited. Please inform Xephon of any infringement.

client. It has been designed to be an ‘add-on’ rather than a replacement for the existing client libraries so that existing programs that use the client channels in the usual way do not need to change or be relinked.

WHAT DOES IT DO?

The Extended Transactional Client operates in conjunction with an external product, such as WebSphere Application Server (WAS) or Tuxedo, so that applications can be written using those products’ transactional APIs while connecting to a queue manager on a different machine. This allows a degree of concentration of messaging activity with perhaps a smaller number of queue managers and machines to administer.

Note that an external coordinator is required. The MQBEGIN verb has not been added to the client so you cannot use WMQ as a transaction manager. There are also no plans to support MQBEGIN in any clients in the future.

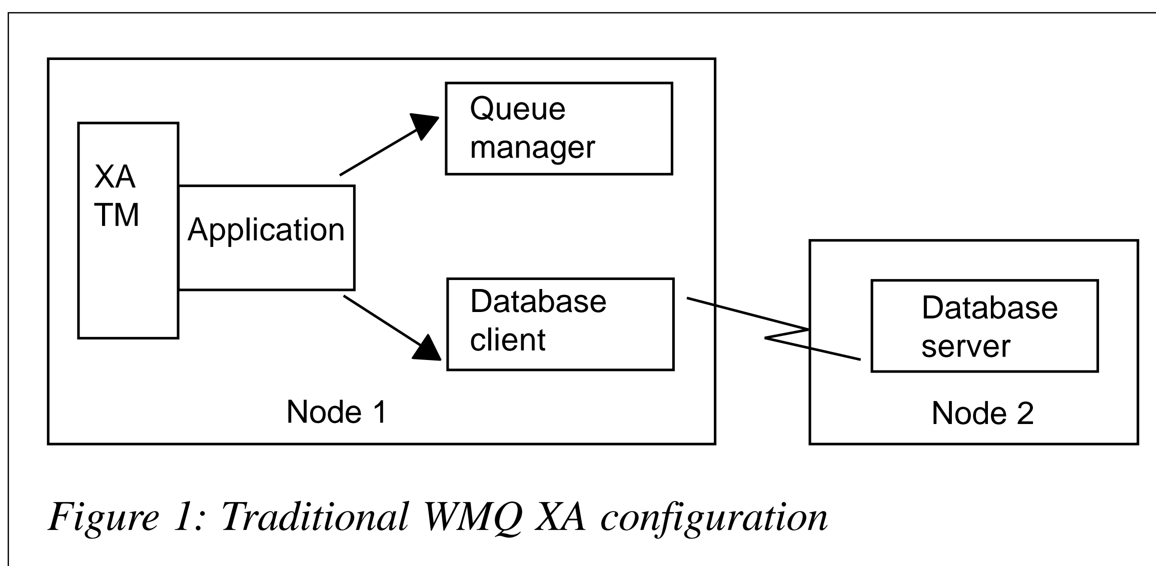

Figure 1: Traditional WMQ XA configuration

Figure 1 shows a configuration that has always been supported with WMQ. Figure 2 shows how that has been extended, allowing the queue manager to move to a separately-controlled machine.

© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.

