MA1D: A Netview Rexx interface for MQSeries for MVS/ESA Version 2.0

Document: MA1D SCRIPT
Issued: 19th June 1997
Revision Date: 19th June, 1997
Previous Revision Date: None
Next Review: As required

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

MQSeries
Commercial Messaging

Take Note!

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

First Edition, June 1997

This edition applies to Version 2.0 of MA1D: A Netview Rexx interface for 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.

Contents

Figures

Tables

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

Date	Changes
2.0 - 19th June 1997	Initial Version (aligned with MA18). My thanks to Phil Vining for testing this interface.

Preface

This SupportPac provides a Netview Rexx interface for IBM MQSeries Version 1.1.4 on MVS/ESA. It permits the usage of MQ function within a Netview Rexx exec.

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 Netview Rexx execs
- 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 Netview 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 from within Netview for MVS to a **local** Queue Manager
- A Rexx Exec which demonstrates usage of the interface
- This paper which documents the interface (in various formats).

Other SupportPacs

MA1F A SupportPac containing a Netview Rexx interface to MQSeries for MVS/ESA to issue MQ Commands (RXMQWC).

MA18 A SupportPac containing a Rexx interface to MQSeries for MVS/ESA

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 Netview for MVS Rexx environment, for Message Queueing access.

A single module **RXMQW** 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 native MVS/ESA environment. You need SupportPac **MA18** instead.

Chapter 2. Installing the SupportPac

Take the following actions to install the SupportPac from the MA1D.ZIP file. You will need a TSO session to install the SupportPac into the relevant Netview.

1. Use **INFOZIP's** UNZIP32 to unpack the **MA1D.ZIP** file.

   This produces

   - MA1DSEQX (the RXMQW load module)
   - MA1DTEST.JCL (sample exec)

2. **MA1DSEQX** 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 **MA1DSEQ**

     send ma1dseqx A:ma1dseq

     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 ma1dseqx ma1dseq

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

PC File	**ma1dseqx**
Host File	**ma1dseq**
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 set the **LRECL** to **80**

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

   receive indsname(MA1DSEQ)

   When prompted for a filename, reply

   dsn(MA1DLOAD)

   This creates a PDS called **mvsuserid.MA1DLOAD** with the single member **RXMQW**

4. Use ISPF 3.2 to delete the MA1DSEQ file

5. Use ISPF 3.3 to copy member **RXMQW** from file **MA1DLOAD** into your load library (which must have DCB=(DSORG=PO,RECFM=U,LRECL=32760,BLKSIZE=32760) and be authorised)

6. Use ISPF 3.2 to delete the MA1DLOAD file

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

8. To permit tracing to appear in a file (as opposed to the current Netview Operators console), you may need to configure a Netview Sequential Logging task, as described in “Netview Trace Configuration” on page 15.

9. **MA1DTEST.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 ma1dtest.jcl A:ma1djcl ascii 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 ma1dtest.jcl ma1djcl

- With Personal Communications, use the "Send Files to Host" option under the Transfer menu item to transmit to TSO

PC File	**ma1dtest.jcl**
Host File	**ma1djcl**
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. Copy this file into a PDS already used by the Netview System to hold user execs (and test 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 Netview Rexx Exec, nothing special has to be done to make it known to Netview's Rexx (there is no equivalent in Netview/Rexx to the OS/2 Rexx RxFuncAdd call).

See “Netview Trace Configuration” on page 15 for information on Netview configuration to support tracing.

Please note these operational characteristics:

- There is a reserved variable name **RXMQWG** which is used by the interface to save information across calls. Therefore,
  - when using internal Rexx Procedures, you **must** expose RXMQWG in the procedure statement
  - Rexx External procedures are not supported by the interface (but you could arrange to provide the callers RXMQWG as part of the parameters and set RXMQWG in this external procedure).
- Enabling the trace will send non-printable characters to the current users console unless you direct it to a file (see “Netview Trace Destination” on page 14).
- Use of **RXMQWC** (from SupportPac MA1F) conflicts with use of RXMQW; so RXMQWC **cannot** be used whilst RXMQW is connected to a Queue Manager.

Chapter 3. Netview and MQSeries implications

Netview and MVS

It is important to understand that MQSeries regards Netview as just another MVS Batch job that is issuing MQ calls. Consequently, the use of this MQ/Rexx/Netview interface makes Netview appear to QM as a **single** batch job; there is no concept of a Netview operator or command having a distinct relationship with the Queue Manager. However, the Netview/Rexx environment is such that each Netview Rexx task runs under its own MVS TCB so providing the MQ isolation.

Triggering

MQSeries triggering cannot start a Netview task when a message arrives in an Initiation Queue. If you want this to happen, then you must implement, as a Netview Automated Task (Autotask), your own Trigger Monitor. This will wait upon the Initiation Queue until a message arrives, and then use the contents of this message to start up whatever Netview processing is required upon the 'real' Queue.

Rexx environment

The normal Netview Rexx rules still apply when this MQ/Rexx/Netview interface is used. In particular, Rexx Variables representing the MQ Constants are not available outside of the Exec in which they were created.

