MA18: A Rexx Interface to IBM MQSeries for MVS/ESA
Version 2.0

Robert Harris
Object Technology Products,
IBM UK Labs Ltd.
Hursley Park
Hursley
Winchester. SO21 2JN
United Kingdom
VNET: HARRISR at WINVMC
Internet: HARRISR@VNET.IBM.COM
Tele: (44) 1962 818151
Take Note!

Before using this User's Guide and the product it supports, be sure to read the general information under "Notices".


This edition applies to Version 2.0 of MA18: A Rexx Interface to IBM MQSeries for MVS/ESA and to all subsequent releases and modifications until otherwise indicated in new editions.

A form for reader's comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM United Kingdom Laboratories
Transaction Systems Marketing Support (MP207)
Hursley Park
Hursley
Hampshire, SO21 2JN, England

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you. You may continue to use the information that you supply.

© Copyright International Business Machines Corporation 1994. All rights reserved.
Note to U.S. Government Users - Documentation related to restricted rights - Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corporation.
## Contents

Chapter 1. Introduction .......................................................... 1

Chapter 2. Installing the SupportPac ........................................... 2
  Installing the interface .................................................. 4

Chapter 3. TSO Support .......................................................... 5
  TSO Batch Support ................................................... 5
  TSO Native Support ................................................... 6
  TSO Split Screen Support ............................................. 6
  TSO ISPF Support ................................................... 6

Chapter 4. Interface Design Philosophy ...................................... 10
  Operations ............................................................... 11
  Return Codes .......................................................... 11
  Last Operation .......................................................... 12
  Return Code naming .................................................. 12
  Message Lengths ...................................................... 12
  Internal procedures .................................................. 13
  Header and Event processing ......................................... 13
  ZLIST ................................................................. 14
  Stem Variables ........................................................ 16
  Trace ................................................................. 17

Chapter 5. General points ....................................................... 11
  Operations ............................................................... 11
  Return Codes .......................................................... 11
  Last Operation .......................................................... 12
  Return Code naming .................................................. 12
  Message Lengths ...................................................... 12
  Internal procedures .................................................. 13
  Header and Event processing ......................................... 13
  ZLIST ................................................................. 14
  Stem Variables ........................................................ 16
  Trace ................................................................. 17

Chapter 6. Handling MQ Descriptors ........................................... 18
  The Object Descriptor .................................................. 20
  The Message Descriptor ................................................ 21
  The Get Message Option Structure .................................... 22
  The Put Message Options Structure ................................... 23

Chapter 7. The Interface ......................................................... 24
  Common Return Codes .................................................. 25
  Initialisation ........................................................... 26
    Description .......................................................... 26
    Parameters .......................................................... 26
    Call ................................................................. 26
    Additional Interface Return Codes and Messages ................. 26
    Example ............................................................. 26
  Setting Literals ........................................................ 27
    Description .......................................................... 27
    Parameters .......................................................... 27
    Call ................................................................. 27
    Additional Interface Return Codes and Messages ................. 27
    Example ............................................................. 27
  Termination ............................................................ 28
    Description .......................................................... 28
    Parameters .......................................................... 28
    Call ................................................................. 28
    Additional Interface Return Codes and Messages ................. 28
    Example ............................................................. 28
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Parameters</th>
<th>Call</th>
<th>Additional Interface Return Codes and Messages</th>
<th>Example</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>RXMQVCONN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>29</td>
</tr>
<tr>
<td>RXMQVDISC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>31</td>
</tr>
<tr>
<td>RXMQVOPEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>32</td>
</tr>
<tr>
<td>RXMQVCLOSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>RXMQVINQ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>37</td>
</tr>
<tr>
<td>RXMQVSET</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>40</td>
</tr>
<tr>
<td>RXMQVCMIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>43</td>
</tr>
<tr>
<td>RXMQVBACK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>44</td>
</tr>
<tr>
<td>RXMQVGET</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>45</td>
</tr>
</tbody>
</table>
RXMQVPUT ........................................ 48
Description ........................................ 48
Parameters ........................................ 48
Call .................................................. 48
Additional Interface Return Codes and Messages ........................................ 49
Example ............................................ 50
RXMQVQ SIGNAL .................................... 51
Description ........................................ 51
Parameters ........................................ 51
Call .................................................. 51
Additional Interface Return Codes and Messages ........................................ 52
Example ............................................ 53
RXMQVBROWSE ...................................... 54
Description ........................................ 54
Parameters ........................................ 54
Call .................................................. 54
Additional Interface Return Codes and Messages ........................................ 55
Example ............................................ 55
RXMQV HXT ........................................... 56
Description ........................................ 56
Parameters ........................................ 56
Call .................................................. 56
Additional Interface Return Codes and Messages ........................................ 57
Extracted information ........................................ 58
Example ............................................ 60
RXMQVEVENT .......................................... 61
Description ........................................ 61
Parameters ........................................ 62
Call .................................................. 62
Usage Notes ......................................... 62
Additional Interface Return Codes and Messages ........................................ 63
Extracted information ........................................ 64
Example ............................................ 72
RXMQVTM ............................................. 74
Description ........................................ 74
Parameters ........................................ 75
Call .................................................. 75
Additional Interface Return Codes and Messages ........................................ 76
Trigger information ........................................ 77
Examples ............................................ 88

Chapter 8. Interface Example ........................................ 80

Appendix A. Rexx/MQ constants ........................................ 84

Appendix B. Rexx/MQ Return Code constants ........................................ 89
Figures

1. TSO Batch JCL .................................................. 5
2. ISPF Exec (MA18T1) .............................................. 7
3. ISPF Panel (MA18P1) .............................................. 8
4. ISPF Panel (success) ............................................... 9
5. ISPF Panel (failure) ............................................... 9
6. ZLIST and Event processing ....................................... 15
7. Removing funny event data ...................................... 73
8. A Trigger Monitor ................................................. 78
9. A Rexx Triggered Process ....................................... 79
10. Interface example ................................................. 80

Tables

1. Object Descriptor Mappings ........................................... 20
2. Message Descriptor Mappings ........................................ 21
3. Get Message Options Mappings ..................................... 22
4. Put Message Options Mappings ..................................... 23
5. Transmission Queue Message Extracts ................................ 58
6. Dead Letter Queue Message Extracts ................................ 59
7. Event Names .......................................................... 64
8. Event Names .......................................................... 65
9. Events and Components .............................................. 71
10. Trigger Components .................................................. 77
Notices.

The following paragraph does not apply in any country where such provisions are inconsistent with local law.

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates.

Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used. Any functionally equivalent program that does not infringe any of the intellectual property rights may be used instead of the IBM product. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, New York 10594, USA.

The information contained in this document has not be submitted to any formal IBM test and is distributed AS IS. The use of the information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item has been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

IBM
MQSeries
OS/2
REXX
Acknowledgments

The material in this SupportPac was provided by Robert Harris, Transaction Systems New Projects, IBM Hursley Park Laboratories, UK.
# Summary of Changes

<table>
<thead>
<tr>
<th>Date</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1.0</strong> - 30th January 1996</td>
<td>Initial version</td>
</tr>
<tr>
<td><strong>1.0.2</strong> - 22nd May 1996</td>
<td>Include instruction for transferring files to MVS using ftp</td>
</tr>
<tr>
<td><strong>2.0</strong> - 19th June 1997</td>
<td>Recompiled to the latest version of MQ (1.1.4), adding new information on event interpretation; added pointers to the NT and Netview versions of this utility</td>
</tr>
</tbody>
</table>
Preface

This SupportPac provides a Rexx Interface for IBM MQSeries Version 1.1.4 on MVS/ESA. It permits the usage of MQ function within a Rexx Exec (native or in the ISPF environment).

This publication is intended to help persons who are investigating IBM MQSeries solutions to position them within their installation's needs. The information in this publication is not intended as the specification of any programming interfaces that are provided by IBM MQSeries for MVS/ESA Version 1.1.4, Program Number 5595-137 or any other Product. See the PUBLICATIONS section of the IBM Programming Announcement for IBM MQSeries for MVS/ESA Version 1.1.4 or the MQI product planned to be used, for more information about what publications are considered to be product documentation.

This interface is different to that described in the Application Programming Reference (SC33-1212-02) book, as the API is customised for the Rexx environment. However, with a few exceptions, all the function described in the APR is available. Some extensions to the API are also provided to ease the usage of the interface.

The Audience

This SupportPac is designed for people who:

- Want to explore Message Queuing within the MVS/ESA environment
- Want to place Message Queueing function within native Rexx execs
- Want to place Message Queueing function within the Rexx/ISPF environment
- Need to prototype MQ Applications within the MVS/ESA environment
- Are interested in the Design of Message Queuing Applications

Users should have a general awareness of Message Queuing function, and be familiar with Rexx coding and the ISPF environment to get the best out of this SupportPac.

What is in this SupportPac

- An MVS/ESA load module that provides support for Rexx/MQ/MVS access to a local Queue Manager
- A native Rexx Exec which demonstrates usage of the interface
- This paper which documents the interface (in various formats).
Other SupportPacs

MA19    A SupportPac containing a Rexx interface to MQSeries for MVS/ESA to issue MQ Commands (RXMQVC).
MA1D    A version of this SupportPac for the Netview for MVS environment
MA31    A SupportPac containing a Rexx interface to MQSeries for OS/2
MA7A    A SupportPac containing a Rexx interface to MQSeries for Windows NT
Chapter 1. Introduction

This SupportPac provides a Rexx Interface, within the MVS/ESA environment, for Message Queueing access.

A single module RXMQV is supplied, which must be placed into a suitable load library. The module does not run in Supervisor State, and so does not require to be placed in an authorised library.

A full implementation of the API as described in the MQSeries for MVS/ESA Version 1.1.4 Application Programming reference SC33-1212-02 is provided, so this book will be needed to use the Rexx Interface. However, there are three restrictions:

- MQPUT1 is not supported, as the author feels that the loss of control that this verb engenders is not suitable for the Rexx environment
- MQINQ only permits a single attribute to be examined, as support for multiple access is too complicated in the Rexx environment
- MQSET only permits the setting of a single attribute

In addition to the standard API functions, the Rexx Interface provides a number of extensions to the API to ease the coding of an Exec:

- A QSIGNAL function is provided to engender support for Get.Signal processing
- A Browse function is provided
- An Header Extraction function is provided to split up a message from a Transmission Queue or a Dead letter Queue into its components
- An Event interpretation function is provided to split up a message from an Event Queue into its components
- An Trigger Message function is provided to split up a Trigger message from an Initiation Queue and to generate/parse execution parameters

This utility will not work within the Rexx environment provided by Netview for MVS. You need SupportPac MA1D instead.
Chapter 2. Installing the SupportPac

Take the following actions to install the SupportPac from the MA18.ZIP file:

1. Use INFOZIP’s UNZIP32 to unpack the MA18.ZIP file.
   
   This produces
   - MA18SEQX (the RXMQV load module)
   - MA18TEST.JCL (sample JCL)

2. MA18SEQX needs to be transferred to the destination TSO system as a sequential binary file with a record format of FB 80. Use one of the following methods to accomplish this:
   - Use the Communications Manager/2 send command below to send the file to TSO as a sequential binary file called MA18SEQ
     
     send ma18seqx A:ma18seq
     
     where A is the TSO session ID.
   - To send it via ftp ensure the BINARY option is set then use the following commands:
     
     site fixrecfm 80
     
     put ma18seqx ma18seq
   - With Personal Communications, use the “Send Files to Host” option under the Transfer menu item to transmit to TSO
     
     PC File ma18seqx
     Host File ma18seq
     Transfer Type loadlib

     The Transfer type of loadlib may need to be correctly setup. To do this, use the “Setup.Define Transfer Types” option under the Transfer menu item and create the loadlib type with the Ascii, CRLF and Append checkboxes all unselected, the Fixed radio button selected and the LRECL set to 80

3. On TSO, issue the following commands to unload this sequential file into TSO partitioned dataset:

   receive indaname(MA18SEQ)

   When prompted for a filename, reply

   dsn(MA18LOAD)

   This creates a PDS called mvsuserid.MA18LOAD with the single member RXMQV

4. Use ISPF 3.2 to delete the MA18SEQ file

5. Use ISPF 3.3 to copy member RXMQV from file MA18LOAD into your load library (which must have
   DCB=(DSORG=PO,RECFM=U,LRECL=32760,BLKSZ=32760))

6. Use ISPF 3.2 to delete the MA18LOAD file

7. Change any appropriate TSO procedure to use the Load Library (if the first library in the SYSLIB concatenation has a BLKSIZE less than 32760, you should add a DCB=BLKSIZE=32760 statement)

   Alternatively, place this Load Library into the ISPLLIB concatenation.
8. To permit tracing to appear, ensure that a
//SYSTSPRT DD TERM=TS
statement is in the TSO Procedure (or dynamically issue a TSO ALLOCATE FI(SYSTSPRT) DA(*)
command).

9. MA18TEST.JCL must be transferred to the destination TSO system as an 80-byte blocksize, ASCII file. Use
one of the following methods to accomplish this:

   • Use the Communications Manager/2 SEND command below to send the file to TSO

   send ma18test.jcl A:ma18jcl recfm(f) blksize(80) crlf
   where A is the TSO session ID.

   • To send it via ftp ensure the ASCII option is set then use the following commands:

   site fixrecfm 80

   put ma18test.jcl ma18jcl

   • With Personal Communications, use the “Send Files to Host” option under the Transfer menu item to
   transmit to TSO

   PC File ma18test.jcl
   Host File ma18jcl
   Transfer Type textf

   The Transfer type of textf may need to be correctly setup. To do this, use the “Setup.Define Transfer
   Types” option under the Transfer menu item and create the textf type with the Ascii and CRLF
   checkboxes selected, the Append Checkbox unselected, the Fixed radio button selected and the LRECL set
to 80

10. Customise and run this JCL to validate the interface (by adding a Job Card, changing the DSN on the SYSLIB
statement to the above library and using the correct Queue Manager)

--- Note ---

The MVS file names have been described without any qualifiers. Please use whatever conventions are suitable
for your installation.
Installing the interface

To use the Rexx MQ function within a Rexx Exec, nothing special has to be done to make it known to Rexx (there is no equivalent in Rexx/MVS to the OS/2 Rexx RxFuncAdd call).

Please note these operational characteristics:

- There is a reserved variable name RXMQVG which is used by the interface to save information across calls. Therefore,
  - when using internal Rexx Procedures, you must expose RXMQVG in the procedure statement
  - Rexx External procedures are not supported by the interface (but you could arrange to provide the callers RXMQVG as part of the parameters and set RXMQVG in this external procedure).

- Enabling the trace will send non-printable characters to a TSO Screen, resulting in lots of repainting due to error recovery.

- Use of RXMQVC (from SupportPac MA19) conflicts with use of RXMQV; so RXMQVC cannot be used whilst RXMQV is connected to a Queue Manager.

- This SupportPac is not supported in the CICS/Rexx environment (as it interferes with native CICS/MQSeries operations), nor within the Rexx environment for Netview for MVS (SupportPac MA1D provides support within Netview's Rexx).
Chapter 3. TSO Support

TSO Batch Support

The RXMQV interface will run in Batch mode via **IKJEFT01**. The **MA18TEST JCL** file uses the following arrangement for Batch usage of a Rexx Exec.

```mips
//LIB EXEC PGM=IEBGENER
/*
  Create the exec library
/*
//SYSUT2 DD DSN=\&LIB(SILLY),DISP=(NEW,PASS),
  UNIT=SYSDA,
  SPACE=(CYL,(1,1,10)),
  DCB=(DSORG=PS,RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSPRINT DD DUMMY
//SYSPRINT DD DUMMY
//SYSIN DD DUMMY
//SYSUT1 DD DATA,DLM='##'
/* A Silly Exec */
say 'RAH is very very very silly'
exit 0
/* End of SILLY exec */
##
/*
//RUN EXEC PGM=IKJEFT01
//SYSPROC DD DSN=\&LIB,DISP=SHR
//STEPLIB DD DSN=RHARRI1.MAI8LOAD,DISP=SHR
//SYSPRINT DD SYSPRINT=*
//SYSPRINT DD SYSPRINT=* 
//SYSTSPRT DD SYSPRINT=*,DCB=(RECFM=F,LRECL=132,BLKSIZE=132)
//SYSTSPRT DD *
SILLY
/*
```

Figure 1. TSO Batch JCL
**TSO Native Support**

When running within a native TSO Exec (ie: one issued outside of the ISPF environment, or via the TSO command within ISPF), the Rexx processor is attached to TSO as a separate TCB. Consequently, the connection to the Queue Manager will only last throughout the lifetime of the Exec. If processing is interrupted via PA1, then the Rexx processor TCB is terminated, and so the MQ Step termination routines will be driven to terminate all extant accesses.

**TSO Split Screen Support**

When running within a TSO Split Screen, the Rexx processor is attached to each part as a separate TCB. Consequently, the same considerations apply as in “TSO Native Support,” so the Queue Manager connection, and the Rexx Variables cannot be shared between the two halves.

**TSO ISPF Support**

As RXMQV operates within the Rexx environment, all the Rexx variables used are available for use within ISPF in the normal fashion. Figure 2 on page 7 shows an ISPF exec using RXMQV to display the Current Number of messages in a Queue via the panel shown in Figure 3 on page 8. If everything works, then Figure 4 on page 9 is displayed, or else something like Figure 5 on page 9.

This exec also shows how RXMQVG has to be exposed in an internal procedure, and how RXMQV('CONS') is used to establish the MQ literals in such a circumstance.

When running an ISPF EXEC, note that the Rexx processor TCB is not detached until the outer most exec has ended. Consequently, the MQ connection to the QM will last over Panel Display operations if not disconnected (you can show this effect in Figure 2 on page 7 by commenting out the rcc6/7/8 RXMQV operations, whereby rcc2 will appear as “-4 0 0 RXMQVCNNN QM already supplied” and rcc3 works as normal).
/* REXX **************************************************************************/

"ISPORY"
lastrc = rc
if (lastrc = 20) then do
    address TSO 'ISPSTART CMD(MA18T1) NEWAPPL(MA18)'
    exit 0
end

address ispexec

qm = '' ; qn = '' ; numm = ''
rcc1 = '' ; rcc2 = '' ; rcc3 = '' ; rcc4 = ''
rcc5 = '' ; rcc6 = '' ; rcc7 = '' ; rcc8 = ''

"VPUT (qm qn numm rcc1 rcc2 rcc3 rcc4 rcc5 rcc6 rcc7 rcc8) SHARED"

Do mainloop = 1 by 1

"DISPLAY PANEL("MA18P1")"
lastrc = rc

if ( lastrc <> 0 ) then leave mainloop

"VPUT (qm qn numm rcc1 rcc2 rcc3 rcc4 rcc5 rcc6 rcc7 rcc8) SHARED"

rcc1 = '' ; rcc2 = '' ; rcc3 = '' ; rcc4 = ''
rcc5 = '' ; rcc6 = '' ; rcc7 = '' ; rcc8 = ''

drop ood. ; drop hqn ; drop numm

rcc1 = RXMQV('INIT')
rcc2 = RXMQV('CONN', qm )
rcc3 = RXMQV('OPEN', qn, MQOO_INQUIRE, 'hqn', 'ood.' )
call doing
rcc6 = RXMQV('CLOSE', hqn, MQCO_NONE )
rcc7 = RXMQV('DISC')
rcc8 = RXMQV('TERM')
end

exit 0

doinq: procedure expose RXMQVG hqn rcc4 rcc5 numm
       rcc4 = RXMQV('CONS')
       rcc5 = RXMQV('INOQ', hqn, MQIA_CURRENT_Q_DEPTH, 'numm' )
       return
Figure 3. ISPF Panel (MA18P1)
MA18P1 Queue Display

QM VRH1
QN SYSTEM.ADMIN.QMGR.EVENT
Current Number of Messages 10

rcc1 0 0 0 RXMQVINIT OK Written by Robert Harris, Version 2.0 Copy
rcc2 0 0 0 RXMQVCONN OK
rcc3 0 0 0 RXMQVOPEN OK
rcc4 0 0 0 RXMQVCONS OK
rcc5 0 0 0 RXMQVQINQ OK
rcc6 0 0 0 RXMQVCLOSE OK
rcc7 0 0 0 RXMQVDISC OK
rcc8 0 0 0 RXMQVTERM OK Written by Robert Harris. Rexx MQ Functions ar

Figure 4. ISPF Panel (success)

MA18P1 Queue Display

QM VRH1
QN SYSTEM.ADMIN.QMGR.EVENT
Current Number of Messages

rcc1 0 0 0 RXMQVINIT OK Written by Robert Harris, Version 2.0 Copy
rcc2 0 0 0 RXMQVCONN OK
rcc3 2085 2 2085 RXMQVOPEN FAILED
rcc4 0 0 0 RXMQVCONS OK
rcc5 -8 0 0 RXMQVQINQ Handle out of range
rcc6 -7 0 0 RXMQVCLOSE Handle out of range
rcc7 0 0 0 RXMQVDISC OK
rcc8 0 0 0 RXMQVTERM OK Written by Robert Harris. Rexx MQ Functions ar

Figure 5. ISPF Panel (failure)
Chapter 4. Interface Design Philosophy

The Rexx MQ Interface API differs from that defined in the APR. This is because the call-type of API is not suitable for the Rexx environment. This has been replaced with a set of verbs that use Rexx Stem variables to contain the relevant information.