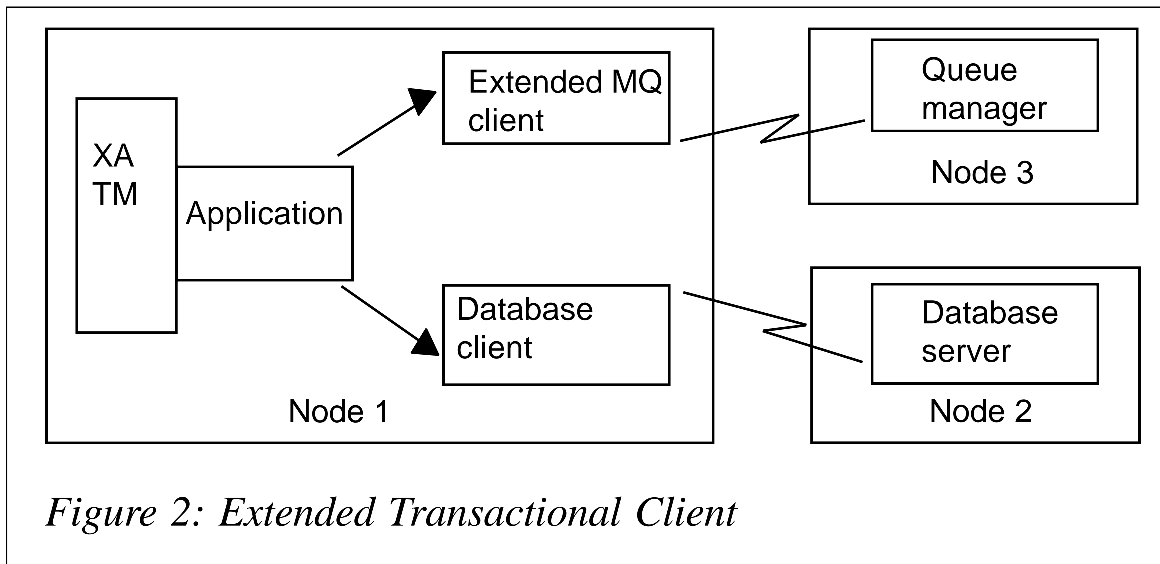

Figure 2: Extended Transactional Client

SUPPORTED INTERFACES

The XA interface, specified by the X/Open Group, is the main standard for bridging between Transaction Managers (TMs) and Resource Managers (RMs). This interface is defined using the C language, with structures, constants, and function calls. All RMs that support XA must document the name of a data structure, which any XA-compliant TM can then access. These structures are defined in the new libraries or DLLs shipped with the ETC so that applications can be linked and configuration files for the TM created.

Two other important interfaces are also supported, built on top of the XA functions. In a Java environment the calls from a TM to RM are defined by the JTA standard. The ETC also provides libraries to allow Microsoft Transaction Server to communicate with WMQ via the client protocols.

For programmers there should be no difference from what is currently supported. The only change that a developer will see is that different libraries need to be selected for the link phase. Similarly administrators of a TM will be able to modify the configuration so that client connections can be used instead of bindings mode.

© 2003. Reproduction prohibited. Please inform Xephon of any infringement.

SUPPORTED VERSIONS OF WMQ

In order for the ETC to work, both the client and server components of WMQ must understand the XA protocols, which are sent across the CLNTCONN/SVRCONN channels.

For client machines this was added in V5.3 by CSD03, which must be installed before the ETC feature can be added. All of the V5.3 distributed server platforms (such as AIX, Solaris, and Windows) support the XA client flows. Although the intention was that no CSD would be required, some late-breaking fixes mean that CSD03 is recommended there too. It will also be possible to attach a client to WMQ for z/OS when version 5.3.1 of that product is available; earlier versions reject attempts to send XA flows.

SUPPORTED TRANSACTION MANAGERS

Several TMs have been tested for compatibility with this feature. This includes WAS, Tuxedo, TXSeries, and MTS. While XA is a defined standard we know that some products do not always adhere to that standard and testing is essential to ensure the reliability of the transactions.

Why use it?

Anyone considering using this feature already knows why they want to use a TM: to keep updates to multiple RMs in a single atomic transaction. It is most likely that the application is using a combination of SQL and MQI calls.

Choosing whether to use the ETC or the original bindings mode to a queue manager should be seen as a topology and ease-of-management question. It is not a way to get cheaper access to WMQ functions because the ETC has to be licensed; it is not free in the way that the regular client libraries are. The first question is whether you intend to use WMQ as the TM. If so, then you cannot use the ETC as it requires an external TM. The next question to ask is whether an MQI client connection would be appropriate if you were to ignore the transactionality requirements of the application.

© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.

- Will the connection between the client machine and the server be reliable? If that connection is not reliable an MQI client is probably not suitable.
- Do you have security requirements between a client and a server? That might mean installation and management of certificates or other security functions.
- Is your application performance-sensitive? A good rule-of-thumb is that every MQI verb takes about 1ms longer when going across a client channel when compared with running the same operation locally.
- Do you have the ability to manage client channels and machines?