The opportunity has also be taken to remove some parameters due to the restriction that a single MVS/ESA thread (Exec in the Rexx environment) can only communicate with a single Queue Manager. Additionally, in order to simplify coding, Input and Output versions of object are provided (this saves deleting and rebuilding things like Message descriptors which are updated by a MQ Verb).

As part of the initialisation call, all the non-string MQ Constants (as described in Chapter 1.5 of the APR) are defined to the Rexx workspace. Thus, you will be able to code options according to the descriptions in the APR. However, these values are not protected against change, so you should avoid using your own variables starting with MQ.
Chapter 5. General points

Operations

All the functions in this Rexx/MQ/MVS interface are accessed via the RXMQV call, with the first parameter indicating the function to be run.

```
rcc = RXMQV('function', p1, p2, p3 ....)
```

RXMQV returns a character string to show the results of the function being run.

This is a different interface than that provided within SupportPac MA31 for Rexx/MQ/OS2, as MVS does not permit module names longer than 8 bytes. However, exactly the same function is provided within the OS/2 and the MVS environments (apart from Get.Signal which is MVS specific).

Return Codes

The RXMQV function returns a standard Rexx Return string. This is structured so that the numeric Return Code (which may be negative) is obtained by a word(RCC,1) call.

The Return Code for an operation can be negative to show that RXMQV has detected the error, otherwise it will be the MQ Completion Code (not the uninformative Reason Code).

The Return String is in text format as follows:

- Word 1: Return Code
- Word 2: MQ Completion Code (or 0 if MQ not done)
- Word 3: MQ Reason Code (or 0 if MQ not done)
- Word 4: RXMQV function being run
- Word >: OK or an helpful error message
Last Operation

In addition, the current (ie: the settings last set) values are available in these variables:

- `RXMQV.LASTRC`: current operation Return Code
- `RXMQV.LASTCC`: current operation MQ Completion Code
- `RXMQV.LASTAC`: current operation MQ Reason Code
- `RXMQV.LASTOP`: current operation RXMQV function
- `RXMQV.LASTMSG`: current operation Return String

Return Code naming

A set of variables called `RXMQV.RCMAP.nn` are also placed in the workspace, where `nn` is the MQ Reason Code. These variables can be used to turn a return code number into the defining string.

Thus:

```plaintext
rcc = '2048 2 2048 RXMQVPUT ERROR'
interpret 'fcs = RXMQV.RCMAP.'word(rcc,1)
/* fcs = MQRC_PERSISTENT_NOT_ALLOWED */
```

Message Lengths

When a `MQGET` is performed, if the buffer size is too small for the message, then the returned message length is the truncated length of the message, not the bigger size which would not fit in the buffer (see `Datalength` for `MQGET` in the APR).

Consequently, if you specify a too small a message length, and do not take any notice of the return code indicating truncation, then the length of the message in stem.0 will be the same as the message in stem.1 (as usual). This may result in a mysterious loss of data in the message.

This processing is different from that provided in the MA31 SupportPac which is the Rexx/MQ/OS2 interface. In the OS2 environment the length in .0 is the length of the message that would have been returned if the buffer was big enough, with the length of the data in .1 being truncated value.
**Internal procedures**

The variable RXMQVG contains information that is saved across execution of RXMQV. Consequently, it must be available throughout the Exec which uses the interface *(do not alter it!)*. So, when using internal procedures, you should EXPOSE RXMQVG as part of the procedure definition. Thus:

```rexx
internal_proc (a, b ,c)

internal_proc: procedure expose RXMQVG

rcc = RXMQV('CONS')
```

When in an internal procedure, all the RXMQV variables are hidden by Rexx. You can create new mappings by using the CONS function (see “Setting Literals” on page 27).

Rexx External procedures are not supported by the Interface (as RXMQVG cannot be directly exposed into the latters Rexx Workspace). However, *at your own risk*, you could manually provide RXMQVG as part of the parameters and set up RXMQVG in the external procedures Rexx Workspace.

**Header and Event processing**

Operations HXT and EVENT will take messages and split them up into the contained components. These exploded components may clash with those for the Message Descriptor (or other like things). Therefore, use different stem names to avoid this possibility.
ZLIST

One of the problems with REXX Stem. variables is that it is difficult to know what components (things after the .) are associated with the stem. You have to know which ones *might* be around, and then test with something like:

```rexx
if ( stem.comp1 <> 'STEM.COMP1' ) then say 'comp1 =/'stem.comp1'/'
if ( stem.comp2 <> 'STEM.COMP2' ) then say 'comp2 =/'stem.comp2'/'
```

To get around this problem, the output descriptors will contain a component called ZLIST. ZLIST will contain a list of words, each word a component name which is attached to the stem variable. You can then use the Rexx WORDS (to get the number of elements) and WORD (to extract the component name) functions to manipulate the stem. variable. ZLIST does not contain itself (ie: ZLIST is not within stem.ZLIST).

The presence of an item in ZLIST implies that the relevant Stem.Component is defined as a Rexx Variable. However, the contents may be null (a length of zero or set to '') depending upon what the underlying MQ object contains.

This facility is not of much use for the OPEN, GET and PUT calls (wherein ZLIST is provided for the Output Object descriptor, Output Message Descriptor, Output Get Message Options and Output Put Message options) as the contents of the Output Stem. variable is of fixed format. However, it can be used to display the stem. variable and can also be useful in copying operations.

For HXT and EVENT processing, ZLIST is of variable format, containing things relevant to the Message or Event being processed. ZLIST for HXT processing contains components 0 and 1 (the original message) as well as NAME and TYPE. For EVENT processing, NAME, TYPE and REA are always present; the rest of the list will depend upon the event being processed (with CED.0 and CED.n if present).

For example to display an Object descriptor:

```rexx
drop iod. ; drop ood.
iod.on = 'N1'
iod.ot = MQOT_Q
rcc = RXMQV('OPEN', 'iod.', mqoo_inquire, 'h1', 'ood.')
say 'RC=' rcc 'H=' h1
do j=1 to words(oood.zlist)
  k = word(oood.zlist,j)
say k '/'ood.k'/'
end
```
ZLIST can be used for Event processing:

```rnc
drop bm. ; drop ed.
rc = RXMQV('BROWSE', he, 'bm.')
say 'Browse RC=' rc

c = RXMQV('EVENT', 'bm.', 'ed.')
say 'Event RC=' c
say '.zlist /'ed.zlist'/'

/* Protect against bad function by being very cautious! */
if ( (ed.zlist <> 'ED.ZLIST') & (words(ed.zlist) <> 0) ) then ,
do j=1 to words(ed.zlist)
k = word(ed.zlist,j)
say 'ed.'k' /'ed.k'/'
end

/* I'm only interested in Unknown Object Events */
/* */
/* However, do not want to access undefined */
/* components. */
/* */
/* Note the '' ' ' around the Event variable to */
/* preserve the FULL length of the data */
/* with blank padding. It would be */
/* better to then do */
/* */
/* interpret 'u'uv' = strip(u'uv','B''') */
/* */
/* to get rid of these blanks */
/* */
/* */
if ( ed.name = 'LLUON' ) then do
uvars = 'QM QN AT AN QQM PN'
uqm = ''; uqn = ''; uat = ''; uan = ''; uoqm = ''; upn = '');
do i=1 to words(uvars)
  uv = word(uvars,i)
  if ( wordpos(uv,ed.zlist) <> 0 ) then ,
    interpret 'u'uv' = ''ed.'uv'''
end

/* So, if PN is not set within the Event */
/* (it's an optional parameter), it will */
/* not be accessed. */
```

Figure 6. ZLIST and Event processing
Stem Variables

As described in “Handling MQ Descriptors” on page 18, Stem variables are extensively used in this interface. A Stem variable is one that has various bits separated by dots (such as `a.b.c`). Everything after the first dot is called a component; so in the above example, `a` is the Stem variable, and `b` & `c` are components.

You should be aware that you can cause conflicts if you use Rexx variables with the same name as components. This is because Rexx will substitute the values of component names as if they were variables before usage.

```
| a.1 = 15   |
| a.2 = 3    |
| b = 2      |
| say a..b   /* -> 3 due to substitution */ |
```

This can cause problems if you use any of the returned component names from this utility as native variables - because you will get an 'unknown' setting due to the substitution.

```
| qn = 'RAH'  |
| ud = 'some userish data' |
| rcc = RXMQV('...', ...data_which_will_set_.qn=A , 'out.') |
| say out.qn   /* tries to resolve out.RAH */ |
| /* -> A */ |
| /* as the utility does the substitution */ |
| say out.ud   /* tries to resolve out.some userish data */ |
| /* -> a Rexx error due to invalid var name */ |
```

Unless you are deliberately doing this sort of processing, I suggest you avoid using variables which are returned as components.
Trace

Tracing is provided by settings in the RXMQVTRACE Rexx variable. Note that the tracing is sent to the currently open STDOUT stream, and some of the settings can produce a lot of output. The settings are:

- **CONN**: `mqconn`
- **DISC**: `mqdisc`
- **OPEN**: `mqopen`
- **CLOSE**: `mqopen`
- **GET**: `mqget`
- **PUT**: `mqput`
- **INQ**: `mqinq`
- **SET**: `mqset`
- **CMIT**: `mqcmiit`
- **BACK**: `mqback`

**QSI**: Query Signal extension

**BRO**: Browse extension

**HXT**: Header extraction extension

**EVENT**: Event expansion extension

**TM**: Trigger message extension

**MMD**: Rexx stem var -> MQMD
**MOD**: Rexx stem var -> MQOD
**MPO**: Rexx stem var -> MQPMO
**MGO**: Rexx stem var -> MQGMO
**BMD**: MQMD -> Rexx stem var
**BOD**: MQOD -> Rexx stem var
**BPO**: MQPMO -> Rexx stem var
**BGO**: MQGMO -> Rexx stem var

**GV**: Obtaining a Rexx Variable

**SV**: Setting a Rexx Variable

**SK**: Return Code processing

**TR**: Thread based processing

**COMMON**: common RXMQV processing
**GG**: Globals processing (RXMQVG)

**INIT**: Initialisation processing

**CONS**: Literal processing

**TERM**: Deregistration processing

* Trace everything!!!

So, to trace Gets and Puts, one would code

```
RXMQVTRACE = 'PUT GET'
```
Chapter 6. Handling MQ Descriptors

The API defined for MQ/MVS in the Application Reference Manual uses various structures to pass information both into and out of the Queue Manager. These structures are:

**MQOD**
The Object Descriptor, used by the MQOPEN call to specify the MQ Object being processed, and return various attributes of the accessed item

**MQMD**
The Message Descriptor, used by MQGET and MQPUT verbs to specify (for the MQPUT) attributes for the emplaced message, and return these attributes (for the MQGET)

**MQGMO**
This structure controls the operation of the MQGET verb

**MQPMO**
This structure controls the operation of the MQPUT verb

These structures are input/output for the MQ Verbs.

In order to supply these structures to the underlying MQ Verbs within this Rexx/MQ Interface, Rexx stem variables are used. In order to reduce complexity, and enhance the ease of usage of the interface, separate Stem variables are used for input and output. This reduces the complexity of the Rexx code, as the input Stem variable may be reused (so removing all the tedious removal of redundant information required by the MQ API).

This approach allows, for simple applications, the initial setup of the stem variables representing the requested options; these are then repeatedly reused, the output versions simply not being accessed.

The structure of the stem variables is fixed. By this I mean that the name of the stem variable (before the dot) can be chosen by the caller, whilst the latter part (after the dot) is fixed by the interface. The things after the dot are called the Components of the stem variable.

The normal Rexx rules apply to these Stem variables, in particular they are case invariant (Rexx treats all variables as being of Upper case), and substitution may occur within the name. Therefore, take care to avoid using variables that could clash with the naming conventions of these interface requirements (see “Stem Variables” on page 16).

When supplying these stem variables to the interface, you have to pass the name of the stem variable (including the trailing dot). Thus, one would normally specify this information as a literal (RXMQV( ..., ‘AGMO.’, ... )).

However, you are at liberty to use the normal Rexx substitutions on an interface call (so Z = ‘AGMO.’ ; RXMQV(...,..., Z ) is correct), and even abandon the stem variable convention completely (but this will lead to unwieldy execs). This abandonment, however, does not apply to one of the RXMQV(OPEN) parameters.

When you build the stem variable, component abbreviations for the full name of the relevant structure’s field is used (eg: CCID for CodedCharSetId) to improve legibility of the Exec. You only specify those fields of interest - the others should be omitted. The omitted components will default to the relevant settings as defined in the APR (a value or nulls).

However, although some fields of the descriptors are only used for input or output, this interface will utilise all of the information within the Stem variable - even if it is not used by the underlying MQ code (such as the Destination Count fields within the PMO descriptor - these are not used by the underlying MQ code, but this interface will process them if so supplied).

When the interface returns a structure to the exec, in the named Stem variable, all the components (fields) will be placed within the stem.structure.

The actual settings for these component variables are documented in the MQ/MVS APR to which you should refer. As the interface places within the Rexx workspace all MQ numeric values, the stem components can be set using
the normal MQ conventions (eg: stem.PER = MQMD_NOT_PERSISTENT). The interface does not check that the values are relevant for the field.

In the case of text fields, the interface will truncate supplied data that is too long for the MQ structure without notification. Fields that are to be null should not be supplied to the interface, and are returned as nulls ("").

Actual message data to/from the Queue Manager is passed via the usual Rexx convention (see “Message Lengths” on page 12 for a warning about truncation):

- **stem.0** contains the length of the data
- **stem.1** contains the message data

Functions **HXT** and **EVENT** will take messages and split them up into the contained components. These exploded components may clash with those for the Message Descriptor (or other like things). Therefore, use different stem names to avoid this possibility.

**ZLIST** processing (see “ZLIST” on page 14) is available for the **Output** Stems representing a MQOD, MQMD, MQGMO or MQPMO. If present within an **Input** Stem. variable, **ZLIST** is ignored.
The Object Descriptor

The Object descriptor is solely used by the OPEN call (the MQOPEN verb).

If you are accessing a Queue, then the short cut form of RXMQV('OPEN') can be used, and so the Object Descriptor is only of interest upon completion of the call. The only interesting part of the OD in this case is the name of the 'real' queue generated when a Model queue is opened.

<table>
<thead>
<tr>
<th>Stem. Component</th>
<th>MQOD Structure name</th>
<th>Input, Output or Both</th>
<th>Number or Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>.OT</td>
<td>ObjectType</td>
<td>I</td>
<td>N</td>
</tr>
<tr>
<td>.ON</td>
<td>ObjectName</td>
<td>B</td>
<td>T</td>
</tr>
<tr>
<td>.OQM</td>
<td>ObjectQMgrName</td>
<td>B</td>
<td>T</td>
</tr>
<tr>
<td>.DQN</td>
<td>DynamicQueue</td>
<td>I</td>
<td>T</td>
</tr>
<tr>
<td>.AUID</td>
<td>AlternateUserid</td>
<td>I</td>
<td>T</td>
</tr>
</tbody>
</table>

Note:
- Input, Output and Both show how the field is used
- Number or Text shows the type of the field (and how it is assembled)
- ZLIST is set to 'AUID DQN ON OQM OT' for Output operations
The Message Descriptor

The Message Descriptor details the type of the message being processed. It also has a meaning where messages are obtained from a queue - whereat it is used to select messages for obtention from the queue. The interface does not check that combinations of components are valid.

As separate versions of a Message Descriptor are required by the interface for Input and Output on each call, the input MD can be reused for subsequent accesses. Components omitted will take the defaults as defined in the APR.

Table 2. Message Descriptor Mappings

<table>
<thead>
<tr>
<th>Stem. Component</th>
<th>MQMD Structure name</th>
<th>Input, Output or Both</th>
<th>Number or Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>REP</td>
<td>Report</td>
<td>O / I</td>
<td>N</td>
</tr>
<tr>
<td>MSG</td>
<td>MsgType</td>
<td>O / I</td>
<td>N</td>
</tr>
<tr>
<td>EXP</td>
<td>Expiry</td>
<td>O / I</td>
<td>N</td>
</tr>
<tr>
<td>FBK</td>
<td>Feedback</td>
<td>O / I</td>
<td>N</td>
</tr>
<tr>
<td>ENC</td>
<td>Encoding</td>
<td>O / I</td>
<td>N</td>
</tr>
<tr>
<td>CCSI</td>
<td>CodedCharSetId</td>
<td>O / I</td>
<td>N</td>
</tr>
<tr>
<td>FORM</td>
<td>Format</td>
<td>O / I</td>
<td>N</td>
</tr>
<tr>
<td>PRI</td>
<td>Priority</td>
<td>O / I</td>
<td>N</td>
</tr>
<tr>
<td>PER</td>
<td>Persistence</td>
<td>O / I</td>
<td>N</td>
</tr>
<tr>
<td>MSGID</td>
<td>MsgId</td>
<td>B / B</td>
<td>T</td>
</tr>
<tr>
<td>CID</td>
<td>CorrelId</td>
<td>B / I</td>
<td>T</td>
</tr>
<tr>
<td>BC</td>
<td>BackoutCount</td>
<td>B / -</td>
<td>N</td>
</tr>
<tr>
<td>RTOQ</td>
<td>ReplyToQ</td>
<td>O / I</td>
<td>T</td>
</tr>
<tr>
<td>RTOQM</td>
<td>ReplyToQMgr</td>
<td>O / I</td>
<td>T</td>
</tr>
<tr>
<td>UID</td>
<td>UserIdentifier</td>
<td>O / B</td>
<td>T</td>
</tr>
<tr>
<td>AT</td>
<td>AccountingToken</td>
<td>O / B</td>
<td>T</td>
</tr>
<tr>
<td>AID</td>
<td>ApplyIdentityData</td>
<td>O / B</td>
<td>T</td>
</tr>
<tr>
<td>PAT</td>
<td>PutApplType</td>
<td>O / B</td>
<td>T</td>
</tr>
<tr>
<td>PAN</td>
<td>PutApplName</td>
<td>O / B</td>
<td>T</td>
</tr>
<tr>
<td>PD</td>
<td>PutDate</td>
<td>O / B</td>
<td>T</td>
</tr>
<tr>
<td>PT</td>
<td>PutTime</td>
<td>O / B</td>
<td>T</td>
</tr>
<tr>
<td>AOD</td>
<td>ApplOriginData</td>
<td>O / B</td>
<td>T</td>
</tr>
</tbody>
</table>

Note:
- Input, Output and Both show how the field is used (- is unused)
- Number or Text shows the type of the field (and how it is assembled)
- ZLIST is set to 'AID AOD AT BC CID CCSI ENC EXP FBK FORM MSG MSGID PAN PAT PD PER PRI PT REP RTOQ RTOQM UID' for Output operations
The Get Message Option Structure

The Get Message Option Structure requests what message is to be obtained from a queue via the *MQGET* verb. As it is updated by this operation, RXMQV('GET') uses an Input and Output Stem variable to hold this information.

<table>
<thead>
<tr>
<th>Stem. Component</th>
<th>MQGMO Structure name</th>
<th>Input, Output or Both</th>
<th>Number or Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>.OPT</td>
<td>Options</td>
<td>I</td>
<td>N</td>
</tr>
<tr>
<td>.WAIT</td>
<td>WaitInterval</td>
<td>I</td>
<td>N</td>
</tr>
<tr>
<td>.RQN</td>
<td>ResolvedQueueName</td>
<td>O</td>
<td>T</td>
</tr>
</tbody>
</table>

**Note:**
- Input, Output and Both show how the field is used
- Number or Text shows the type of the field (and how it is assembled)
- ZLIST is set to 'OPT RQN WAIT' for Output operations
The Put Message Options Structure

The Put Message Option Structure requests what type of message is to be placed in a queue via the `MQPUT` verb. As it is updated by this operation, RXMQV('PUT') uses an Input and Output Stem variable to hold this information.