Answers to these questions should guide whether you should be looking at using the ETC or continuing to use server bindings for your transactional applications.

APPLICATION AVAILABILITY

Some people consider that using the ETC will improve application availability as they may now have fewer queue managers to operate and monitor, while permitting a larger number of client machines that can handle incoming work.

On the other hand, using the ETC may reduce application availability if there is a long downtime when either the TM or the queue manager is not running; transactions will be held in-doubt and locks may be held. In the 'classic' configuration it is very likely that both the TM and the RM would be simultaneously unavailable as the machine broke down; recovery would be automatic when the machine was restarted. There is no easy answer or rule for this dilemma; the solution architect will have to consider what is appropriate and acceptable.

SPECIAL CONSIDERATIONS FOR WAS

If you are using WAS V5, which includes the JMS classes and a version of WMQ, then there is no need to install the ETC

© 2003. Reproduction prohibited. Please inform Xephon of any infringement.

feature; WAS 5 can drive the J2EE and JTA operations for JMS directly. Earlier versions of WAS still require the ETC, as will any use of the non-JMS interfaces.

Developing applications for the ETC

There is very little new that an application programmer needs to know in order to use the ETC. It has been designed so that existing applications that use bindings mode can be very simply relinked to get access to the client mode. There are some new libraries, and the XA switch structure is available through them. In some cases the application program has to provide the XAOpenString, a configuration parameter that shows how to connect to the resource. For the ETC the XAOpenString has been defined to allow specification of client channel definitions. You can either define the equivalent of the MQSERVER environment variable or point to the channel definition file.

This is required because the initial connection between a TM and WMQ might be done through the xa_open verb, whose parameters are standardized and cannot be modified. Because we need to know, during xa_open, how to reach the queue manager and because we need to use exactly the same route as the application's subsequent MQCONN call, the XAOpenString gives that information.

Note that the queue manager field in the XAOpenString must match the name of the queue manager you are connecting to. This is to ensure that applications always connect to the same place. Using wildcards or generic connections to select one from a list of queue managers is permitted with non-transactional clients but not allowed once you start using XA. This is so that, after a failure, recovery operations know exactly where any in-doubt transactions are held. An example XAOpenString is:

```
channel=MARS.SVR,trptype=tcp,conname=MARS(1415),qmname=MARS
```

This tells the client the same information as would be picked from the MQSERVER environment variable, along with the name of the queue manager. You need to read the documentation for your

© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.

TM to determine how to configure the XAOpenString. For some products it will be typed into an ini file, for others it might be coded inside the application program. More information about the ETC can be found at *http://www.ibm.com/software/integration/wmq/transclient.html.*

Mark E Taylor
MQSeries Technical Strategist
IBM Hursley (UK)

© IBM 2003

July 2002 – June 2003 index

There follows an index of all topics covered in *MQ Update* since Issue 37, July 2002. The numbers in **bold** are issue numbers and the ones in brackets are page numbers. Back issues of *MQ Update* are available from Xephon – see page 2 for details.

© 2003. Reproduction prohibited. Please inform Xephon of any infringement.

Topic	**Issue** (page)
API Exits	**37** (24-42)
Back-ups	
on AIX	**37** (42-43)
Unix queue manager	**41** (14-22)
Channel	
Event Queue Viewer	**44** (35-43)
Heartbeat	**39** (43-47)
Security exit	**40** (3-4)
Clusters: hints and tips	**41** (23-32)
Configuring	
Termination of work instances	**46** (9-29)
Web client on Windows NT	**38** (19-32)
	39 (12-21)
Controlling resources	**47** (3-12)
Data grouping	**44** (30-35)
Error log analysis	**37** (7-23)
Exception processing	
Request/reply messages	**38** (3-12)
Subflows	**39** (32-42)
Extended Transactional Client	**48** (37-42)
Firewalls	**46** (3-9)
Generic profiles	**41** (3-14)
Java MQMail	**44** (9-17)
JMS	
To IMS via WMQ	**45** (29-43)
Authentication, authorization	**43** (36-43)
Message	
Availability	**39** (22-32)
Availability update	**44** (18)
Expiry	**44** (3-9)
Persistence	**48** (30-37)
Preserving order	**48** (25-29)
MQAI	
Creating objects	**46** (45-51)
Listing queues	**44** (19-30)
MQJava for Notes	**42** (36-43)
MQSC facilities	**42** (3-8)
MQSeries First Steps	**45** (3-11)
MQTelnet interface	**46** (30-44)