<table>
<thead>
<tr>
<th>Stem Component</th>
<th>MQPMO Structure name</th>
<th>Input, Output or Both</th>
<th>Number or Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>.OPT</td>
<td>Options</td>
<td>I</td>
<td>N</td>
</tr>
<tr>
<td>.TIME</td>
<td>Timeout</td>
<td>I</td>
<td>N</td>
</tr>
<tr>
<td>.CON</td>
<td>Context</td>
<td>I</td>
<td>T</td>
</tr>
<tr>
<td>.KDC</td>
<td>KnownDestCount</td>
<td>-</td>
<td>N</td>
</tr>
<tr>
<td>.UDC</td>
<td>UnKnownDestCount</td>
<td>-</td>
<td>N</td>
</tr>
<tr>
<td>.IDC</td>
<td>InvalidDestCount</td>
<td>-</td>
<td>N</td>
</tr>
<tr>
<td>.RQN</td>
<td>ResolvedQueueName</td>
<td>O</td>
<td>T</td>
</tr>
<tr>
<td>.RQMN</td>
<td>ResolvedQueueMgrName</td>
<td>O</td>
<td>T</td>
</tr>
</tbody>
</table>

Note:
- Input, Output and Both show how the field is used (- is unused)
- Number or Text shows the type of the field (and how it is assembled)
- ZLIST is set to 'CON IDC KDC OPT RQMN RQN TIME UDC' for Output operations
- The CONTEXT setting is the handle returned by RXMQV (it is converted internally to the correct MQ Handle)
Chapter 7. The Interface

The functions provided by this Rexx/MQ interface roughly follow those provided by the underlying MQ API, with some extensions and the calls required by Rexx to initialise the interface.

All the parameters specified for a RXMQV call are required; none can be omitted. The first parameter is always the function being run.

When the interface detects an error, a negative return code will be provided as the first word in the return string. These are documented with the associated message under the individual calls.

The Initialisation and Termination functions:

- **Initialisation**
  - “Initialisation” on page 26
- **Setting Literals**
  - “Setting Literals” on page 27
- **Termination**
  - “Termination” on page 28

The Standard MQ functions:

- **MQBACK**
  - “RXMQVBACK” on page 44
- **MQCLOSE**
  - “RXMQVCLOSE” on page 35
- **MQCMIT**
  - “RXMQVCMIT” on page 43
- **MQCONN**
  - “RXMQVCONN” on page 29
- **MQDISC**
  - “RXMQVDISC” on page 31
- **MQGET**
  - “RXMQVGET” on page 45
- **MQINQ**
  - “RXMQVINQ” on page 37
- **MQOPEN**
  - “RXMQVOPEN” on page 32
- **MQPUT**
  - “RXMQVPUT” on page 48
- **MQSET**
  - “RXMQVSET” on page 40

The Extension functions:

- **Query Signal**
  - “RXMQVQ SIGNAL” on page 51
- **Browse**
  - “RXMQVBROWSE” on page 54
- **Header Extraction**
  - “RXMQVHXT” on page 56
- **Event Determination**
  - “RXMQVEVENT” on page 61
- **Trigger Extraction**
  - “RXMQVTM” on page 74
Common Return Codes

These Return Codes can be commonly returned by RXMQV:

0 0 0 RXMQV Nothing happened

Explanation: Although all the supplied parameters were alright, in combination they result in a NOOP. This is OK as far as the RXMQV interface is concerned. This can commonly occur when RXMQV('DISC') is issued without any QM being currently connected.

-99 0 0 RXMQV Incorrect number of Parms supplied

Explanation: You must specify at least one parm to RXMQV, the function to be run.

-98 0 0 RXMQV Globals not found (RXMQVG exposed?)

Explanation: The reserved Rexx Variable RXMQVG which contains information that lasts across the individual RXMQV call was not located. In an internal procedure, this probably means that you have not exposed RXMQVG on the procedure statement (see “Internal procedures” on page 13).

-97 0 0 RXMQV Globals look terrible (RXMQVG exposed)!!

Explanation: The reserved Rexx Variable RXMQVG which contains information that lasts across the individual RXMQV call was located, but it did not contain the required information. You have probably altered it in a naughty way. In an unsupported external procedure, this probably means that you have not have not RXMQVG correctly (see “Internal procedures” on page 13).

-96 0 0 RXMQV Unknown request

Explanation: The first parameter to RXMQV is the function to run, and this specified an unknown function.

-99 0 0 RXMQVxxxx UNKNOWN FAILURE

Explanation: Some unknown error has occurred in function RXMQV('xxxx').
**Initialisation**

**Description**

This function initialises the interface, defines all the functions for Rexx usage, and places all the MQ_ non-string constants into the Rexx workspace. These mappings are listed in the Appendix.

The RIXMQV('INIT') call needs to be done **once** with the Exec.

**Parameters**

None

**Call**

```plaintext
rcc = RIXMQV('INIT')
```

**Additional Interface Return Codes and Messages**

None

**Example**

```plaintext
rcc = RIXMQV('INIT')
```
Setting Literals

Description

This function places all the MQ_non-string constants into the Rexx workspace. This is only useful if not executing any 'proper' MQ functions, but only the MQ_mappings are required (such as when executing within an internal procedure). Setting the literals is also useful when operating within internal functions. These mappings are listed in the Appendix. This function can be called when there is no Queue Manager activity.

Parameters

None

Call

```
rc = RXMQV('CONS')
```

Additional Interface Return Codes and Messages

None

Example

```
rcc = RXMQV('CONS')
```
Termination

Description

This function simply removes the information which is saved across RXMQV executions (the things in the reserved Rexx Variable RXMQVG). It does not initiate MQ Termination processing. If a prior RXMQV('CLOSE') or a RXMQV('DISC') have not been done, then the usual End-of-Step MQ function will (eventually) stop access to the Queue Manager. If this call is omitted, there will be a small memory leakage.

The MQ_ definitions are left in the Rexx workspace, so that new commands can be composed using the 'real' notations.

Parameters

None

Call

```
rcc = RXMQV('TERM')
```

Additional Interface Return Codes and Messages

None

Example

```
rcc = RXMQV('TERM')
```
RXMQVCONN

Description

This function connects the Rexx Interface to the Queue Manager. Note that there is a MQ/MVS restriction such that only one Queue Manager can be contacted from an MVS/ESA TCB (the Rexx processor, in this case).

This call has to be made after the RXMQV('INIT') call, and only be made once (unless a RXMQV('DISC') is made).

Owing to the above restriction, the Queue Manager Handle returned by the use of MQCONN within RXMQV is not a useful thing, and so is not returned to the Rexx Exec.

Parameters

1. The name of the Queue Manager to connect to (Input only).

Call

```
rcc = RXMQV('CONN', QM )
```
Additional Interface Return Codes and Messages

-1 0 0 RXMQVCONN Bad number of parms
Explanation: You must specify only one parameter to RXMQV('CONN'); this parameter being the name of the Queue Manager to contact.

-2 0 0 RXMQVCONN Supplied QM name is too short
Explanation: The Queue Manager Name supplied was of Zero Length (ie: ").

-3 0 0 RXMQVCONN Supplied QM name too long
Explanation: The maximum length of a Valid Queue Manager Name is 48 bytes.

-4 0 0 RXMQVCONN QM already supplied
Explanation: The QM name has already been supplied to RXMQVG (an attempt to contact more than one QM is invalid).

-5 0 0 RXMQVCONN QM already connected
Explanation: A QM is already connected to RXMQV.

-6 0 0 RXMQVCONN Thread already connected
Explanation: The current TCB is already connected to a QM.

-7 0 0 RXMQVCONN Thread already connected to QM
Explanation: The current TCB is already connected to a QM.

Example

```
rcc = RXMQV('CONN', 'RAH1')
```

This call will contact the local Queue Manager called RAH1. If this Queue Manager is not defined, or not running, then the call will fail.
RXMQVDISC

Description

This function disconnects (MQDISC) from the currently connected Queue Manager. As an extension to the function, the interface will issue a MQCLOSE(none) for any still open queue accessed via the interface (this is to cope with Rexx Tracing, and so give the user a simple way of 'gracefully' exiting when in test mode).

Parameters

None.

Call

```rabex
rcc = RXMQV('DISC')
```

Additional Interface Return Codes and Messages

None

Example

```rabex
rcc = RXMQV('DISC')
```

This call will disconnect from the currently accessed Queue Manager, doing a MQCLOSE(None) on any Queues still open at this point.
RXMQVOPEN

Description

This verb provides access to a MQ Object via a MQOPEN call. Upto 100 Objects can be accessed via this interface in any one TCB. Although one will normally be accessing a Queue, any of the allowed MQ objects can be accessed.

Parameters

1. The name of a Stem variable (including the dot) specifying the Object Descriptor for the MQ Object to access. This is an input only field. The format of this Stem variable is described in “The Object Descriptor” on page 20.

   If the name given does not end in a dot, then the data is taken to be the name of a Queue (or Model Queue) to access. This short cut removes the requirement to fully format up a stem variable for 'normal' Queue access; but note that you supply the name of the Queue, not the name of the variable containing the name of the Queue.

2. The MQOPEN Options (as described in the APR). This is an input only field, and should resolve into a number (not the name of a field containing the Options).

3. The name of a variable to contain a handle for the MQ Object being accessed. This is an output field, and should be the name of the field to receive the handle.

   The handle returned is not the handle returned by the underlying MQOPEN verb; this latter value is not accessible outside of the interface. This handle must be quoted on all subsequent accesses to the Object.

4. The name of a Stem variable (including the dot) into which is placed the Object Descriptor returned by the underlying MQOPEN verb. This is an output only field.

   The format of this Stem variable is described in “The Object Descriptor” on page 20; ZLIST processing is provided.

Call

```rx
rcc = RXMQV('OPEN', 'Stem.Input.OD.', OpenOptions, 'VarHandle', 'Stem.Output.OD.')
```

or

```rx
rcc = RXMQV('OPEN', QueueName , OpenOptions, 'VarHandle', 'Stem.Output.OD.')
```
Additional Interface Return Codes and Messages

-1 0 0 RXMQVOPEN Bad number ofParms

Explanation: You must specify four parameters to the RXMQV('OPEN') call.

-2 0 0 RXMQVOPEN Input OD Stem. not supplied

Explanation: A null has been supplied for the first parameter, the name of a stem variable for an input Open Descriptor or the name of a Queue to access.

-3 0 0 RXMQVOPEN Input Open Options not supplied

Explanation: No value has been keyed for the second parameter, a number representing the Open Options. To specify No Options, supply a 0.

-4 0 0 RXMQVOPEN Output Handle Var name not supplied

Explanation: No value has been keyed for the third parameter, the name of a variable which will be set to the obtained handle for the accessed MQ Object.

-5 0 0 RXMQVOPEN Output OD Stem. not supplied

Explanation: No value has been keyed for the forth parameter, the name of a stem variable which will be set to the obtained Object Descriptor for the accessed MQ Object.

-6 0 0 RXMQVOPEN Open Options not numeric

Explanation: The value supplied for the second parameter, a number representing the Open Options is not actually numeric. To specify No Options, supply a 0.

-7 0 0 RXMQVOPEN QM not connected

Explanation: The current TCB is not Connected to a Queue Manager

-8 0 0 RXMQVOPEN Thread not connected

Explanation: The current TCB is not Connected to a Queue Manager

-9 0 0 RXMQVOPEN Too many opened Objects

Explanation: The limit of MQ Objects supported by this interface has been reached.
Example

```rexx
opts = MQOO_INQUIRE + MQOO_INPUT_SHARED,
       + MQOO_BROWSE + MQOO_SAVE_ALL_CONTEXT,
       + MQOO_FAIL_IF_QUIESCING
rcc  = RXMQV('OPEN', N1, opts, 'hn1', 'od.' )
```

This call opens the Queue N1 for a Browse access, and permits the inquiry of the queue's attributes. If the open succeeds, then the variable hn1 is set to the handle for subsequent access to N1, and the stem variable od. is set to the contents of the Object Descriptor for N1 (eg: od.ON = 'N1').

```rexx
iod.OF = MQOT_Q
iod.ON = 'N1'

rcc  = RXMQV('OPEN', 'iod.', MQOO_BROWSE+MQOO_INQUIRE, 'hn1', 'ood.' )
```

This example shows how the Queue N1 would be accessed if the full Object Descriptor method is used to specify the MQ Object to be accessed.
RXMQVCLOSE

Description

This verb stops access to a MQ Object, using the underlying MQCLOSE verb.

Parameters

1. The Handle for the object obtained from a prior RXMQV('OPEN') call. This is an input parameter. After this call completes, the handle is no longer valid for use.
2. The Close options. This is an input parameter representing the type of MQCLOSE operation to be performed.

Call

```rexx
crc = RXMQV('CLOSE', handle, CloseOptions )
```
Additional Interface Return Codes and Messages

-1 0 0 RXMQVCLOSE Bad number of Parms
Explanation: You must specify two parameters to the RXMQV('CLOSE') call.

-2 0 0 RXMQVCLOSE Handle not supplied
Explanation: No value has been keyed for the first parameter, the handle representing the MQ object.

-3 0 0 RXMQVCLOSE Close Options not supplied
Explanation: No value has been keyed for the second parameter, a number representing the Close Options. To specify No Options, supply a 0.

-4 0 0 RXMQVCLOSE Close Options not numeric
Explanation: The value supplied for the second parameter, a number representing the Close Options is not actually numeric. To specify No Options, supply a 0.

-5 0 0 RXMQVCLOSE QM not connected
Explanation: The current TCB is not Connected to a Queue Manager

-6 0 0 RXMQVCLOSE Thread not connected
Explanation: The current TCB is not Connected to a Queue Manager

-7 0 0 RXMQVCLOSE Handle out of range
Explanation: The value of the handle supplied is not in the known range for a handle within the interface.

-8 0 0 RXMQVCLOSE Handle invalid
Explanation: The handle specified does not relate to an accessed MQ Object.

Example

```rexx
rcc = RXMQV('CLOSE', hn1, MQCO_NONE )
```

This call closes the object referred to by the handle specified in the hn1 variable, with no special closing actions being requested.
RXMQVINQ

Description

This call will inquire upon a single attribute of a MQ object. This is a difference between this interface and the function of the underlying MQINQ verb.

The relevant data is returned in character format, so numeric attributes need not be converted for Rexx usage. The requested attribute is specified via MQIA_ or MQCA_ variables.

Parameters

1. The Handle for the object obtained from a prior RXMQV('OPEN') call, whereat the object was opened for Inquiry. This is an input parameter.
2. The Attribute Number to be Inquired upon (setting starting with MQIA_ or MQCA_). This is an input parameter.
3. The name of a variable into which will be returned the current setting of the desired attribute. Numeric attributes (like Maximum Message Size) are converted into character settings (so ‘17’ might be returned rather than ‘11’x). This is an output parameter.

Call

```
rcc = RXMQV('INQ', handle, Attribute, 'VarAttributeValue' )
```
Additional Interface Return Codes and Messages

-1 0 0 RXMQVINQ Bad number ofParms
Explanation: You must specify three parameters to the RXMQV(INQ) call.

-2 0 0 RXMQVINQ Handle not supplied
Explanation: No value has been keyed for the first parameter, the handle representing the MQ object.

-3 0 0 RXMQVINQ Attribute not supplied
Explanation: No value has been keyed for the second parameter, a number representing the attribute of the MQ object to be obtained.

-4 0 0 RXMQVINQ Attribute not numeric
Explanation: The value supplied for the second parameter, a number representing representing the attribute of the MQ object to be obtained, is not actually numeric.

-5 0 0 RXMQVINQ Output Variable name not supplied
Explanation: No value has been keyed for the third parameter, the name of a variable to receive the value of the requested attribute.

-6 0 0 RXMQVINQ QM not connected
Explanation: The current TCB is not Connected to a Queue Manager

-7 0 0 RXMQVINQ Thread not connected
Explanation: The current TCB is not Connected to a Queue Manager

-8 0 0 RXMQVINQ Handle out of range
Explanation: The current TCB is not Connected to a Queue Manager

-9 0 0 RXMQVINQ Handle invalid
Explanation: The value of the handle supplied is not in the known range for a handle within the interface.

-10 0 0 RXMQVINQ Unknown Char Attribute
Explanation: The value of the requested attribute was found to be within the range for a Character attribute, but was not defined as a Character attribute.

-11 0 0 RXMQVINQ Attribute out of valid range
Explanation: The value of the attribute under consideration was outside of the ranges defined for Integer and Character attributes.
Example

```
  rcc = RXMQV('INQ', hn1, MQIA_MAX_MSG_LENGTH, 'maxmsg')
  /* maxmsg = 3109856 */
```

This call obtains the current Maximum Message Length attribute for the queue referenced by the handle contained in hn1. In this case, the maxmsg variable is set to 3109856, the value of the desired attribute.
RXMQVSET

Description

This call will set a given attribute of a MQ object. This is a difference between this interface and the underlying MQSET verb, whereat many attributes can be manipulated in a single execution.

The relevant data is specified in character format, so numeric attributes need not be converted for interface usage. The attribute is specified via MQIA_ or MQCA_ variables.

Parameters

1. The Handle for the object obtained from a prior RXMQV('OPEN') call, whereat the object was opened for Setting. This is an input parameter.
2. The Attribute Number to be Set (starting with MQIA_ or MQCA_). This is an input parameter.
3. The value of the attribute which is to be be Set in the MQ Object. Numeric attributes (like Trigger Depth) are specified in the normal Rexx character format (so use '17' rather than '11'x). This is an input parameter.

Call

rcc = RXMQV('SET', handle, Attribute, AttributeSetting )
Additional Interface Return Codes and Messages

-1 0 0 RXMQVSET Bad number of Parms
Explanation: You must specify three parameters to the RXMQVSET call.

-2 0 0 RXMQVSET Handle not supplied
Explanation: No value has been keyed for the first parameter, the handle representing the MQ object.

-3 0 0 RXMQVSET Attribute not supplied
Explanation: No value has been keyed for the second parameter, a number representing the attribute of the MQ object to be set.

-4 0 0 RXMQVSET Attribute not numeric
Explanation: The value supplied for the second parameter, a number representing representing the attribute of the MQ object to be obtained, is not actually numeric.

-5 0 0 RXMQVSET Attribute Setting not supplied
Explanation: No value was supplied for the attribute under consideration.

-6 0 0 RXMQVSET QM not connected
Explanation: The current TCB is not Connected to a Queue Manager

-7 0 0 RXMQVSET Thread not connected
Explanation: The current TCB is not Connected to a Queue Manager

-8 0 0 RXMQVSET Handle out of range
Explanation: The current TCB is not Connected to a Queue Manager

-9 0 0 RXMQVSET Handle invalid
Explanation: The value of the handle supplied is not in the known range for a handle within the interface.

-10 0 0 RXMQVSET Integer Attribute not numeric
Explanation: The value supplied for the third parameter, a number representing representing the integer attribute of the MQ object to be set, is not actually numeric.

-11 0 0 RXMQVSET Unknown Char Attribute
Explanation: The value of the requested attribute was found to be within the range for a Character attribute, but was not defined as a Character attribute.

-12 0 0 RXMQVSET Attribute out of valid range
Explanation: The value of the attribute under consideration was outside of the ranges defined for Integer and Character attributes.
Example

```
rcc = RXMQV('SET', hn1, MQIA_TRIGGER_DEPTH, 21)
```

This call sets the Trigger Depth for the Queue specified by hn1 (which must have been Opened with Set access) to 21 messages.
RXMQVCMIT

Description

This verb will issue a MQCMIT verb. It syncpoints the current Queue Manager accesses. Note that this operation affects all the currently accessed queues which have extant operations within Unit of Work control within the current thread (ie: it does not effect other threads within the process).

Parameters

None

Call

```
rcc = RXMQV('CMIT')
```

Additional Interface Return Codes and Messages

-1 0 0 RXMQVCMIT Bad number ofParms

Explanation: You cannot specify any parameters to this call.

-2 0 0 RXMQVCMIT Thread not connected to QM

Explanation: The current thread is not Connected to a Queue Manager

Example

```
rcc = RXMQV('CMIT')
```

The accesses to all currently accessed Queues (that are within Unit of Work control) are committed. Accesses outside of UOW control are unaffected by this call.
RXMQVBACK

Description

This verb will issue a MQBACK verb. It rolls back the current Queue Manager accesses. Note that this operation affects all the currently accessed queues which have extant operations within Unit of Work control within the current thread (ie: it does not effect other threads within the process).

Parameters

None

Call

```
rcc = RXMQV('BACK')
```

Additional Interface Return Codes and Messages

-1 0 0 RXMQVBACK Bad number ofParms

Explanation: You cannot specify any parameters to this call.

-2 0 0 RXMQVBACK Thread not connected toQM

Explanation: The current thread is not Connected to a Queue Manager

Example

```
rcc = RXMQV('BACK')
```

The accesses to all currently accessed Queues (that are within Unit of Work control) are rolled back. Accesses outside of UOW control are unaffected by this call.
RXMQVGET

Description

This call will obtain a message from a Queue, using the underlying MQGET verb. All the abilities of this verb are supported by this interface.

A quick way of issuing Browse calls is provided by “RXMQVBROWSE” on page 54. When using a Get.Signal operation, the ECB can only be tested by using the RXMQV('QSIGNAL') operation as described in “RXMQVQSIGNAL” on page 51.

Parameters

1. The Handle for the Queue obtained from a prior RXMQV('OPEN') call, whereat the Queue was opened for Input (or Browse) access. This is an Input parameter.

2. The name of a Rexx Stem variable (including the dot) into which the obtained message will be placed. This is an input/output parameter. Upon the call, Component 0 must contain the Maximum length of the message to be received. After the call, Component 0 will contain the length of the message received (or would have been received if the initial setting was 0) and Component 1 will contain the obtained message (if any). See “Message Lengths” on page 12 for a warning about truncation.

3. The name of a Stem variable (including the dot) containing the Input Message Descriptor describing the Message to be obtained from the Queue. This is an input parameter.

4. The name of a Stem variable (including the dot) into which will be returned a Message Descriptor describing the message obtained by the call. This is an output parameter, so ZLIST processing is provided.

5. The name of a Stem variable (including the dot) containing the Get Message Options for the operation. This is an input parameter.

6. The name of a Stem variable (including the dot) into which will be placed the updated Get Message Options resulting from the call. This is an output parameter, so ZLIST processing is provided.

Call

```rexx

Additional Interface Return Codes and Messages

-1 0 0 RXMQVGET Bad number ofParms
Explanation: You must specify six parameters to the RXMQVGET call.

-2 0 0 RXMQVGET Handle not supplied
Explanation: No value has been keyed for the first parameter, the handle representing the MQ object.

-3 0 0 RXMQVGET Stem. Data Variable name not supplied
Explanation: No value has been keyed for the second parameter, the name of a Stem Variable containing the maximum length of message to be obtained.

-4 0 0 RXMQVGET Input Stem. MD Var name not supplied
Explanation: No value has been keyed for the third parameter, the name of a Stem Variable containing the Input Message Variable for the operation.

-5 0 0 RXMQVGET Output Stem. MD Var name not supplied
Explanation: No value has been keyed for the forth parameter, the name of a Stem Variable into which will be placed the resulting Message Descriptor from the operation.

-6 0 0 RXMQVGET Input Stem. GMO Var name not supplied
Explanation: No value has been keyed for the fifth parameter, the name of a Stem Variable containing the Get Message Options for the operation.

-7 0 0 RXMQVGET Output Stem. GMO Var name not supplied
Explanation: No value has been keyed for the sixth parameter, the name of a Stem Variable into which will be placed the resulting Get Message

-8 0 0 RXMQVGET QM not connected
Explanation: The current TCB is not Connected to a Queue Manager

-9 0 0 RXMQVGET Thread not connected
Explanation: The current TCB is not Connected to a Queue Manager

-10 0 0 RXMQVGET Handle out of range
Explanation: The current TCB is not Connected to a Queue Manager

-11 0 0 RXMQVGET Handle invalid
Explanation: The value of the handle supplied is not in the known range for a handle within the interface.
Example

```rxml
message.0 = 100
message.1 = ''

igmo.opt  = MQGMO_WAIT + MQGMO_SYNCPOINT + MQGMO_FAIL_IF_QUIESCING
igmo.wait = 1

imd.MSGID = ''
imd.CID   = ''

c = RXMQV('GET', hnl, 'message.', 'imd.', 'omd.', 'igmo.', 'ogmo.' )  
/* on return, say.....
   message.0 = 13
   message.1 = 'RAH rules OK1'
   omd.msg   = MQMT_DATAGRAM
   omd.PER   = MQPER_PERSISTENT
   ...
   ogmo.rqn  = 'N1'
*/
```

This call destructively obtains the next message from the Queue. The message can be up to 100 bytes long - a bigger message is not obtained (as the options does not specify MQGMO_ACCEPT_TRUNCATED_MSG). The obtained message (which will not physically be removed from the Queue until a Syncpoint is issued, as it is obtained under Unit Of Work control) is 13 bytes long, and is persistent.
RXMQVPUT

Description

This call will place a message into a Queue, using the underlying MQPUT verb. All the abilities of this verb are supported by this interface.

Parameters

1. The Handle for the Queue obtained from a prior RXMQV('OPEN') call, whereat the Queue was opened for Output access. This is an Input parameter.

2. The name of a Rexx Stem variable (including the dot) containing the message to be placed on the Queue. This is an input parameter. Component 0 must contain the length of Component 1, which is the message to be put into the Queue.

3. The name of a Stem variable (including the dot) containing the Input Message Descriptor describing the Message to be placed on the Queue. This is an input parameter.

4. The name of a Stem variable (including the dot) into which will be returned a Message Descriptor describing the message placed by the call. This is an output parameter, so ZLIST processing is provided.

5. The name of a Stem variable (including the dot) containing the Put Message Options for the operation. This is an input parameter.

6. The name of a Stem variable (including the dot) into which will be placed the updated Put Message Options resulting from the call. This is an output parameter, so ZLIST processing is provided.

Call

``` Rexx
```
Additional Interface Return Codes and Messages

-1 0 0 RXMQVPUT Bad number of Parms
Explanation: You must specify six parameters to the RXMQVPUT call.

-2 0 0 RXMQVPUT Handle not supplied
Explanation: No value has been keyed for the first parameter, the handle representing the MQ object.

-3 0 0 RXMQVPUT Stem. Data Variable name not supplied
Explanation: No value has been keyed for the second parameter, the name of a Stem Variable containing the maximum length of message to be obtained.

-4 0 0 RXMQVPUT Input Stem. MD Var name not supplied
Explanation: No value has been keyed for the third parameter, the name of a Stem Variable containing the Input Message Variable for the operation.

-5 0 0 RXMQVPUT Output Stem. MD Var name not supplied
Explanation: No value has been keyed for the forth parameter, the name of a Stem Variable into which will be placed the resulting Message Descriptor from the operation.

-6 0 0 RXMQVPUT Input Stem. PMO Var name not supplied
Explanation: No value has been keyed for the fifth parameter, the name of a Stem Variable containing the Put Message Options for the operation.

-7 0 0 RXMQVPUT Output Stem. PMO Var name not supplied
Explanation: No value has been keyed for the sixth parameter, the name of a Stem Variable into which will be placed the resulting Put Message

-8 0 0 RXMQVPUT QM not connected
Explanation: The current TCB is not Connected to a Queue Manager

-9 0 0 RXMQVPUT Thread not connected
Explanation: The current TCB is not Connected to a Queue Manager

-10 0 0 RXMQVPUT Handle out of range
Explanation: The current TCB is not Connected to a Queue Manager

-11 0 0 RXMQVPUT Handle invalid
Explanation: The value of the handle supplied is not in the known range for a handle within the interface.
Example

```rexx
message.0 = 27
message.1 = 'RAH''s wonderful interface!

ipmo.opt = MQGMO_NO_SYNCPOINT + MQPMO_NO_CONTEXT,
            + MQPMO_FAIL_IFQUIESCING

imd.MSG = MQMT_DATAGRAM
imd.per = MQPER_NOT_PERSISTENT

rcc = RXMQV('PUT', h1, 'message.', 'imd.', 'omd.', 'ipmo.', 'opmo. ')

/* on return, say.....
  omd. PD = 19940831
...
  opmo.rqn = 'N1'
  */
```

This call places the given non-persistent message on the Queue outside of a Unit of Work.
RXMQVQ SIGNAL

Description

This call is used after a proceeding Get.Signal (via RXMQV('GET') with the Options specifying MQGMO_SET_SIGNAL). RXMQV('Q SIGNAL') is used to test the ECB (which is not externalised outside of RXMQV itself) to see whether the Get.Signal has completed or not.

If the Get.Signal has not completed (via the arrival of a Message or MQ timing out the Get.Signal request etc.), then the first word of the Return Code will be <=0 (-ve for an error, 0 for request pending).

Upon completion, the first word of the Return Code will be >0, this being the Completion Code as described in the Signal part of the GMO documentation in the APR).

Parameters

1. The Handle for the object obtained from a prior RXMQV('OPEN') call, whereat the object was opened for Input and the last input operation was a Get.Signal (RXMQV('GET') with the Options specifying MQGMO_SET_SIGNAL).

Call

```ruby
rcc = RXMQV('Q SIGNAL', handle)
```
Additional Interface Return Codes and Messages

-1 0 0 RXMQVQSIG Bad number ofParms
Explanation: You must specify one parameter to the RXMQV('QSIGNAL') call.

-2 0 0 RXMQVQSIG Handle not supplied
Explanation: No value has been keyed for the first parameter, the handle representing the MQ object.

-3 0 0 RXMQVQSIG QM not connected
Explanation: The current TCB is not Connected to a Queue Manager

-4 0 0 RXMQVQSIG Thread not connected
Explanation: The current TCB is not Connected to a Queue Manager

-5 0 0 RXMQVQSIG Handle out of range
Explanation: The current TCB is not Connected to a Queue Manager

-6 0 0 RXMQVQSIG Handle invalid
Explanation: The value of the handle supplied is not in the known range for a handle within the interface.

0 0 0 RXMQVQSIG OK Signal not raised
Explanation: The Get.Signal has not yet completed

>0 0 0 RXMQVQSIG OK Signal raised
Explanation: The Get.Signal has completed for the reason given in the first word
Example

``` Rexx
message.0 = 100
message.1 = '

igmo.opt = MQGMO_WAIT + MQGMO_SYNCPOINT + MQGMO_SET_SIGNAL
igmo.wait = 0

imd.MSGID = '
imd.CID = '

rcc = RXMQV('GET', hnl, 'message.', 'imd.', 'omd.', 'igmo.', 'ogmo.' )

/* on return, rcc = '2070 1 2070 RXMQVGET WARNING'
   showing the Get.Signal is pending  */

do forever
   rcc = RXMQV('Q SIGNAL', hnl )
   if ( word(rcc,1) > 0 ) then do
      say 'Get.Signal completed '
      leave
   end
   /* do something interesting whilst waiting */
end

message.0 = 100
message.1 = '

igmo.opt = MQGMO_WAIT + MQGMO_SYNCPOINT
igmo.wait = 0

imd.MSGID = '
imd.CID = '

rcc = RXMQV('GET', hnl, 'message.', 'imd.', 'omd.', 'igmo.', 'ogmo.' )

/* on immediate return, say .......

   message.0 = 13
   message.1 = 'RAH rules OK1'

   omd.msg = MQMT Datagram
   omd.PER = MQPER_PERSISTENT
   ...
   ogmo.rqn = 'N1'

   */
```

This example shows how RXMQVQSIGNAL is used to permit processing whilst awaiting the arrival of a message.
RXMQVBROWSE

Description

This call is an extension to the MQ/MVS API as documented in the APR. This call will obtain the next message from a Queue via a Browse operation, using the underlying Browse function of the MQGET verb.

As this call is designed to be simple way to browse messages on a Queue, no Get Message Options or Message Descriptors are available. If access to these is required, then use the base “RXMQVGET” on page 45.

Similarly, the position of the Browse cursor cannot be manipulated.

Parameters

1. The Handle for the Queue obtained from a prior RXMQV('OPEN') call, whereat the Queue was opened for Browse access. This is an Input parameter.

2. The name of a Rexx Stem variable (including the dot) into which the obtained message will be placed. This is an input/output parameter. Upon the call, Component 0 must contain the Maximum length of the message to be received. After the call, Component 0 will contain the length of the message received (or would have been received if the initial setting was 0) and Component 1 will contain the obtained message (if any). See “Message Lengths” on page 12 for a warning about truncation (Browse will always truncate the message and return in .0 the length of the data returned, not that which would have been returned if .0 was big enough).

Call

\[
\text{rcc} = \text{RXMQV('BROWSE', handle, 'Stem.Message.' )}
\]
Additional Interface Return Codes and Messages

-1 0 0 RXMQVBROWSE Bad number of Parms
Explanation: You must specify two parameters to the RXMQVBROWSE call.

'-2 0 0 RXMQVBROWSE Handle not supplied
Explanation: No value has been keyed for the first parameter, the handle representing the MQ object.

-3 0 0 RXMQVBROWSE Stem. Data Variable name not supplied
Explanation: No value has been keyed for the second parameter, the name of a Stem Variable containing the maximum length of message to be obtained.

-4 0 0 RXMQVBROWSE QM not connected
Explanation: The current TCB is not Connected to a Queue Manager

-5 0 0 RXMQVBROWSE Thread not connected
Explanation: The current TCB is not Connected to a Queue Manager

-6 0 0 RXMQVBROWSE Handle out of range
Explanation: The current TCB is not Connected to a Queue Manager

-7 0 0 RXMQVBROWSE Handle invalid
Explanation: The value of the handle supplied is not in the known range for a handle within the interface.

Example

```ruby
message.0 = 100
message.1 = ''

rcc = RXMQV('BROWSE', hn1, 'message.' )

/* on return, say..... message.0 = 2 ; message.1 = 'M1' */

message.0 = 100
message.1 = ''

rcc = RXMQV('BROWSE', hn1, 'message.' )

/* on return, say..... message.0 = 8 ; message.1 = '>>>M2<<<' */
```

This example shows how a Browse is used to scan a Queue; observe that the message. Stem variable is cleared before each use.
RXMQVHXT

Description

This call will take a message obtained from a Transmission Queue or a Dead Letter Queue (identified by the relevant prefix in the message) and split it up into its components.

This Header Extraction, therefore, permits the obtention of the 'real' message and an explanation of the control data associated with it.

The message to be split up is specified in the usual way as the name of a stem variable; with component 0 representing the length of the message which is supplied in component 1. See “Message Lengths” on page 12 for a warning about truncated messages used with this function.

The Extracted data is placed in another stem variable (whose name is supplied); with component 0 representing the length of the 'actual' message which is placed in component 1. The associated data is placed in other components, as shown in Table 5 on page 58 and Table 6 on page 59. It is not recommended that the input and output stem variables are the same (as this might loose information in the case of an error and additionally the component names clash with those generated as part of the Message descriptor).

In order to identify the type of header extracted, a component called TYPE is also created, taking the value of XQH or DLH (this is also provided in the NAME component).

Parameters

1. The name of a Rexx Stem variable (including the dot) containing a message to be splitup. This is an input parameter. Upon the call, Component 0 must contain the length of the message in Component 1; the message must have been obtained from a Transmission Queue or a Dead Letter Queue. See “Message Lengths” on page 12 for a warning about truncation.

2. The name of a Rexx Stem variable (including the dot) into which the splitup message will be placed. This is an input/output parameter. After the call, Component 0 will contain the length of the 'actual' message and Component 1 will contain the 'actual' message (if any). Other components will be created (as documented in Table 5 on page 58 and Table 6 on page 59) to return the extracted Header information from the input message. ZLIST processing is provided for this Stem variable.

Call

```
  rcc = RXMQV('HXT', 'Stem.Message.', 'Stem.Splitup.')
```
Additional Interface Return Codes and Messages

-1 0 0 RXMQVHXT Bad number ofParms
Explanation: You must specify two parameters to the RXMQVHXT call.

-2 0 0 RXMQVHXT Stem. Data Variable name not supplied
Explanation: No value has been keyed for the first parameter, the name of a Stem. variable representing the message to be splitup.

-3 0 0 RXMQVHXT Output Stem. Var name not supplied
Explanation: No value has been keyed for the second parameter, the name of a Stem. variable representing the splitup message.

-4 0 0 RXMQVHXT No Data for Header Extraction
Explanation: The input Stem.0 was zero, indicating no message to process

-5 0 0 RXMQVHXT Message is too short for an Header
Explanation: The input Stem.0 was <= 3, indicating no header in the message

-6 0 0 RXMQVHXT Message is too short for a DLH
Explanation: Although the input Stem.1 looked like a DLH, Stem.0 was too small for the message to originate from a Dead Letter Queue, and so cannot be splitup

-7 0 0 RXMQVHXT Message is too short for a XQH
Explanation: Although the input Stem.1 looked like a XQH, Stem.0 was too small for the message to originate from a Transmission Queue, and so cannot be splitup

-8 0 0 RXMQVHXT Unknown Message Header
Explanation: The first 4 bytes of the input Stem.1 was not 'DLH ' or 'XQH ', so the message did not come from a Dead Letter Queue or a Transmission Queue, and so cannot be splitup
## Extracted information

### Transmission Queue Messages

Table 5. Transmission Queue Message Extracts

<table>
<thead>
<tr>
<th>Stem. Component</th>
<th>MQXQH Structure name</th>
<th>Number or Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>.0</td>
<td>actual message length</td>
<td>N</td>
</tr>
<tr>
<td>.1</td>
<td>actual message</td>
<td>T</td>
</tr>
<tr>
<td>.TYPE</td>
<td>set to XQH</td>
<td>T</td>
</tr>
<tr>
<td>.NAME</td>
<td>set to XQH</td>
<td>T</td>
</tr>
<tr>
<td>.RQM</td>
<td>RemoteQMgrName</td>
<td>T</td>
</tr>
<tr>
<td>.RQN</td>
<td>RemoteQName</td>
<td>T</td>
</tr>
<tr>
<td>.REP</td>
<td>MsgDesc.Report</td>
<td>N</td>
</tr>
<tr>
<td>.MSG</td>
<td>MsgDesc.MsgType</td>
<td>N</td>
</tr>
<tr>
<td>.EXP</td>
<td>MsgDesc.Expiry</td>
<td>N</td>
</tr>
<tr>
<td>.FBK</td>
<td>MsgDesc.Feedback</td>
<td>N</td>
</tr>
<tr>
<td>.ENC</td>
<td>MsgDesc.Encoding</td>
<td>N</td>
</tr>
<tr>
<td>.CCSI</td>
<td>MsgDesc.CodedCharSetId</td>
<td>N</td>
</tr>
<tr>
<td>.FORM</td>
<td>MsgDesc.Format</td>
<td>N</td>
</tr>
<tr>
<td>.PRI</td>
<td>MsgDesc.Priority</td>
<td>N</td>
</tr>
<tr>
<td>.PER</td>
<td>MsgDesc.Persistence</td>
<td>N</td>
</tr>
<tr>
<td>.MSGID</td>
<td>MsgDesc.MsgId</td>
<td>T</td>
</tr>
<tr>
<td>.CID</td>
<td>MsgDesc.CorrelId</td>
<td>T</td>
</tr>
<tr>
<td>.BC</td>
<td>MsgDesc.BackoutCount</td>
<td>N</td>
</tr>
<tr>
<td>.RTOQ</td>
<td>MsgDesc.ReplyToQ</td>
<td>T</td>
</tr>
<tr>
<td>.RTOQM</td>
<td>MsgDesc.ReplyToQMgr</td>
<td>T</td>
</tr>
<tr>
<td>.UID</td>
<td>MsgDesc.UserIdentifier</td>
<td>T</td>
</tr>
<tr>
<td>.AT</td>
<td>MsgDesc.AccountingToken</td>
<td>T</td>
</tr>
<tr>
<td>.AID</td>
<td>MsgDesc.ApplyIdentityData</td>
<td>T</td>
</tr>
<tr>
<td>.PAT</td>
<td>MsgDesc.PutApplType</td>
<td>T</td>
</tr>
<tr>
<td>.PAN</td>
<td>MsgDesc.PutApplName</td>
<td>T</td>
</tr>
<tr>
<td>.PD</td>
<td>MsgDesc.PutDate</td>
<td>T</td>
</tr>
<tr>
<td>.PT</td>
<td>MsgDesc.PutTime</td>
<td>T</td>
</tr>
<tr>
<td>.AOD</td>
<td>MsgDesc.ApplOriginData</td>
<td>T</td>
</tr>
</tbody>
</table>

Note:
- Number or Text shows the type of the field
- ZLIST is set to '0 1 AID AOD AT BC CID CCSI ENC EXP FBK FORM MSG MSGID NAME PAN PAT PD PER PRI PT REP RQM RQN RTOQ RTOQM TYPE UID'
## Dead Letter Queue Messages

Table 6. Dead Letter Queue Message Extracts

<table>
<thead>
<tr>
<th>Stem Component</th>
<th>MQDLH Structure name</th>
<th>Number or Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>.0</td>
<td>actual message length</td>
<td>N</td>
</tr>
<tr>
<td>.1</td>
<td>actual message</td>
<td>T</td>
</tr>
<tr>
<td>.TYPE</td>
<td>set to DLH</td>
<td>T</td>
</tr>
<tr>
<td>.NAME</td>
<td>set to DLH</td>
<td>T</td>
</tr>
<tr>
<td>.REA</td>
<td>Reason</td>
<td>N</td>
</tr>
<tr>
<td>.DQM</td>
<td>DestinationQMgrName</td>
<td>T</td>
</tr>
<tr>
<td>.DQN</td>
<td>DestinationQName</td>
<td>T</td>
</tr>
<tr>
<td>.ENC</td>
<td>Encoding</td>
<td>N</td>
</tr>
<tr>
<td>.CCSI</td>
<td>CodedCharSetId</td>
<td>N</td>
</tr>
<tr>
<td>.FORM</td>
<td>Format</td>
<td>N</td>
</tr>
<tr>
<td>.PAT</td>
<td>PutApplType</td>
<td>T</td>
</tr>
<tr>
<td>.PAN</td>
<td>PutApplName</td>
<td>T</td>
</tr>
<tr>
<td>.PD</td>
<td>PutDate</td>
<td>T</td>
</tr>
<tr>
<td>.PT</td>
<td>PutTime</td>
<td>T</td>
</tr>
</tbody>
</table>

**Note:**
- Number or Text shows the type of the field
- ZLIST is set to '0 1 CCSI DQM DQN ENC FORM NAME PAN PAT PD PT REA TYPE'
Example

/* A message has been obtained such that ... */

message.0 = 438
message.1 = <XQH>1234567890

/* Clear the result variable */
drop x.

/* Split the message */

rcc = RXMQV('HXT', 'message.', 'x.' )

/* on return, the following (and more) are set */

say x.0 /* 10 */
say x.1 /* 1234567890 */
say x.RQM /* RAH2 */
say x.RQN /* CP1 */
say x.PER /* 1 */
say x.TYPE /* XQH */

This example shows how a message obtained from a Transmission Queue is splitup, showing information extracted from the XQH and the 'actual' message being transmitted.
**RXMQVEVENT**

**Description**

This call will take a message obtained from an Event Queue (in general the default system queues called SYSTEM.ADMIN.QMGR.EVENT, SYSTEM.ADMIN.PERFM.EVENT and SYSTEM.ADMIN.CHANNEL.EVENT) and split it up into its components.

This Event Extraction, therefore, permits the detection of the event and an explanation of the control data associated with it.

The message to be split up is specified in the usual way as the name of a stem. variable; with component 0 representing the length of the message which is supplied in component 1. See “Message Lengths” on page 12 for a warning about truncated messages used with this function. This message will have come from a prior RXMQV('BROWSE') or RXMQV('GET') operation.

The Extracted data is placed in another stem. variable (whose name is supplied), with the various components contained information about the event. Table 8 on page 65 gives a mapping between the PCF variable name and the component name. It is **not** recommended that the input and output stem variables are the same (as this might loose information in the case of an error and additionally the component names clash with those generated as part of the Message descriptor). Observe that some information is held in the event message’s Message Descriptor (like Date and Time), so obtaining the message should be done via a Browse-type of RXMQV('GET') rather than the RXMQV('BROWSE') call which does not return the Message Descriptor if this type of information is required.

In order to identify the type of event extracted, a component called TYPE is created and set to EVENT, and another called NAME which interprets the Event (see Table 7 on page 64 for this mapping).

Information about Events is discussed in SC33-1482-01, the Programmable System Management book which you should use to interpret the expansion.

--- Warning ---

The PCF Documentation on events sometimes does not agree with what is actually recorded in the Event Message. Please take care in this arena, and treat deviations from the Documentation pragmatically (ie: raise an APAR, but process as this interface returns).

The Components returned are those documented in SC33-1482-01 for each event (with these fields mapped according to Table 7 on page 64). Table 8 on page 65 shows this information in a tabular form. However, a general usage should test each component to discover whether or not this information is returned. Alternatively, use ZLIST processing (as described in “ZLIST” on page 14). A returned component may be null (or have a zero length) if the Event Field is present without any data.
Parameters

1. The name of a Rexx Stem variable (including the dot) containing an event message to be split up. This is an input parameter. Upon the call, Component 0 must contain the length of the message in Component 1; the message must have been obtained from an Event Queue. See “Message Lengths” on page 12 for a warning about truncation.

2. The name of a Rexx Stem variable (including the dot) into which the split up message will be placed. This is an input/output parameter. After the call, components will be created (as documented in Table 8 on page 65 and Table 9 on page 71) to return the extracted event information from the input message. ZLIST processing is provided for this Stem variable.

Call