Topic	**Issue** (page)
Natural – MQ Interface	**40** (32-32)
Nodes	
NEON	**37** (3-7)
Writing	**48** (3-8)
Queue manager alias	**42** (29-36)
Queues	
A tale of two queues	**40** (9-26)
Very large queues	**40** (5-8)
Parameters: Solaris Kernel	**38** (41-43)
Performance Event Queue Viewer	
MQ for OS/390	**38** (12-18)
Publish/Subscribe	
Service Pack MA0C	**40** (26-32)
WMQI	**42** (9-20)
Scripts	**43** (3-9)
Security	**38** (33-41)
Sending data	**45** (12-19)
SSL	
Improving performance	**47** (30-36)
Support	**47** (22-30)
WMQ for Windows	**42** (20-27)
Temporary files	**43** (34-36)
Trigger monitor	
Another batch monitor	**43** (27-34)
Batch	**40** (33-34)
CKTI	**40** (34-39)
Utilities	
CSQ4BVJ1	**42** (28)
dmpmqaut parser	**47** (37-43)
WebSphere Financial Network Integrator: technical preview	**40** (40-47)
WMQI	
Cross-reference	**39** (3-12)
Performance evaluation	**47** (12-22)
Plug-in nodes	**43** (10-27)
Procedures	**45** (20-28)
Retrieving messages	**48** (9-24)
Workflow	
Client/server setup	**41** (33-47)

© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.

MQSoftware has signed an agreement with Willow Technology that, claims the company, will expand significantly MQSoftware's Q Pasa! platform support for WebSphere MQ.

Willow specializes in operating system and middleware support for heterogeneous operating environments. Under the terms of the agreement Willow will produce Q Pasa! agents for all its WebSphere MQ server platforms.

The agreement is also intended to accelerate the use of Q Pasa! for international customers who have legacy platforms and who require real-time operational monitoring and management information.

Willow Technology's WebSphere MQ Server support includes the following platforms: BSD, Data General DG/UX, Hewlett-Packard MPE/iX, IBM DYNIX/ptx, Linux (Alpha and SPARC Editions), NCR MP-RAS, SCO OpenServer, SGI IRIX, UnixWare, and others.

For more information contact:
MQSoftware,1660 South Highway 100, Suite 400, Minneapolis, Minnesota 55416, USA.
Tel: +1 952 345 8720.
Fax: +1 952 345 8721.
Web: http://www.mqsoftware.com

MQSoftware, Surrey Technology Centre, 40 Occam Road, Surrey Research Park, Guildford, Surrey, GU2 7YG, UK.
Tel: +44 1483 295400.
Fax: +44 1483 573704.

* * *

Candle has introduced PathWAI Secure for WebSphere MQ, which is said to expand the protection of information across the WebSphere MQ environment by combining existing security and management applications with encryption software from RSA.

Increased authentication verifies the identities of message senders and recipients, PKI support strengthens security, and there's an expanded ability to validate that data transmissions and message archives have not been altered.

The software supplements the user authorization capabilities of various external security programs, such as RACF, ACF2, and Top Secret on OS/390, as well as operating system security tools for Unix and Windows systems.

The software supports OS/390, z/OS, AS/400, AIX, HP-UX, Solaris, and Windows NT, 2000, and XP.

For more information contact:
Candle, 100 N Sepulveda Blvd, El Segundo, CA, 90245, USA.
Tel: +1 310 535 3600.
Fax: +1 310 727 4287.
Web: http://www.candle.com

Candle, 1 Archipelago, Lyon Way, Frimley, Camberley, Surrey, GU16 7ER, UK.
Tel: +44 1276 414 700.
Fax: +44 1276 414 777.

* * *