```
rcc = RXMQV('EVENT', 'Stem.Message.', 'Stem.Splitup.')```

Usage Notes

Bear in mind the following when using RXMQVEVENT:

- A component is returned when the relevant parameter is present in the PCF Event Message. The returned data may consist of binary zeros, a null string ("") or all spaces if the contents do not exist (this is due to the way MQ/MVS builds the PCF Event message). Therefore, use ZLIST processing to remove binary zeros and excess spaces as shown in Figure 7 on page 73. Certain Rexx processors object to long strings of Binary zeros, so you have been warned!
- The PCF Event documentation may differ from the data actually returned. This is either a bug in the documentation or the MQ/MVS code. Always use ZLIST processing to see what is going on!
- The EID, AEDI1, AEDI2 and CED fields are not returned as numbers, but rather in Hex. This will aid problem determination for these Channel error codes.
- There may be more than one CED field. In this case, .CED.0 will contain the number of fields, with the data being in .CED.n
- The Date and Time of an Event is not held within the event, but in the Message Descriptor for the event.
- .TYPE is set to 'EVENT' for all events.
**Additional Interface Return Codes and Messages**

- **-1 0 0 RXMQVEVENT Bad number ofParms**
  
  **Explanation:** You *must* specify two parameters to the RXMQVEVENT call.

- **-2 0 0 RXMQVEVENT Input Variable name/data not supplied**

  **Explanation:** No value has been keyed for the first parameter, the name of a Stem. variable representing the message to be splitup.

- **-3 0 0 RXMQVEVENT Output Stem. Var name not supplied**

  **Explanation:** No value has been keyed for the second parameter, the name of a Stem. variable representing the splitup message.

- **-4 0 0 RXMQVEVENT No Data for Event Extraction**

  **Explanation:** The input Stem.0 was zero, indicating no message to process.

- **-5 0 0 RXMQVEVENT Message is too short for an Event**

  **Explanation:** Although the input Stem.1 looked like an Event Message, Stem.0 was too small for the message to originate from an Event Queue, and so cannot be splitup.

- **-6 0 0 RXMQVEVENT Message is not an Event**

  **Explanation:** The first 4 bytes of the input Stem.1 was not `<MQCFH_EVENT>`, so the message did not come from an Event Queue, and so cannot be splitup.

- **-7 0 0 RXMQVEVENT Unknown Event Category**

  **Explanation:** Although the input Stem.1 looked like an Event Message, the PCF *Command* field did not contain a recognisable event category, and so the message cannot be splitup.

- **-8 0 0 RXMQVEVENT Unknown Event Reason**

  **Explanation:** Although the input Stem.1 looked like an Event Message, the PCF *Reason* field did not contain a recognisable event identifier, and so the message cannot be splitup.

- **-9 0 0 RXMQVEVENT No elements in the Event**

  **Explanation:** Although the input Stem.1 looked like an Event Message, there were no PCF fields within the Message, and so the message cannot be splitup.
## Extracted information

### Event Names

<table>
<thead>
<tr>
<th>PCF Reason field value</th>
<th>Reason Number</th>
<th>.NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQRC_Q_MGR_ACTIVE</td>
<td>2222</td>
<td>QMACT</td>
</tr>
<tr>
<td>MQRC_Q_MGR_NOT_ACTIVE</td>
<td>2223</td>
<td>QMINA</td>
</tr>
<tr>
<td>MQRC_GET_INHIBITED</td>
<td>2016</td>
<td>INGET</td>
</tr>
<tr>
<td>MQRC_PUT_INHIBITED</td>
<td>2051</td>
<td>INPUT</td>
</tr>
<tr>
<td>MQRC_ALIAS_BASE_Q_TYPE_ERROR</td>
<td>2001</td>
<td>LLAQT</td>
</tr>
<tr>
<td>MQRC_UNKNOWN_ALIAS_BASE_Q</td>
<td>2082</td>
<td>LLABQ</td>
</tr>
<tr>
<td>MQRC_UNKNOWN_OBJECT_NAME</td>
<td>2085</td>
<td>LLUON</td>
</tr>
<tr>
<td>MQRC_CHANNEL_CONV_ERROR</td>
<td>2284</td>
<td>CHCONV</td>
</tr>
<tr>
<td>MQRC_CHANNEL_STARTED</td>
<td>2282</td>
<td>CHSTRK</td>
</tr>
<tr>
<td>MQRC_CHANNEL_STOPPED</td>
<td>2283</td>
<td>CHSTOP</td>
</tr>
<tr>
<td>MQRC_CHANNEL_ACTIVATED</td>
<td>2295</td>
<td>CHACT</td>
</tr>
<tr>
<td>MQRC_CHANNEL_NOT_ACTIVATED</td>
<td>2296</td>
<td>CHNACT</td>
</tr>
<tr>
<td>MQRC_BRIDGE_STARTED</td>
<td>2125</td>
<td>BRSTRT</td>
</tr>
<tr>
<td>MQRC_BRIDGE_STOPPED</td>
<td>2126</td>
<td>BRSTOP</td>
</tr>
<tr>
<td>MQRC_Q_DEPTH_HIGH</td>
<td>2224</td>
<td>PFQDH</td>
</tr>
<tr>
<td>MQRC_Q_DEPTH_LOW</td>
<td>2225</td>
<td>PFQDL</td>
</tr>
<tr>
<td>MQRC_Q_FULL</td>
<td>2053</td>
<td>PFQFU</td>
</tr>
<tr>
<td>MQRC_Q_SERVICE_INTERVAL_HIGH</td>
<td>2226</td>
<td>PFQSH</td>
</tr>
<tr>
<td>MQRC_Q_SERVICE_INTERVAL_OK</td>
<td>2227</td>
<td>PFQSO</td>
</tr>
<tr>
<td>MQRC_DEF_XMIT_Q_TYPE_ERROR</td>
<td>2198</td>
<td>RMDXQT</td>
</tr>
<tr>
<td>MQRC_DEF_XMIT_Q_USAGE_ERROR</td>
<td>2199</td>
<td>RMDXQU</td>
</tr>
<tr>
<td>MQRC_Q_TYPE_ERROR</td>
<td>2057</td>
<td>RMQUTY</td>
</tr>
<tr>
<td>MQRC_REMOTE_Q_NAME_ERROR</td>
<td>2184</td>
<td>RMREQA</td>
</tr>
<tr>
<td>MQRC_XMIT_Q_TYPE_ERROR</td>
<td>2091</td>
<td>RMXQTY</td>
</tr>
<tr>
<td>MQRC_XMIT_Q_USAGE_ERROR</td>
<td>2092</td>
<td>RMXQUS</td>
</tr>
<tr>
<td>MQRC_UNKNOWN_DEF_XMIT_Q</td>
<td>2197</td>
<td>RMUDXQ</td>
</tr>
<tr>
<td>MQRC_UNKNOWN_REMOTE_Q_MGR</td>
<td>2087</td>
<td>RMURQM</td>
</tr>
<tr>
<td>MQRC_UNKNOWN_XMIT_Q</td>
<td>2196</td>
<td>RRUXQN</td>
</tr>
<tr>
<td>MQRC_NOT_AUTHORIZED</td>
<td>2035</td>
<td>NAAUT</td>
</tr>
</tbody>
</table>

**Note:**
- The Event name is returned as `.NAME`
- The Reason is in `.REA`
- `.TYPE` is set to 'EVENT'
## Component Names

Table 8 (Page 1 of 6). Event Names

<table>
<thead>
<tr>
<th>.item</th>
<th>PCF Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLID</td>
<td>MQCA_APPL_ID</td>
</tr>
<tr>
<td>BREQQN</td>
<td>MQCA_BACKOUT_REQ_Q_NAME</td>
</tr>
<tr>
<td>BQN</td>
<td>MQCA_BASE_Q_NAME</td>
</tr>
<tr>
<td>CIQN</td>
<td>MQCA_COMMAND_INPUT_Q_NAME</td>
</tr>
<tr>
<td>CREDATE</td>
<td>MQCA_CREATION_DATE</td>
</tr>
<tr>
<td>CRETIME</td>
<td>MQCA_CREATION_TIME</td>
</tr>
<tr>
<td>DLQQN</td>
<td>MQCA_DEAD_LETTER_Q_NAME</td>
</tr>
<tr>
<td>DEFXQN</td>
<td>MQCA_DEF_XMIT_Q_NAME</td>
</tr>
<tr>
<td>ENVDATA</td>
<td>MQCA_ENV_DATA</td>
</tr>
<tr>
<td>IQN</td>
<td>MQCA_INITIATION_Q_NAME</td>
</tr>
<tr>
<td>NAMES</td>
<td>MQCA_NAMES</td>
</tr>
<tr>
<td>PDESC</td>
<td>MQCA_PROCESS_DESC</td>
</tr>
<tr>
<td>PN</td>
<td>MQCA_PROCESS_NAME</td>
</tr>
<tr>
<td>QDESC</td>
<td>MQCA_Q_DESC</td>
</tr>
<tr>
<td>QMDESC</td>
<td>MQCA_Q_MGR_DESC</td>
</tr>
<tr>
<td>QM</td>
<td>MQCA_Q_MGR_NAME</td>
</tr>
<tr>
<td>QN</td>
<td>MQCA_Q_NAME</td>
</tr>
<tr>
<td>RQM</td>
<td>MQCA_REMOTE_Q_MGR_NAME</td>
</tr>
<tr>
<td>RQN</td>
<td>MQCA_REMOTE_Q_NAME</td>
</tr>
<tr>
<td>STGCLASS</td>
<td>MQCA_STORAGE_CLASS</td>
</tr>
<tr>
<td>TRIGDATA</td>
<td>MQCA_TRIGGER_DATA</td>
</tr>
<tr>
<td>USERDATA</td>
<td>MQCA_USER_DATA</td>
</tr>
<tr>
<td>XXQN</td>
<td>MQCA_XMIT_Q_NAME</td>
</tr>
<tr>
<td>AQNS</td>
<td>MQCACF_ALIAS_Q_NAMES</td>
</tr>
<tr>
<td>AN</td>
<td>MQCACF_APPL_NAME</td>
</tr>
<tr>
<td>AEDS1</td>
<td>MQCACF_AUX_ERROR_DATA_STR_1</td>
</tr>
<tr>
<td>AEDS2</td>
<td>MQCACF_AUX_ERROR_DATA_STR_2</td>
</tr>
<tr>
<td>AEDS3</td>
<td>MQCACF_AUX_ERROR_DATA_STR_3</td>
</tr>
<tr>
<td>BRNAME</td>
<td>MQCACF_BRIDGE_NAME</td>
</tr>
<tr>
<td>ESC</td>
<td>MQCACF_ESCAPE_TEXT</td>
</tr>
<tr>
<td>FCN</td>
<td>MQCACF_FROM_CHANNEL_NAME</td>
</tr>
<tr>
<td>FPN</td>
<td>MQCACF_FROM_PROCESS_NAME</td>
</tr>
<tr>
<td>FQN</td>
<td>MQCACF_FROM_Q_NAME</td>
</tr>
<tr>
<td>LQNS</td>
<td>MQCACF_LOCAL_Q_NAMES</td>
</tr>
<tr>
<td>MQNS</td>
<td>MQCACF_MODEL_Q_NAMES</td>
</tr>
<tr>
<td>OQM</td>
<td>MQCACF_OBJECT_Q_MGR_NAME</td>
</tr>
</tbody>
</table>
Table 8 (Page 2 of 6). Event Names

<table>
<thead>
<tr>
<th>.item</th>
<th>PCF Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PNS</td>
<td>MQCACP_PROCESS_NAMES</td>
</tr>
<tr>
<td>QNS</td>
<td>MQCACP_Q_NAMES</td>
</tr>
<tr>
<td>RECCNS</td>
<td>MQCACP_RECEIVER_CHANNEL_NAMES</td>
</tr>
<tr>
<td>REMQNS</td>
<td>MQCACP_REMOTE_Q_NAMES</td>
</tr>
<tr>
<td>REQCNS</td>
<td>MQCACP_REQUESTER_CHANNEL_NAMES</td>
</tr>
<tr>
<td>SENDCNS</td>
<td>MQCACP_SENDER_CHANNEL_NAMES</td>
</tr>
<tr>
<td>SERVCNS</td>
<td>MQCACP_SERVER_CHANNEL_NAMES</td>
</tr>
<tr>
<td>TCN</td>
<td>MQCACP_TO_CHANNEL_NAME</td>
</tr>
<tr>
<td>TPN</td>
<td>MQCACP_TO_PROCESS_NAME</td>
</tr>
<tr>
<td>TQN</td>
<td>MQCACP_TO_Q_NAME</td>
</tr>
<tr>
<td>UID</td>
<td>MQCACP_USER_IDENTIFIER</td>
</tr>
<tr>
<td>CN</td>
<td>MQCACH_CHANNEL_NAME</td>
</tr>
<tr>
<td>CNS</td>
<td>MQCACH_CHANNEL_NAMES</td>
</tr>
<tr>
<td>CSD</td>
<td>MQCACH_CHANNEL_START_DATE</td>
</tr>
<tr>
<td>CST</td>
<td>MQCACH_CHANNEL_START_TIME</td>
</tr>
<tr>
<td>CONN</td>
<td>MQCACH_CONNECTION_NAME</td>
</tr>
<tr>
<td>CLUWID</td>
<td>MQCACH_CURRENT_LUWID</td>
</tr>
<tr>
<td>DESC</td>
<td>MQCACH_DESC</td>
</tr>
<tr>
<td>FORMAT</td>
<td>MQCACH_FORMAT_NAME</td>
</tr>
<tr>
<td>LLUWID</td>
<td>MQCACH_LAST_LUWID</td>
</tr>
<tr>
<td>LMSGD</td>
<td>MQCACH_LAST_MSG_DATE</td>
</tr>
<tr>
<td>LMSGT</td>
<td>MQCACH_LAST_MSG_TIME</td>
</tr>
<tr>
<td>MCAJN</td>
<td>MQCACH_MCA_JOB_NAME</td>
</tr>
<tr>
<td>MCAN</td>
<td>MQCACH_MCA_NAME</td>
</tr>
<tr>
<td>MCAUID</td>
<td>MQCACH_MCA_USER_ID</td>
</tr>
<tr>
<td>MCAMODE</td>
<td>MQCACH_MODE_NAME</td>
</tr>
<tr>
<td>MRXITN</td>
<td>MQCACH_MR_EXIT_NAME</td>
</tr>
<tr>
<td>MRXITUD</td>
<td>MQCACH_MR_EXIT_USER_DATA</td>
</tr>
<tr>
<td>MSGXN</td>
<td>MQCACH_MSG_EXIT_NAME</td>
</tr>
<tr>
<td>MSGXUD</td>
<td>MQCACH_MSG_EXIT_USER_DATA</td>
</tr>
<tr>
<td>PASSWORD</td>
<td>MQCACH_PASSWORD</td>
</tr>
<tr>
<td>RCVXN</td>
<td>MQCACH_RCV_EXIT_NAME</td>
</tr>
<tr>
<td>RCVXUD</td>
<td>MQCACH_RCV_EXIT_USER_DATA</td>
</tr>
<tr>
<td>SECCXN</td>
<td>MQCACH_SEC_EXIT_NAME</td>
</tr>
<tr>
<td>SECCXUD</td>
<td>MQCACH_SEC_EXIT_USER_DATA</td>
</tr>
<tr>
<td>SENDXN</td>
<td>MQCACH_SEND_EXIT_NAME</td>
</tr>
<tr>
<td>SENDXUD</td>
<td>MQCACH_SEND_EXIT_USER_DATA</td>
</tr>
</tbody>
</table>
### Table 8 (Page 3 of 6). Event Names

<table>
<thead>
<tr>
<th>.item</th>
<th>PCF Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPNAME</td>
<td>MQCACH_TP_NAME</td>
</tr>
<tr>
<td>USERID</td>
<td>MQCACH_USER_ID</td>
</tr>
<tr>
<td>XQN</td>
<td>MQCACH_XMIT_Q_NAME</td>
</tr>
<tr>
<td>AT</td>
<td>MQIA_APPL_TYPE</td>
</tr>
<tr>
<td>AE</td>
<td>MQIA_AUTHORITY_EVENT</td>
</tr>
<tr>
<td>BT</td>
<td>MQIA_BACKOUT_THRESHOLD</td>
</tr>
<tr>
<td>CCSID</td>
<td>MQIA_CODED_CHAR_SET_ID</td>
</tr>
<tr>
<td>CLEV</td>
<td>MQIA_COMMAND_LEVEL</td>
</tr>
<tr>
<td>CPILEV</td>
<td>MQIA_CPI_LEVEL</td>
</tr>
<tr>
<td>CURDEPTH</td>
<td>MQIA_CURRENT_Q_DEPTH</td>
</tr>
<tr>
<td>DEFIO</td>
<td>MQIA_DEF_INPUT_OPEN_OPTION</td>
</tr>
<tr>
<td>DEFFER</td>
<td>MQIA_DEF_PERSISTENCE</td>
</tr>
<tr>
<td>DEFPRI</td>
<td>MQIA_DEF_PRIORITY</td>
</tr>
<tr>
<td>DEFTYPE</td>
<td>MQIA_DEFINITION_TYPE</td>
</tr>
<tr>
<td>HARDENGB</td>
<td>MQIA_HARDEN_GET_BACKOUT</td>
</tr>
<tr>
<td>HQD</td>
<td>MQIA_HIGH_Q_DEPTH</td>
</tr>
<tr>
<td>IE</td>
<td>MQIA_INHIBIT_EVENT</td>
</tr>
<tr>
<td>IGET</td>
<td>MQIA_INHIBIT_GET</td>
</tr>
<tr>
<td>IPUT</td>
<td>MQIA_INHIBIT_PUT</td>
</tr>
<tr>
<td>LOCEV</td>
<td>MQIA_LOCAL_EVENT</td>
</tr>
<tr>
<td>MAXH</td>
<td>MQIA_MAX_HANDLES</td>
</tr>
<tr>
<td>MAXMSG</td>
<td>MQIA_MAX_MSG_LENGTH</td>
</tr>
<tr>
<td>MAXPRI</td>
<td>MQIA_MAX_PRIORITY</td>
</tr>
<tr>
<td>MAXDEPTH</td>
<td>MQIA_MAX_Q_DEPTH</td>
</tr>
<tr>
<td>NUMUNC</td>
<td>MQIA_MAX_UNCOMMITTED_MSGS</td>
</tr>
<tr>
<td>DELSEQ</td>
<td>MQIA_MSG_DELIVERY_SEQUENCE</td>
</tr>
<tr>
<td>MDC</td>
<td>MQIA_MSG_DEQ_COUNT</td>
</tr>
<tr>
<td>MEC</td>
<td>MQIA_MSG_ENQ_COUNT</td>
</tr>
<tr>
<td>NAMEC</td>
<td>MQIA_NAME_COUNT</td>
</tr>
<tr>
<td>IOC</td>
<td>MQIA_OPEN_INPUT_COUNT</td>
</tr>
<tr>
<td>OOC</td>
<td>MQIA_OPEN_OUTPUT_COUNT</td>
</tr>
<tr>
<td>PEV</td>
<td>MQIA_PERFORMANCE_EVENT</td>
</tr>
<tr>
<td>PLATFORM</td>
<td>MQIA_PLATFORM</td>
</tr>
<tr>
<td>QDHEV</td>
<td>MQIA_Q_DEPTH_HIGH_EVENT</td>
</tr>
<tr>
<td>QDHLM</td>
<td>MQIA_Q_DEPTH_HIGH_LIMIT</td>
</tr>
<tr>
<td>QDLEV</td>
<td>MQIA_Q_DEPTH_LOW_EVENT</td>
</tr>
<tr>
<td>QDLLIM</td>
<td>MQIA_Q_DEPTH_LOW_LIMIT</td>
</tr>
<tr>
<td>Item</td>
<td>PCF Parameter Name</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>QDMEV</td>
<td>MQIA_Q_DEPTH_MAX_EVENT</td>
</tr>
<tr>
<td>SI</td>
<td>MQIA_Q_SERVICE_INTERVAL</td>
</tr>
<tr>
<td>SIEV</td>
<td>MQIA_Q_SERVICE_INTERVAL_EVENT</td>
</tr>
<tr>
<td>QT</td>
<td>MQIA_Q_TYPE</td>
</tr>
<tr>
<td>REV</td>
<td>MQIA_REMOTE_EVENT</td>
</tr>
<tr>
<td>RETINT</td>
<td>MQIA_RETENTION_INTERVAL</td>
</tr>
<tr>
<td>SCOPE</td>
<td>MQIA_SCOPE</td>
</tr>
<tr>
<td>SHARE</td>
<td>MQIA_SHAREABILITY</td>
</tr>
<tr>
<td>SSEV</td>
<td>MQIA_START_STOP_EVENT</td>
</tr>
<tr>
<td>SYNC</td>
<td>MQIA_SYNCPOINT</td>
</tr>
<tr>
<td>TSR</td>
<td>MQIA_TIME_SINCE_RESET</td>
</tr>
<tr>
<td>TRIGC</td>
<td>MQIA_TRIGGER_CONTROL</td>
</tr>
<tr>
<td>TRIGD</td>
<td>MQIA_TRIGGER_DEPTH</td>
</tr>
<tr>
<td>TRIGI</td>
<td>MQIA_TRIGGER_INTERVAL</td>
</tr>
<tr>
<td>MSGPRI</td>
<td>MQIA_TRIGGER_MSG_PRIORITY</td>
</tr>
<tr>
<td>TRIT</td>
<td>MQIA_TRIGGER_TYPE</td>
</tr>
<tr>
<td>USAGE</td>
<td>MQIA_USAGE</td>
</tr>
<tr>
<td>ALL</td>
<td>MQIACF_ALL</td>
</tr>
<tr>
<td>AED1</td>
<td>MQIACF_AUX_ERROR_DATA_INT_1</td>
</tr>
<tr>
<td>AED2</td>
<td>MQIACF_AUX_ERROR_DATA_INT_2</td>
</tr>
<tr>
<td>BRTYPE</td>
<td>MQIACF_BRIDGE_TYPE</td>
</tr>
<tr>
<td>CATTR</td>
<td>MQIACF_CHANNEL_ATTRS</td>
</tr>
<tr>
<td>COM</td>
<td>MQIACF_COMMAND</td>
</tr>
<tr>
<td>CONVRC</td>
<td>MQIACF_CONV_REASON_CODE</td>
</tr>
<tr>
<td>EID</td>
<td>MQIACF_ERROR_ID</td>
</tr>
<tr>
<td>ERRORID</td>
<td>MQIACF_ERROR_IDENTIFIER</td>
</tr>
<tr>
<td>ERROROF</td>
<td>MQIACF_ERROR_OFFSET</td>
</tr>
<tr>
<td>ESCTYPE</td>
<td>MQIACF_ESCAPE_TYPE</td>
</tr>
<tr>
<td>FORCE</td>
<td>MQIACF_FORCE</td>
</tr>
<tr>
<td>OOPTS</td>
<td>MQIACF_OPEN_OPTIONS</td>
</tr>
<tr>
<td>PARMID</td>
<td>MQIACF_PARAMETER_ID</td>
</tr>
<tr>
<td>PROCATTR</td>
<td>MQIACF_PROCESS_ATTRS</td>
</tr>
<tr>
<td>PURGE</td>
<td>MQIACF_PURGE</td>
</tr>
<tr>
<td>QATTR</td>
<td>MQIACF_Q_ATTRS</td>
</tr>
<tr>
<td>QMATTR</td>
<td>MQIACF_Q_MGR_ATTRS</td>
</tr>
<tr>
<td>QUIESCE</td>
<td>MQIACF QUIESCE</td>
</tr>
<tr>
<td>RQUAL</td>
<td>MQIACF_REASON_QUALIFIER</td>
</tr>
<tr>
<td>.item</td>
<td>PCF Parameter Name</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>REPLACE</td>
<td>MQIACF_REPLACE</td>
</tr>
<tr>
<td>SELECTOR</td>
<td>MQIACF_SELECTOR</td>
</tr>
<tr>
<td>BATCHSZ</td>
<td>MQIACH_BATCH_SIZE</td>
</tr>
<tr>
<td>BATCHES</td>
<td>MQIACH_BATCHES</td>
</tr>
<tr>
<td>NBUFRCV</td>
<td>MQIACH_BUFFERS_RCVD</td>
</tr>
<tr>
<td>NBUFRCV</td>
<td>MQIACH_BUFFERS_RECEIVED</td>
</tr>
<tr>
<td>NBUFSENT</td>
<td>MQIACH_BUFFERS_SENT</td>
</tr>
<tr>
<td>NBYTERCVD</td>
<td>MQIACH_BYTES_RCVD</td>
</tr>
<tr>
<td>NBYTESENT</td>
<td>MQIACH_BYTES_RECEIVED</td>
</tr>
<tr>
<td>NBYTERCVD</td>
<td>MQIACH_BYTES_SENT</td>
</tr>
<tr>
<td>CED</td>
<td>MQIACH_CHANNEL_ERROR_DATA</td>
</tr>
<tr>
<td>CIATTR</td>
<td>MQIACH_CHANNEL_INSTANCE_ATTRS</td>
</tr>
<tr>
<td>CITYPE</td>
<td>MQIACH_CHANNEL_INSTANCE_TYPE</td>
</tr>
<tr>
<td>CSTATUS</td>
<td>MQIACH_CHANNEL_STATUS</td>
</tr>
<tr>
<td>CTABLE</td>
<td>MQIACH_CHANNEL_TABLE</td>
</tr>
<tr>
<td>CTYPE</td>
<td>MQIACH_CHANNEL_TYPE</td>
</tr>
<tr>
<td>CURMSG</td>
<td>MQIACH_CURRENT_MSGS</td>
</tr>
<tr>
<td>CURSEQ</td>
<td>MQIACH_CURRENT_SEQ_NUMBER</td>
</tr>
<tr>
<td>CURSEQ</td>
<td>MQIACH_CURRENT_SEQUENCE_NUMBER</td>
</tr>
<tr>
<td>CURDATAVC</td>
<td>MQIACH_DATA_CONVERSION</td>
</tr>
<tr>
<td>CURDATAIN</td>
<td>MQIACH_DATA_COUNT</td>
</tr>
<tr>
<td>CURDISCIN</td>
<td>MQIACH_DISC_INTERVAL</td>
</tr>
<tr>
<td>IND</td>
<td>MQIACH_IN_DOUBT</td>
</tr>
<tr>
<td>INDSEQ</td>
<td>MQIACH_INDOUBT_STATUS</td>
</tr>
<tr>
<td>LASTSEQ</td>
<td>MQIACH_LAST_SEQUENCE_NUMBER</td>
</tr>
<tr>
<td>LRTRYYL</td>
<td>MQIACH_LONG RETRIES_LEFT</td>
</tr>
<tr>
<td>LTRY</td>
<td>MQIACH_LONG_RETRY</td>
</tr>
<tr>
<td>LTIME</td>
<td>MQIACH_LONG_TIMER</td>
</tr>
<tr>
<td>MAXMSGL</td>
<td>MQIACH_MAX_MSG_LENGTH</td>
</tr>
<tr>
<td>MCAST</td>
<td>MQIACH_MCA_STATUS</td>
</tr>
<tr>
<td>MCTYPE</td>
<td>MQIACH_MCA_TYPE</td>
</tr>
<tr>
<td>MRCOUNT</td>
<td>MQIACH_MR_COUNT</td>
</tr>
<tr>
<td>MRINT</td>
<td>MQIACH_MR_INTERVAL</td>
</tr>
<tr>
<td>MSGSEQNO</td>
<td>MQIACH_MSG_SEQUENCE_NUMBER</td>
</tr>
<tr>
<td>MSGS</td>
<td>MQIACH_MSGS</td>
</tr>
<tr>
<td>PUTAUTH</td>
<td>MQIACH_PUT_AUTHORITY</td>
</tr>
<tr>
<td>SEQNW</td>
<td>MQIACH_SEQUENCE_NUMBER_WRAP</td>
</tr>
</tbody>
</table>
Table 8 (Page 6 of 6). Event Names

<table>
<thead>
<tr>
<th>.item</th>
<th>PCF Parameter Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRTRYL</td>
<td>MQIACH_SHORT_RETRIES_LEFT</td>
</tr>
<tr>
<td>SRTRY</td>
<td>MQIACH_SHORT_RETRY</td>
</tr>
<tr>
<td>STIME</td>
<td>MQIACH_SHORT_TIMER</td>
</tr>
<tr>
<td>STOPREQ</td>
<td>MQIACH_STOP_REQUESTED</td>
</tr>
<tr>
<td>XPT</td>
<td>MQIACH_XMIT_PROTOCOL_TYPE</td>
</tr>
<tr>
<td>NA</td>
<td>MQIAV_NOT_APPLICABLE</td>
</tr>
</tbody>
</table>

**Note:**
- These are the .Component names setup for the PCF Fields within the Event message
- .REA and .NAME are set for all events
- .TYPE is set to 'EVENT' for all events
- .EID .AEDI1 .AEDI2 and .CED are returned in Hex
- .XQN is generated from two parameters
## Components and Events

Table 9. Events and Components

| Event Name | QMACT | QMINA | INPUT | LLAOQ | LLARQ | LLUON | CCHQNY | CCHSRK | CCHSTOP | PPFQH | PPFQUL | PPFQFU | PPFQSO | RMXDQT | RMXDQU | RMXQT | RMXQAL | RMXQAN | RMXQOS | RMXQDQ | RMXQDM | RMXQNM | RMUXQ | RMXUQ | NAUT | CHACT | CCHACT | BRSRKT | BRSTOP |
|------------|-------|-------|-------|-------|-------|-------|--------|--------|--------|-------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| AEDI1      | ✓     |       | ✓     |       |       |       |        |        |        |       |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| AEDI2      | ✓     |       | ✓     |       |       |       |        |        |        |       |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| AEDS1      | ✓     |       |       |       |       |       |        |        |        |       |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| AEDS2      | ✓     |       |       |       |       |       |        |        |        |       |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| AEDS3      | ✓     |       |       |       |       |       |        |        |        |       |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| AN         | ✓     | ✓     | ✓     | ✓     | ✓     | ✓     | ✓      | ✓      | ✓      | ✓     | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      |
| AT         | ✓     | ✓     | ✓     | ✓     | ✓     | ✓     | ✓      | ✓      | ✓      | ✓     | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      |
| BRNAME     | ✓     | ✓     |       |       |       |       |        |        |        |       |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| BRTYPE     | ✓     | ✓     |       |       |       |       |        |        |        |       |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| BQN        | ✓     | ✓     |       |       |       |       |        |        |        |       |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| CED        | ✓     |       |       |       |       |       |        |        |        |       |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| CN         | ✓     | ✓     | ✓     | ✓     | ✓     | ✓     | ✓      | ✓      | ✓      | ✓     | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      |
| COM        | ✓     |       |       |       |       |       |        |        |        |       |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| CONN       | ✓     | ✓     | ✓     | ✓     | ✓     | ✓     | ✓      | ✓      | ✓      | ✓     | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      |
| CONVRC     | ✓     |       |       |       |       |       |        |        |        |       |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| EID        | ✓     |       |       |       |       |       |        |        |        |       |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| FORMAT     | ✓     |       |       |       |       |       |        |        |        |       |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| HQD        | ✓     | ✓     | ✓     | ✓     | ✓     | ✓     | ✓      | ✓      | ✓      | ✓     | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      |
| MDC        | ✓     | ✓     | ✓     | ✓     | ✓     | ✓     | ✓      | ✓      | ✓      | ✓     | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      |
| MEC        | ✓     | ✓     | ✓     | ✓     | ✓     | ✓     | ✓      | ✓      | ✓      | ✓     | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      |
| OOPTS      | ✓     |       |       |       |       |       |        |        |        |       |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| OQM        | ✓     | ✓     | ✓     | ✓     | ✓     | ✓     | ✓      | ✓      | ✓      | ✓     | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      |
| PN         | ✓     |       |       |       |       |       |        |        |        |       |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| QM         | ✓     | ✓     | ✓     | ✓     | ✓     | ✓     | ✓      | ✓      | ✓      | ✓     | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      |
| QN         | ✓     | ✓     | ✓     | ✓     | ✓     | ✓     | ✓      | ✓      | ✓      | ✓     | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      |
| QT         | ✓     |       |       |       |       |       |        |        |        |       |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| RQUAL      | ✓     |       |       |       |       |       |        |        |        |       |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| TSR        | ✓     | ✓     | ✓     | ✓     | ✓     | ✓     | ✓      | ✓      | ✓      | ✓     | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      |
| UID        | ✓     |       |       |       |       |       |        |        |        |       |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| XQN        | ✓     | ✓     | ✓     | ✓     | ✓     | ✓     | ✓      | ✓      | ✓      | ✓     | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      | ✓      |

Note:
- `.QM .REA` and `.NAME` are set for all events
- `.TYPE` is set to `EVENT` for all events
- `.EID .AEDI1 .AEDI2` and `.CED` are returned in Hex
- `.CED` may contain a number of elements; in this case `.CED.0` contains the number of elements with `.CED.n` the actual data
- `.ZLIST` contains a list of all the present component names
Example

```rexx
/* A message has been obtained such that ... */
message.O = n
message.1 = <EVENT Header><Event Data>

/* Clear the result variable */
drop x.

/* Split the message */
rcc = RXMQV('EVENT', 'message.', 'x. ')

/* on return, the following (and more) are set */
say x.TYPE       /* EVENT */
say x.NAME       /* CHSTOP */
say x.QM         /* RAH2 */
say x.CN         /* C4T036N */
say x.XQN        /* T4T036N */
```

This example shows how a message obtained from SYSTEM.ADMIN.CHANNEL.EVENT is split up, showing the information relating to the Channel Stop Event.

See Figure 6 on page 15 for an example using ZLIST processing to cope with the variable format component names.

```rexx
/*Explurge an Event */
message.O = n
message.1 = <EVENT Header><Event Data>
drop x.
rcc = RXMQV('EVENT', 'message.', 'x. ')

/* Testing the returned information */
say x.TYPE    /* EVENT */
say x.NAME    /* INGET */
say x.REA     /* 2016 */
if ( x.at <> 'X.QN' ) then x.qn /* works - returned comp */
if ( x.BQN <> 'X.BQN' ) then x.bqn /* fails - not in event */
```

This example shows how the components of an exploded Event message can be tested to fully extract all the returned information if ZLIST processing is not used.
ZLIST processing is also useful to cope with situations were an event String Field is defined, but set to all binary zeros. These can easily be changed into blanks (with space truncation) as follows:

```rexx
message.0 = n
message.1 = 'EVENT'<Event Data>
drop x.
    rcc = RXMQV('EVENT', 'message.', 'x.')
    do i=1 to words(x.zlist)
      ts = word(x.zlist,i)
      x.ts = translate(x.ts,' ','00'x)
      x.ts = strip(x.ts,'B')
    end
```

Figure 7. Removing funny event data
RXMQVTM

Description

This call will take a message obtained from an Initiation Queue (a Trigger Message) and split it up into its components. It will also parse the data passed to a started Rexx Exec (via a MQ Trigger Monitor).

This processing, therefore, permits the obtention of the control information associated with a Trigger: whether this is in the format of a MQ Message (garnered from an Initiation Queue) or passed as parameters to a Rexx Exec (as the Triggered Process).

The action of this function is controlled by the format of its first parameter, in particular whether or not it ends in a dot.

- If it ends in a dot, then RXMQVTM is processing a message derived from an Initiation Queue.
  
  The message to be processed is specified in the usual way as the name of a stem. variable; with component 0 representing the length of the message which is supplied in component 1. See “Message Lengths” on page 12 for a warning about truncated messages used with this function.
  
  This is called Message Mode.

- If it does not end in a dot, then RXMQVTM is processing the parameter data passed via a Trigger Monitor to the Rexx Exec which is acting as a Triggered Process (ie: replaces the initial parse arg processing). It is the actual data, not a variable name that is supplied (ie: a substituted variable, not the variable name).
  
  This is called Data Mode.

The Extracted data is placed in another stem. variable (whose name is supplied); with components representing the various sub-fields of the Trigger Message or Trigger parms.

Sub-fields which are all blanks (or start with a Binary Zero) are not extracted. ZLIST processing (see “ZLIST” on page 14) is provided so that the various extant components can be determined.

In Message Mode (a Trigger Message provided to RXMQVTM in a Stem. variable) an additional component (not in ZLIST) called PL is provided which is the Parameter list for a process to be invoked by the reception of the Trigger Message in the Initiation Queue (if the current thread is connected to a Queue Manager, its name will be present in .PL). You should ensure that this component is not truncated in any way (as this will may well effect the activity of the process which uses it).

You can, therefore, use a Rexx Exec as the Triggered Process, extracting the supplied information using RXMQVTM in Data Mode.

The use of Message Mode permits the coding of your own Trigger Monitor (recall the Trigger Messages only get placed in an Initiation Queue if the priorities are right, the process exists, and the Initiation Queue is Open for Getting) in Rexx (see Figure 8 on page 78), and Data Mode permits the use of Rexx Execs as Triggered Processes (see Figure 9 on page 79).
Parameters

1. This parameter takes one of these formats:

   **In Message Mode**
   The name of a Rexx Stem variable (including the dot) containing a message to be splitup. This is an input parameter. Upon the call, Component 0 must contain the length of the message in Component 1; the message must have been obtained from an Initiation Queue. See “Message Lengths” on page 12 for a warning about truncation.

   **In Data Mode**
   The actual data (not a variable name) representing the MQTMC2 structure which is used to initiate a Triggered Process.

2. The name of a Rexx Stem variable (including the dot) into which the extracted data will be placed. This is an input/output parameter. After the call, components will be created (as documented in Table 10 on page 77) to return the extracted information. ZLIST processing is provided for this Stem variable. In the case of Message Mode, component PL will contain an area suitable for use by a Triggered Process as its parameters.

Call

```
Message Mode:
   rcc = RXMQV('TM', 'Stem.Message.', 'Stem.Splitup.' )

Data Mode:
   rcc = RXMQV('TM', MQTMC2_data , 'Stem.Splitup.' )
```
Additional Interface Return Codes and Messages

-1 0 0 RXMQVTMM Bad number ofParms
Explanation: You must specify two parameters to the RXMQVTM call.

-2 0 0 RXMQVTMM Input Variable name/data not supplied
Explanation: No value has been keyed for the first parameter, the name of a Stem. variable representing the message to be split up or data representing a MQTMC2 structure to be parsed.

-3 0 0 RXMQVTMM Input data parm is too big
Explanation: The length of the input data was larger than that permitted for a Trigger Message

-4 0 0 RXMQVTMM Output Stem. Var name not supplied
Explanation: No value has been keyed for the second parameter, the name of a Stem. variable representing the split up data.

-5 0 0 RXMQVTMM No Data for TM Extraction
Explanation: The input Stem.0 was zero, indicating no message to process (message mode)

-6 0 0 RXMQVTMM Message is too short for a TM
Explanation: The length of the input Stem.1 was <= 3, indicating no header in the message (message mode)

-7 0 0 RXMQVTMM Message is too long for a TM
Explanation: The length of the input data was larger than that permitted for a Trigger Message (message mode)

-8 0 0 RXMQVTMM Message is not a TM
Explanation: The first 4 bytes of the input Stem.1 or data was not 'TM ', so the message did not come from an Initiation Queue or a Triggered Process' parameter, and so cannot be split up (message mode)

-9 0 0 RXMQVTMM No Data for TM Extraction
Explanation: The input Stem.0 was zero, indicating no message to process (data mode)

-10 0 0 RXMQVTMM Message is too short for a TM
Explanation: The length of the input Stem.1 was <= 3, indicating no header in the message (data mode)

-11 0 0 RXMQVTMM Message is too long for a TM
Explanation: The length of the input data was larger than that permitted for a Trigger Message (data mode)

-12 0 0 RXMQVTMM Message is not a TM
Explanation: The first 4 bytes of the input Stem.1 or data was not 'TM ', so the message did not come from an Initiation Queue or a Triggered Process' parameter, and so cannot be split up (data mode)
## Trigger information

### Table 10. Trigger Components

<table>
<thead>
<tr>
<th>Stem Component</th>
<th>MQTM/MQTMC2 Structure name</th>
<th>Number or Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>.QN</td>
<td>QName</td>
<td>T</td>
</tr>
<tr>
<td>.PN</td>
<td>ProcessName</td>
<td>T</td>
</tr>
<tr>
<td>.TD</td>
<td>TriggerData</td>
<td>T</td>
</tr>
<tr>
<td>.AT</td>
<td>ApplType</td>
<td>N</td>
</tr>
<tr>
<td>.AID</td>
<td>ApplId</td>
<td>T</td>
</tr>
<tr>
<td>.ED</td>
<td>EnvData</td>
<td>T</td>
</tr>
<tr>
<td>.UD</td>
<td>UserData</td>
<td>T</td>
</tr>
<tr>
<td>.QM</td>
<td>QMgrName</td>
<td>T</td>
</tr>
<tr>
<td>.PL</td>
<td>MQTMC2 parameter</td>
<td>T</td>
</tr>
</tbody>
</table>

**Note:**
- Number or Text shows the type of the field
- Text items which are all Blanks (or start with a Binary Zero) are not generated
- .AT and .PL are only available in Message Mode
- .QM is only available in Data Mode
- .ZLIST processing is available for QN, PN, TD, AT, AID, ED, UD & QM if they are generated.
- PL is not placed in ZLIST
Examples

/* A message has been obtained from an Initiation Queue */

message.O = 684
message.I = <MQTM>

/* Clear the result variable */
drop t.

/* Split the message */
rcc = RXMQV('TM', 'message.', 't.' )

/* on return, the following are set */
say t.QN /* L3N1 */
say t.PN /* P3T046N */

/* Truncated non-parm areas for usage */
do j=1 to words(t.zlist)
   item = word(t.zlist,j)
   t.item = strip(t.item,'B')
end

/* Some processing to decide on something to do */

/* Start a Process to service the Queue */
'someproc' t.pl

exit

Figure 8. A Trigger Monitor

This example shows how a message obtained from an Initiation Queue is split up, showing how the PL component is used to start a process to service the Queue which generated the Trigger. Note that all the parameters passed in the Message can be used however one wants when one codes one’s own Trigger Monitor.
Figure 9. A Rexx Triggered Process

This example shows how a Rexx Exec being initiated via a Trigger Monitor accesses its passed data.
Chapter 8. Interface Example

This example shows the use of all of the functions in the interface. It uses a Queue Manager called **VRH1** and Queues **N1** and **P1**. This exec is provided within this SupportPac in the MA18TEST JCL file.

```rexx
/* MA18TEST Exec - a Rexx/MQ/MVS Example */
/* Initialise the interface */
RMQVTRACE = ''
rc = RMQV('INIT')
say 'rc=' rc

/* Connect to Queue Manager - VRH1 */
RMQVTRACE = ''
rc = RMQV('CONN', 'VRH1')
say 'RC=' rc

/* Open Queue N1 for Inquire Access Only, tracing Object Descriptor accesses */
iod.on = 'N1'
iod.ot = MQOT_Q
RMQVTRACE = 'BOD MOD'
rc = RMQV('OPEN', 'iod.', mqqo_inquire, 'h1', 'ood.')
say 'RC=' rc 'H=' h1

/* Open Queue P1 for Output and Browse Access, plus Attribute manipulation */
RMQVTRACE = ''
oo = mqqo_inquire+mqqo_output+mqqo_browse+mqqo_set
rc = RMQV('OPEN', 'P1', oo , 'h2', 'ood.')
say 'RC=' rc 'H=' h2

/* Write a Persistent Message, within UOW, to Queue P1; trace everything */
RMQVTRACE = 'PUT BMD MMD MPO BPO'
d.0 = 20
d.1 = time() '0123456789'
imd.PER = MQPER_PERSISTENT
ipmo.opt = MQPMO_SYNCPOINT
rc = RMQVPUT( h2,'d.','imd.','omd.','ipmo.','opmo.')
say 'RC=' rc
```

Figure 10 (Part 1 of 4). Interface example
/* Inquire upon the number of Messages now in Queue P1 */

RXMLQVTRACE = ''
atri = mqia_current_q_depth
atrou = ''
rcc  = RXMLQV('INQ', h2, atrin, 'atrou')
say 'RC=' rcc 'Atr' atrin 'Setting <atrou>'

/* Show the name of the Queue which is using handle 1 */

RXMLQVTRACE = ''
atri = mqca_q_name
atrou = ''
rcc  = RXMLQV('INQ', h1, atrin, 'atrou')
say 'RC=' rcc 'Atr' atrin 'Setting <atrou>'

/* Toggle the GETtability of a Queue, providing a change each time */

RXMLQVTRACE = ''
atrsn = MQIA_INHIBIT_GET
atrsd = MQQA_GET_INHIBITED
rcc  = RXMLQV('SET', h2, atrsn, atrsd)
say 'RC=' rcc

RXMLQVTRACE = ''
atri = mqia_inhibit_get ; atrou = ''
rcc  = RXMLQV('INQ', h2, atrin, 'atrou')
say 'RC=' rcc 'Atr' atrin 'Setting <atrou>'

RXMLQVTRACE = ''
atrsn = MQIA_INHIBIT_GET
atrsd = MQQA_GET_ALLOWED
rcc  = RXMLQV('SET', h2, atrsn, atrsd)
say 'RC=' rcc

RXMLQVTRACE = ''
atri = MQIA_INHIBIT_GET ; atrou = ''
rcc  = RXMLQV('INQ', h2, atrin, 'atrou')
say 'RC=' rcc
say 'Atr' atrin 'Setting <atrou>'

Figure 10 (Part 2 of 4). Interface example
/* Set the Trigger Data for a Queue ... */

RXMLQVTRACE = ''
atrns = MQCA_TRIGGER_DATA
atrsd = 'RAH Trigger Data'
rc = RXMLQV('SET', h2, atrns, atrsd)
say 'RC=' rc

/* ... and show that it's worked */

RXMLQVTRACE = ''
atrin = mqca_trigger_data
atrou = ''
rc = RXMLQV('INQ', h2, atrin, 'atrou')
say 'RC=' rc 'Atr' atrin 'Setting '<'atrou'>'

/* Syncpoint all accesses to the QM */

RXMLQVTRACE = ''
rc = RXMLQV('CMIT')
say 'RC=' rc

/* Browse all messages on queue P1, tracing everything, and showing updates */

RXMLQVTRACE = 'BGO MGO MMD BMD GET'
do i=1
  g.0 = 200
  g.1 = ''
  igmo.opt = MQGMO_WAIT+MQGMO_BROWSE_NEXT
  rc = RXMLQV('GET', h2,'g.','igmd.','ogmd.','igmo.','ogmo.')
say 'RC=' rc
  say '..................' i 'data '<g.1> length' g.0
  say 'ogmd.pd' ogmd.pd 'ogmo.rqn'<ogmo.rqn>'
  if ( word(rc,1) <> 0 ) then leave
end

/* Rollback a Unit of Work (empty in this case) */
RXMLQVTRACE = ''
rc = RXMLQV('BACK')
say 'RC=' rc

/* Stop access to a Queue */

RXMLQVTRACE = ''
rc = RXMLQV('CLOSE', h2, mqco_none)
say 'RC=' rc

Figure 10 (Part 3 of 4). Interface example
/* Re-open the P1 Queue for Browse only access */

RXMQVTRACE = ''
rc = RXMQV('OPEN', 'P1', mqoo_browse, 'h3', 'ood.')</code>
say 'RC= ' rc ' H= ' h3

/* Browse the Queue using the Extension function */

RXMQVTRACE = ''
do i=1
g.0 = 200
g.1 = ''
rc = RXMQV('BROWSE', h3, g.1)
say 'RC= ' rc
say '0000000000000' i 'data = 'g.1' length'= g.0
if (word(rc,1) <> 0) then leave
end

/* Show the last command used etc. */
say 'Last Message = 'RXMQV.LASTMSG' Last call 'RXMQV.LASTOP,
' which ended with RC('RXMQV.LASTRC') and MQCC('RXMQV.LASTCC')',
' MQRC('RXMQV.LASTAC')''

/* Issue a Bad command to show effect of -ve RC */

RXMQVTRACE = ''
rc = RXMQV('OPEN')
say 'Last Message = 'RXMQV.LASTMSG' Last call 'RXMQV.LASTOP,
' which ended with RC('RXMQV.LASTRC') and MQCC('RXMQV.LASTCC')',
' MQRC('RXMQV.LASTAC')''

/* Stop access to the Queue */

RXMQVTRACE = ''
rc = RXMQV('CLOSE', h3, mqco_none)
say 'RC= ' rc

/* Disconnect from the QM (Closing h1 in the process) */

RXMQVTRACE = ''
rc = RXMQV('DISC')
say 'RC= ' rc

/* Remove the Interface functions from the Rexx Workspace ... */

RXMQVTRACE = 'TERM'
rc = RXMQV('TERM')
say 'RC= ' rc

/* ... but leave the MQ_constants around */
say 'MQPER_PERSISTENT' MQPER_PERSISTENT 'RC(2048) is' RXMQV.RCMAP.2048

/* End of MA18TEST exec */

Figure 10 (Part 4 of 4). Interface example
Appendix A. Rexx/MQ constants

MQ_ACCOUNTING_TOKEN_LENGTH
MQ_APPL_IDENTITY_DATA_LENGTH
MQ_APPL_NAME_LENGTH
MQ_APPL_ORIGIN_DATA_LENGTH
MQ_AUTHENTICATOR_LENGTH
MQ_BRIDGE_NAME_LENGTH
MQ_CHANNEL_DATE_LENGTH
MQ_CHANNEL_DESC_LENGTH
MQ_CHANNEL_NAME_LENGTH
MQ_CHANNEL_TIME_LENGTH
MQ_CONN_NAME_LENGTH
MQ_CORREL_ID_LENGTH
MQ_CREATION_DATE_LENGTH
MQ_CREATION_TIME_LENGTH
MQ_EXIT_DATA_LENGTH
MQ_EXIT_NAME_LENGTH
MQ_EXIT_USER_AREA_LENGTH
MQ_FORMAT_LENGTH
MQ_LTERM_OVERRIDE_LENGTH
MQ_LUWID_LENGTH
MQ_MCA_JOB_NAME_LENGTH
MQ_MCA_NAME_LENGTH
MQ_MCA_USER_DATA_LENGTH
MQ_MODE_NAME_LENGTH
MQ_MSG_HEADER_LENGTH
MQ_MSG_ID_LENGTH
MQ_NAMELIST_DESC_LENGTH
MQ_NAMELIST_NAME_LENGTH
MQ_PASSWORD_LENGTH
MQ_PROCESS_APPL_ID_LENGTH
MQ_PROCESS_DESC_LENGTH
MQ_PROCESS_ENV_DATA_LENGTH
MQ_PROCESS_NAME_LENGTH
MQ_PROCESS_USER_DATA_LENGTH
MQ_PROGRAM_NAME_LENGTH
MQ_PUT_APPL_NAME_LENGTH
MQ_PUT_DATE_LENGTH
MQ_PUT_TIME_LENGTH
MQ_Q_DESC_LENGTH
MQ_Q_MGR_DESC_LENGTH
MQ_Q_MGR_NAME_LENGTH
MQ_Q_NAME_LENGTH
MQ_SHORT_CONN_NAME_LENGTH
MQ_STORAGE_CLASS_LENGTH
MQ_TP_NAME_LENGTH
MQ_TRAN_INSTANCE_ID_LENGTH
MQ_TRIGGER_DATA_LENGTH
MQ_USER_ID_LENGTH
MQ_AIX
MQ_CICS
MQ_CICS_VSE
MQ_DEFAULT
MQ_DOS
MQ_GUARDIAN
MQ_IMS
MQ_IMS_BRIDGE
MQ_MVS
MQ_NO_CONTEXT
MQ_OS2
MQ_OS400
MQ_QMGR
MQ_UNIX
MQ_UNKNOWN
MQ_AT_BACKOUT_REQ_Q_NAME
MQ_AT_BASE_Q_NAME
MQ_AT_COMMAND_INPUT_Q_NAME
MQ_AT_CREATION_DATE
MQ_AT_CREATION_TIME
MQ_AT_DEAD_LETTER_Q_NAME
MQ_AT_DEF_XMIT_Q_NAME
MQ_AT_ENV_DATA
MQ_AT_FIRST
MQ_AT_INITIATION_Q_NAME
MQ_AT_LAST
MQ_AT_LAST_USED
MQ_AT_NAMELIST_DESC
MQ_AT_NAMELIST_NAME
MQ_AT_NAMES
MQ_AT_PROCESS_DESC
MQ_AT_PROCESS_NAME
MQ_AT_Q_DESC
MQ_AT_Q_MGR_DESC
MQ_AT_Q_MGR_NAME
MQ_AT_Q_NAME
MQ_AT_REMOTE_Q_MGR_NAME
MQ_AT_REMOTE_Q_NAME
MQ_AT_STORAGE_CLASS
MQ_AT_TRIGGER_DATA
MQ_AT_USER_DATA
MQ_AT_XMIT_Q_NAME
MQ_AT_ALIAS_Q_NAMES
MQ_AT_APPL_NAME
MQ_AT_AUX_ERROR_DATA_STR_1
MQ_AT_AUX_ERROR_DATA_STR_2
MQ_AT_AUX_ERROR_DATA_STR_3
MQ_AT_BRIDGE_NAME
MQ_AT_ESCAPE_TEXT
MQ_AT_FIRST
MQ_AT_FROM_CHANNEL_NAME
MQ_AT_FROM_Q_NAME
MQ_AT_LAST_USED
MQ_AT_LOCAL_Q_NAMES
MQ_AT_MODEL_Q_NAMES
MQ_AT_OBJECT_Q_MGR_NAME
MQ_AT_RECEIVER_CHANNEL_NAMES
MQ_AT_SENDER_CHANNEL_NAMES
MQ_AT_SERVER_CHANNEL_NAMES
MQ_AT_CHANNEL_NAME
MQ_AT_Q_MGR_NAME
MQ_AT_CHANNEL_NAMES
MQ_AT_CHANNEL_NAME
MQ_AT_Q_MGR_NAME
MQ_AT_CHANNEL_NAMES
MQFB_APPL_LAST
MQFB_APPL_TYPE_ERROR
MQFB_BUFFER_OVERFLOW
MQFB_CHANNEL_COMPLETED
MQFB_CHANNEL_FAIL
MQFB_CHANNEL_FAIL_RETRY
MQFB_COA
MQFB_COD
MQFB_DATA_LENGTH_NEGATIVE
MQFB_DATA_LENGTH_TOO_BIG
MQFB_DATA_LENGTH_ZERO
MQFB_EXPIRATION
MQFB_IIH_ERROR
MQFB_IMS_ERROR
MQFB_IMS_FIRST
MQFB_IMS_LAST
MQFB_LENGTH_OFF_BY_ONE
MQFB_NONE
MQFB_NOT_AUTHORIZED_FOR_IMS
MQFB_QUIT
MQFB_STOPPED_BY_MSG_EXIT
MQFB_SYSTEM_FIRST
MQFB_SYSTEM_LAST
MQFB_TM_ERROR
MQFB_XMIT_Q_MSG_ERROR
MQFC_NO
MQFC_YES
MQFMT_ADMIN
MQFMT_CHANNEL_COMPLETED
MQFMT_COMMAND_1
MQFMT_COMMAND_2
MQFMT_DEAD_LETTER_HEADER
MQFMT_EVENT
MQFMT_IMS
MQFMT_IMS_VAR_STRING
MQFMT_PCF
MQFMT_STRING
MQFMT_TRIGGER
MQFMT_XMIT_Q_HEADER
MQGMO_ACCEPT_TRUNCATED_MSG
MQGMO_ACCEPT_TRUNCATED_MSG
MQGMO_BROWSE_FIRST
MQGMO_BROWSE_NEXT
MQGMO_CONVERT
MQGMO_FAIL_IF_QUIESCING
MQGMO_MARK_SKIP_BACKOUT
MQGMO_MSG_UNDER_CURSOR
MQGMO_NO_SYNCPOINT
MQGMO_NO_WAIT
MQGMO_NONE
MQGMO_PURGE
MQGMO_SET_SIGNAL
MQGMO_SYNCPOINT
MQGMO_TRIGGER
MQGMO_XMIT_Q_HEADER
MQIA_APPL_TYPE
MQIA_AUTHORITY_EVENT
MQIA_BACKOUT_THRESHOLD
MQIA_CODING_LEVEL
MQIA_COMMAND_LEVEL
MQIA_CIPHER_LEVEL
MQIA_CURRENT_Q_DEPTH
MQIA_DEF_INPUT_OPEN_OPTION
MQIA_DEF_PERSISTENCE
MQIA_DEF_PRIORITY
MQIA_DEFINITION_TYPE
MQIA_FIRST
MQIA_HARDEN_GET_BACKOUT
MQIA_HIGH_Q_DEPTH
MQIA_INHIBIT_EVENT
MQIA_INHIBIT_GET
MQIA_INHIBIT_PUT
MQIA_LAST
MQIA_LAST_USED
MQIA_LOCAL_EVENT
MQIA_MAX_HANDLES
MQIA_MAX_MSG_LENGTH
MQIA_MAX_PRIORITY
MQIA_MAX_Q_DEPTH
MQIA_MAX_UNCOMMITTED_MSGS
MQIA_MSG_DELIVERY_SEQUENCE
MQIA_MSG_ENQ_COUNT
MQIA_NAME_COUNT
MQIA_OPEN_INPUT_COUNT
MQIA_OPEN_OUTPUT_COUNT
MQIA_PERFORMANCE_EVENT
MQIA_PLATFORM
MQIA_Q_DEPTH_HIGH_EVENT
MQIA_Q_DEPTH_LOW_EVENT
MQIA_Q_DEPTH_MAX_EVENT
MQIA_Q_SERVICE_INTERVAL
MQIA_Q_SERVICE_INTERVAL_EVENT
MQIA_Q_TYPE
MQIA_REMOTE_EVENT
MQIA_RETENTION_INTERVAL
MQIA_SCOPE
MQIA_SHAREABILITY
MQIA_START_STOP_EVENT
MQIA_SYNCPOINT
MQIA_TIME_SINCE_RESET
MQIA_TRIGGER_CONTROL
MQIA_TRIGGER_DEPTH
MQIA_TRIGGER_INTERVAL
MQIA_TRIGGER_MSG_PRIORITY
MQIA_TRIGGER_TYPE
MQIA_USAGE
MQIACF_ALL
MQIACF_AUX_ERROR_DATA_INT_1
MQIACF_AUX_ERROR_DATA_INT_2
MQIACF_BRIDGE_TYPE
MQIACF_CHANNEL_ATTRS
MQIACF_COMMAND
MQIACF_CONV_REASON_CODE
MQIACF_ERROR_ID
MQIACF_ERROR_IDENTIFIER
MQIACF_ERROR_OFFSET
MQIACF_ESCAPE_TYPE
MQIACF_FIRST
MQIACF_FORCE
MQIACF_LAST_USED
MQIACF_OPEN_OPTIONS
MQIACF_PARAMETER_ID
MQIACF_PROCESS_ATTRS
MQIACF_Purge
MQIACF_Q_ATTRS
MQIACF_Q_MGR_ATTRS
MQIACF_REPLACE
MQIACF_QUESCE
MQIACF_REASON_QUALIFIER
MQIACF_REPLACE
MQIACF_SELECTOR
MQIACF_BATCH_SIZE
MQIACF_BATCHES
MQIACF_BUFFERS_RCVD
MQQSIE_HIGH
MQQSIE_NONE
MQQSIE_OK
MQQT_ALIAS
MQQT_ALL
MQQT_LOCAL
MQQT_MODEL
MQQT_REMOTE
MQRO_ACCEPT_UNSUP_IF_XMIT_MASK
MQRO_ACCEPT_UNSUP_MASK
MQRO_COA
MQRO_COA_WITH_DATA
MQRO_COA_WITH_FULL_DATA
MQRO_COD
MQRO_COD_WITH_DATA
MQRO_COD_WITH_FULL_DATA
MQRO_COPY_MSG_ID_TO_CORREL_ID
MQRO_DEADLETTER_Q
MQRO_DISCARD_MSG
MQRO_EXCEPTION
MQRO_EXCEPTION_WITH_DATA
MQRO_EXCEPTION_WITHFULL_DATA
MQRO_EXPIRATION
MQRO_EXPIRATION_WITH_DATA
MQRO_EXPIRATION_WITHFULL_DATA
MQRO_NEW_MSG_ID
MQRO_NONE
MQRO_PASS_CORREL_ID
MQRO_PASS_MSG_ID
MQRP_NO
MQRP.YES
MQRQ_BRIDGE_STOPPED_ERROR
MQRQ_BRIDGE_STOPPED_OK
MQRQ_CHANNEL_STOPPED_DISABLED
MQRQ_CHANNEL_STOPPED_ERROR
MQRQ_CHANNEL_STOPPED_OK
MQRQ_CHANNEL_STOPPED_RETRY
MQRQ_CLOSE_NOTAUTHORIZED
MQRQ_CMD_NOTAUTHORIZED
MQRQ_CONN_NOTAUTHORIZED
MQRQ_OPEN_NOTAUTHORIZED
MQRQ_Q_MGRQUIESCING
MQRQ_Q_MGR_STOPPING
MQSCO_CELL
MQSCO_Q_MGR
MQSP_AVAILABLE
MQSP_NOTAVAILABLE
MQTC_OFF
MQTC_ON
MQTM_VERSION_1
MQUS_NORMAL
MQUS_TRANSMISSION
MQWI_UNLIMITED
MQXQH_VERSION_1
MQACT_NONE
MQCI_NONE
MQITII_NONE
MQMI_NONE
Appendix B. Rexx/MQ Return Code constants

MQRC_OK
MQRC_NONE
MQRC_WARNING
MQRC_FAILED
MQRC_UNKNOWN
MQRC_ADAPTER_CONN_LOAD_ERROR
MQRC_ADAPTER_CONV_LOAD_ERROR
MQRC_ADAPTER_DEFS_LOAD_ERROR
MQRC_ADAPTER_DISC_LOAD_ERROR
MQRC_ADAPTER_NOT_AVAILABLE
MQRC_ADAPTER_SERV_LOAD_ERROR
MQRC_ADAPTER_STORAGE_SHORTAGE
MQRC_ALIAS_BASE_Q_TYPE_ERROR
MQRC_ALREADY_CONNECTED
MQRC_ANOTHER_Q_MGR_CONNECTED
MQRC_API_EXIT_LOAD_ERROR
MQRC_API_EXIT_NOT_FOUND
MQRC_ASID_MISMATCH
MQRC_BACKED_OUT
MQRC_BRIDGE_STARTED
MQRC_BRIDGE_STOPPED
MQRC_BUFFER_ERROR
MQRC_BUFFER_LENGTH_ERROR
MQRC_CALL_IN_PROGRESS
MQRC_CHANNEL_ACTIVATED
MQRC_CHANNEL_CONV_ERROR
MQRC_CHANNEL_NOT_ACTIVATED
MQRC_CHANNEL_STARTED
MQRC_CHANNEL_STOPPED
MQRC_CHAR_ATTR_LENGTH_ERROR
MQRC_CHAR_ATTRS_ERROR
MQRC_CHAR_ATTRS_TOO_SHORT
MQRC_CICS_WAIT_FAILED
MQRC_COD_NOT_VALID_FOR_XCF_Q
MQRC_CONN_ID_IN_USE
MQRC_CONNECTION_BROKEN
MQRC_CONNECTION_NOT_AUTHORIZED
MQRC_CONNECTION_QUIESCING
MQRC_CONNECTION_STOPPING
MQRC_CONTEXT_HANDLE_ERROR
MQRC_CONTEXT_NOT_AVAILABLE
MQRC_CONVERTED_MSG_TOO_BIG
MQRC_CORREL_ID_ERROR
MQRC_DATA_LENGTH_ERROR
MQRC_DBCS_ERROR
MQRC_DEF_XMIT_Q_TYPE_ERROR
MQRC_DEF_XMIT_Q_USAGE_ERROR
MQRC_DUPLICATE_RECOV_COORD
MQRC_DYNAMIC_Q_NAME_ERROR
MQRC_ENVIRONMENT_ERROR
MQRC_EXPIRY_ERROR
MQRC_FEEDBACK_ERROR
MQRC_FILE_NOT_AUDITED
MQRC_FILE_SYSTEM_ERROR
MQRC_FORMAT_ERROR
MQRC_FUNCTION_ERROR
MQRC_GET_INHIBITED
MQRC_GMO_ERROR
MQRC_HANDLE_NOT_AVAILABLE
MQRC_HCONFIG_ERROR
MQRC_HCONN_ERROR
MQRC_HOBJ_ERROR
MQRC_INHIBIT_VALUE_ERROR
MQRC_INITIALIZATION_FAILED
MQRC_INT_ATTRIB_COUNT_ERROR
MQRC_INT_ATTRIB_COUNT_TOO_SMALL
MQRC_INT_ATTRS_ARRAY_ERROR
MQRC_MAX_CONNNS_LIMIT_REACHED
MQRC_MD_ERROR
MQRC_MISSING_REPLY_TO_Q
MQRC_MSG_ID_ERROR
MQRC_MSG_TOO_BIG_FOR_CHANNEL
MQRC_MSG_TOO_BIG_FOR_Q
MQRC_MSG_TYPE_ERROR
MQRC_NAME_IN_USE
MQRC_NAME_NOT_VALID_FOR_TYPE
MQRC_NO_MSG_AVAILABLE
MQRC_NO_MSG_LOCKED
MQRC_NO_MSG_UNDER_CURSOR
MQRC_NOT_AUTHORIZED
MQRC_NOT_CONVERTED
MQRC_NOT_OPEN_FOR_BROWSE
MQRC_NOT_OPEN_FOR_INPUT
MQRC_NOT_OPEN_FOR_INQUIRE
MQRC_NOT_OPEN_FOR_PASS_ALL
MQRC_NOT_OPEN_FOR_PASS_IDENT
MQRC_OBJECT_ALREADY_EXISTS
MQRC_OBJECT_CHANGED
MQRC_OBJECT_DAMAGED
MQRC_OBJECT_IN_USE
MQRC_OBJECT_TYPE_ERROR
MQRC_OD_ERROR
MQRC_OPTION_NOT_VALID_FOR_TYPE
MQRC_OPTIONS_ERROR
MQRC_PAGSET_ERROR
MQRC_PAGSET_FULL
MQRC_PERSISTENCE_ERROR
MQRC_PERSISTENT_NOT_ALLOWED
MQRC_PMO_ERROR
MQRC_PRIORITY_ERROR
MQRC_PRIORITY_EXCEEDS_MAXIMUM
MQRC_PUT_INHIBITED
MQRC_Q_ALREADY_EXISTS
MQRC_Q_DELETED
MQRC_Q_DEPTH_HIGH
MQRC_Q_DEPTH_LOW
MQRC_Q_FULL
MQRC_Q_MGR_ACTIVE
MQRC_Q_MGR_NAME_ERROR
MQRC_Q_MGR_NOT_ACTIVE
MQRC_Q_MGR_NOT_AVAILABLE
MQRC_Q_MGR_QUIESCING
MQRC_Q_MGR_STOPPING
MQRC_Q_NOT_EMPTY
MQRC_Q_SERVICE_INTERVAL_HIGH
MQRC_Q_SERVICE_INTERVAL_OK
MQRC_Q_SPACE_NOT_AVAILABLE
MQRC_Q_TYPE_ERROR
MQRC_REMOTE_Q_NAME_ERROR
MQRC_REPORT_OPTIONS_ERROR
MQRCRESOURCE_PROBLEM
MQRCSECOND_MARK_NOT_ALLOWED
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQRC_SECURITY_ERROR</td>
<td>MQRC_SECURITY_ERROR</td>
</tr>
<tr>
<td>MQRC_SELECTOR_COUNT_ERROR</td>
<td>MQRC_SELECTOR_COUNT_ERROR</td>
</tr>
<tr>
<td>MQRC_SELECTOR_ERROR</td>
<td>MQRC_SELECTOR_ERROR</td>
</tr>
<tr>
<td>MQRC_SELECTOR_LIMIT_EXCEEDED</td>
<td>MQRC_SELECTOR_LIMIT_EXCEEDED</td>
</tr>
<tr>
<td>MQRC_SELECTOR_NOT_FOR_TYPE</td>
<td>MQRC_SELECTOR_NOT_FOR_TYPE</td>
</tr>
<tr>
<td>MQRC_SERVICE_ERROR</td>
<td>MQRC_SERVICE_ERROR</td>
</tr>
<tr>
<td>MQRC_SERVICE_NOT_AVAILABLE</td>
<td>MQRC_SERVICE_NOT_AVAILABLE</td>
</tr>
<tr>
<td>MQRC_SIGNAL_OUTSTANDING</td>
<td>MQRC_SIGNAL_OUTSTANDING</td>
</tr>
<tr>
<td>MQRC_SIGNAL_REQUEST_ACCEPTED</td>
<td>MQRC_SIGNAL_REQUEST_ACCEPTED</td>
</tr>
<tr>
<td>MQRC_SIGNAL1_ERROR</td>
<td>MQRC_SIGNAL1_ERROR</td>
</tr>
<tr>
<td>MQRC_SOURCE_BUFFER_ERROR</td>
<td>MQRC_SOURCE_BUFFER_ERROR</td>
</tr>
<tr>
<td>MQRC_SOURCE_CCSID_ERROR</td>
<td>MQRC_SOURCE_CCSID_ERROR</td>
</tr>
<tr>
<td>MQRC_SOURCE_DECIMAL_ENC_ERROR</td>
<td>MQRC_SOURCE_DECIMAL_ENC_ERROR</td>
</tr>
<tr>
<td>MQRC_SOURCE_FLOAT_ENC_ERROR</td>
<td>MQRC_SOURCE_FLOAT_ENC_ERROR</td>
</tr>
<tr>
<td>MQRC_SOURCE_INTEGER_Enc_ERROR</td>
<td>MQRC_SOURCE_INTEGER_Enc_ERROR</td>
</tr>
<tr>
<td>MQRC_SOURCE_LENGTH_ERROR</td>
<td>MQRC_SOURCE_LENGTH_ERROR</td>
</tr>
<tr>
<td>MQRC_STORAGE_CLASS_ERROR</td>
<td>MQRC_STORAGE_CLASS_ERROR</td>
</tr>
<tr>
<td>MQRC_STORAGE_NOT_AVAILABLE</td>
<td>MQRC_STORAGE_NOT_AVAILABLE</td>
</tr>
<tr>
<td>MQRC_SUPPRESSED_BY_EXIT</td>
<td>MQRC_SUPPRESSED_BY_EXIT</td>
</tr>
<tr>
<td>MQRC_SYNCPOINT_LIMIT_REACHED</td>
<td>MQRC_SYNCPOINT_LIMIT_REACHED</td>
</tr>
<tr>
<td>MQRC_SYNCPOINT_NOT_AVAILABLE</td>
<td>MQRC_SYNCPOINT_NOT_AVAILABLE</td>
</tr>
<tr>
<td>MQRC_TARGET_BUFFER_ERROR</td>
<td>MQRC_TARGET_BUFFER_ERROR</td>
</tr>
<tr>
<td>MQRC_TARGET_CCSID_ERROR</td>
<td>MQRC_TARGET_CCSID_ERROR</td>
</tr>
<tr>
<td>MQRC_TARGET_DOMAIN_ERROR</td>
<td>MQRC_TARGET_DOMAIN_ERROR</td>
</tr>
<tr>
<td>MQRC_TARGET_FLOAT_ENC_ERROR</td>
<td>MQRC_TARGET_FLOAT_ENC_ERROR</td>
</tr>
<tr>
<td>MQRC_TARGET_INACTIVE_ERROR</td>
<td>MQRC_TARGET_INACTIVE_ERROR</td>
</tr>
<tr>
<td>MQRC_TARGET_LENGTH_ERROR</td>
<td>MQRC_TARGET_LENGTH_ERROR</td>
</tr>
<tr>
<td>MQRC_TERMINATION_FAILED</td>
<td>MQRC_TERMINATION_FAILED</td>
</tr>
<tr>
<td>MQRC_TRIGGER_CONTROL_ERROR</td>
<td>MQRC_TRIGGER_CONTROL_ERROR</td>
</tr>
<tr>
<td>MQRC_TRIGGER_DEPTH_ERROR</td>
<td>MQRC_TRIGGER_DEPTH_ERROR</td>
</tr>
<tr>
<td>MQRC_TRIGGER_MSG_PRIORITY_ERR</td>
<td>MQRC_TRIGGER_MSG_PRIORITY_ERR</td>
</tr>
<tr>
<td>MQRC_TRIGGER_TYPE_ERROR</td>
<td>MQRC_TRIGGER_TYPE_ERROR</td>
</tr>
<tr>
<td>MQRC_TRUNCATED</td>
<td>MQRC_TRUNCATED</td>
</tr>
<tr>
<td>MQRC_TRUNCATED_MSG_ACCEPTED</td>
<td>MQRC_TRUNCATED_MSG_ACCEPTED</td>
</tr>
<tr>
<td>MQRC_TRUNCATED_MSG_FAILED</td>
<td>MQRC_TRUNCATED_MSG_FAILED</td>
</tr>
<tr>
<td>MQRC_UNEXPECTED_ERROR</td>
<td>MQRC_UNEXPECTED_ERROR</td>
</tr>
<tr>
<td>MQRC_UNKNOWN_ALIAS_BASE_Q</td>
<td>MQRC_UNKNOWN_ALIAS_BASE_Q</td>
</tr>
<tr>
<td>MQRC_UNKNOWN_AUTH_ENTITY</td>
<td>MQRC_UNKNOWN_AUTH_ENTITY</td>
</tr>
<tr>
<td>MQRC_UNKNOWN_DEF_XMIT_Q</td>
<td>MQRC_UNKNOWN_DEF_XMIT_Q</td>
</tr>
<tr>
<td>MQRC_UNKNOWN_ENTITY</td>
<td>MQRC_UNKNOWN_ENTITY</td>
</tr>
<tr>
<td>MQRC_UNKNOWN_OBJECT_NAME</td>
<td>MQRC_UNKNOWN_OBJECT_NAME</td>
</tr>
<tr>
<td>MQRC_UNKNOWN_OBJECT_Q_MGR</td>
<td>MQRC_UNKNOWN_OBJECT_Q_MGR</td>
</tr>
<tr>
<td>MQRC_UNKNOWN_Q_NAME</td>
<td>MQRC_UNKNOWN_Q_NAME</td>
</tr>
<tr>
<td>MQRC_UNKNOWN_REF_OBJECT</td>
<td>MQRC_UNKNOWN_REF_OBJECT</td>
</tr>
<tr>
<td>MQRC_UNKNOWN_REMOTE_Q_MGR</td>
<td>MQRC_UNKNOWN_REMOTE_Q_MGR</td>
</tr>
<tr>
<td>MQRC_UNKNOWN_REPORT_OPTION</td>
<td>MQRC_UNKNOWN_REPORT_OPTION</td>
</tr>
<tr>
<td>MQRC_UNKNOWN_XMIT_Q</td>
<td>MQRC_UNKNOWN_XMIT_Q</td>
</tr>
<tr>
<td>MQRC_WAIT_INTERVAL_ERROR</td>
<td>MQRC_WAIT_INTERVAL_ERROR</td>
</tr>
<tr>
<td>MQRC_XMIT_Q_TYPE_ERROR</td>
<td>MQRC_XMIT_Q_TYPE_ERROR</td>
</tr>
<tr>
<td>MQRC_XMIT_Q_USAGE_ERROR</td>
<td>MQRC_XMIT_Q_USAGE_ERROR</td>
</tr>
<tr>
<td>MQRC_XWAIT_CANCELED</td>
<td>MQRC_XWAIT_CANCELED</td>
</tr>
<tr>
<td>MQRC_XWAIT_ERROR</td>
<td>MQRC_XWAIT_ERROR</td>
</tr>
<tr>
<td>MQRC_ALLOCATE_FAILED</td>
<td>MQRC_ALLOCATE_FAILED</td>
</tr>
<tr>
<td>MQRC_CFATTR_VALUE_ERROR</td>
<td>MQRC_CFATTR_VALUE_ERROR</td>
</tr>
<tr>
<td>MQRC_BATCH_SIZE_ERROR</td>
<td>MQRC_BATCH_SIZE_ERROR</td>
</tr>
<tr>
<td>MQRC_BIND_FAILED</td>
<td>MQRC_BIND_FAILED</td>
</tr>
<tr>
<td>MQRC_CCSID_ERROR</td>
<td>MQRC_CCSID_ERROR</td>
</tr>
<tr>
<td>MQRC_CELL_DIR_NOT_AVAILABLE</td>
<td>MQRC_CELL_DIR_NOT_AVAILABLE</td>
</tr>
<tr>
<td>MQRC_CFH_COMMAND_ERROR</td>
<td>MQRC_CFH_COMMAND_ERROR</td>
</tr>
<tr>
<td>MQRC_CFH_CONTROL_ERROR</td>
<td>MQRC_CFH_CONTROL_ERROR</td>
</tr>
<tr>
<td>MQRC_CFH_LENGTH_ERROR</td>
<td>MQRC_CFH_LENGTH_ERROR</td>
</tr>
<tr>
<td>MQRC_CF_HMSG_SEQ_NUMBER_Err</td>
<td>MQRC_CF_HMSG_SEQ_NUMBER_Err</td>
</tr>
<tr>
<td>MQRC_CF_PARM_COUNT_ERROR</td>
<td>MQRC_CF_PARM_COUNT_ERROR</td>
</tr>
<tr>
<td>MQRC_CF_TYPE_ERROR</td>
<td>MQRC_CF_TYPE_ERROR</td>
</tr>
<tr>
<td>MQRC_CF_VERSION_ERROR</td>
<td>MQRC_CF_VERSION_ERROR</td>
</tr>
<tr>
<td>MQRC_CF_ILC_F_COUNT_ERROR</td>
<td>MQRC_CF_ILC_F_COUNT_ERROR</td>
</tr>
<tr>
<td>MQRC_CF_ILC_F_DUPLICATE_VALUE</td>
<td>MQRC_CF_ILC_F_DUPLICATE_VALUE</td>
</tr>
</tbody>
</table>

MQRC: MQSeries Return Codes

MQCCF: MQSeries C/C++ Application Programming Interface (API) Return Codes
(end of document)
Sending your comments to IBM

MA18: A Rexx Interface to IBM MQSeries for MVS/ESA
Version 2.0

MA18 SCRIPT

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book. Please limit your comments to the information in this book and the way in which the information is presented.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By mail, use the Readers’ Comment Form.
- By fax:
  - From outside the U.K., after your international access code use 44 1962 841409
  - From within the U.K., use 01962 841409
- Electronically, use the appropriate network ID:
  - IBMLink: WINVMD(TSCC)
  - Internet: tscc@hursley.ibm.com

Whichever you use, ensure that you include:

- The publication number and title
- The page number or topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.
Readers’ Comments

MA18: A Rexx Interface to IBM MQSeries for MVS/ESA
Version 2.0

MA18 SCRIPT

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Be sure to print your name and address below if you would like a reply.

__________________________  _________________________
Name  Address

__________________________
Company or Organization

__________________________
Telephone  Email
You can send your comments POST FREE on this form from any one of these countries:

Australia  Finland  Iceland  Netherlands  Singapore  United States
Belgium    France    Israel     New Zealand     Spain      of America
Bermuda    Germany   Italy      Norway        Sweden       
Cyprus      Greece    Luxembourg Portugal        Switzerland
Denmark     Hong Kong Monaco    Republic of Ireland United Arab Emirates

If your country is not listed here, your local IBM representative will be pleased to forward your comments to us. Or you can pay the postage and send the form direct to IBM (this includes mailing in the U.K.).

IBM

REponce PayEE
GRANDE-BRETAGNE

IBM United Kingdom Laboratories Limited
Information Development Department (MP 095)
Hursley Park
WINCHESTER, Hants
SO21 2ZZ
United Kingdom

From:
Name
Company or Organization
Address
EMAIL
Telephone