Please take care to ensure that the RXMQWG Global Variable is not mistreated in any way. See “Internal procedures” on page 9 for full details on this variable.

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 Netview 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/Netview interface are accessed via the RXMQW call, with the first parameter indicating the function to be run.

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

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

Return Codes

The RXMQW 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 RXMQW 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 2 MQ Reason Code (or 0 if MQ not done)

Word 4 RXMQW 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:

RXMQW.LASTRC current operation Return Code

RXMQW.LASTCC current operation MQ Completion Code

RXMQW.LASTAC current operation MQ Reason Code

RXMQW.LASTOP current operation RXMQW function

RXMQW.LASTMSG current operation Return String

Return Code naming

A set of variables called RXMQW.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:

```
rcc = '2048 2 2048 RXMQWPUT ERROR'
interpret 'fcs = RXMQW.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 **RXMQWG** contains information that is saved across execution of RXMQW. Consequently, it must be available throughout the Exec which uses the interface (*do not alter it!*). So, when using internal procedures, you should EXPOSE RXMQWG as part of the procedure definition. Thus:

```
internal_proc (a, b ,c)


internal_proc: procedure expose RXMQWG

     rcc = RXMQW('CONS')
```

When in an internal procedure, all the RXMQW 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 RXMQWG cannot be directly exposed into the latters Rexx Workspace). However, **at your own risk**, you could manually provide RXMQWG as part of the parameters and set up RXMQWG 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:

```
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:

```
drop iod. ; drop ood.
iod.on = 'N1'
iod.ot = MQOT_Q

rcc = RXMQW('OPEN', 'iod.', mqoo_inquire, 'h1', 'ood.')
say 'RC=' rcc  'H=' h1
do j=1 to words(ood.zlist)
   k = word(ood.zlist,j)
   say k '/'ood.k'/'
   end
```

ZLIST can be used for Event processing:

```
drop bm. ; drop ed.
rcc = RXMQW('BROWSE', he, 'bm.')
say 'Browse RC=' rcc

rcc = RXMQW('EVENT', 'bm.', 'ed.')
say 'Event RC =' rcc
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
   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 OQM 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
     end

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

Figure 1. 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 = RXMQW('...', ...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 RXMQWTRACE Rexx variable. The trace is sent to the current Netview Operator's Command Facility screen as **U** type messages. Some of the settings can produce a lot of output. See “Netview Trace Destination” on page 14 for information about redirecting the trace. The settings are:

CONN	mqconn
DISC	mqdisc
OPEN	mqopen
CLOSE	mqopen
GET	mqget
PUT	mqput
INQ	mqinq
SET	mqset
CMIT	mqcmit
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 RXMQW processing
GG	Globals processing (RXMQWG)
INIT	Initialisation processing
CONS	Literal processing
TERM	Deregistration processing
*****	Trace everything!!!

So, to trace Gets and Puts, one would code

```
RXMQWTRACE = 'PUT GET'
```

Netview Trace Destination

The destination of the trace can be altered from the current Operators screen (U-type messages) into a file, **provided** that the relevant Netview configuration has been performed.

This routing is controlled by the **RXMQWTRACETO** variable. If it is set to *****, then the trace is sent to the User's Console. Alternatively, RXMQWTRACETO can be set to the name of a Netview Sequential Logging task. If this task is active, then the results of the trace are sent to this task for disposal (into a file).

Don't forget that you should set RXMQWTRACETO before RXMQWTRACE (or else you will not get what you expect).

If the named Netview Sequential Logging task does not exist, or is inactive, then the trace is merely lost without objection.

If you need to contact the author with a problem, the trace will be required. Therefore, I recommend that you configure Netview to support tracing to a file.

Netview Trace Configuration

You need to define (or use an existing) Netview Sequential Logging task. For full details of how to setup Netview for this, see the Netview Customisation book (this works via the DSIWLS SAMTASK facility).

The following steps show how one might configure Netview with a Sequential Logging Task called **RXMQWT** (so one would code RXMQWTRACETO='RXMQWT' within the exec to route the trace to a file) :

1. In Netview, define a Sequential Log to handle writing the the RXMQW trace records to DASD

   In a suitable Netview DISPARM dataset create a new member, with the name RXMQWTD, that contains the following:

```
*************************************************************
*                                                           *
*  Netview Sequential Log Task RXMQWT                       *
*                                                           *
*  This Task handles writing RXMQW trace records to DASD.   *
*                                                           *
*  RXMQW tracing is controlled by the RXMQW variable        *
*  RXMQWTRACETO.                                            *
*                                                           *
*  RXMQW is a Netview REXX function that interfaces to an   *
*  MQSeries Queue Manager.                                  *
*                                                           *
*************************************************************

         DSTINIT FUNCT=OTHER
         DSTINIT DSRBO=1
         DSTINIT PBSDN=RXMQWTP
         DSTINIT SBSDN=RXMQWTS
         LOGINIT AUTOFLIP=YES
         LOGINIT RESUME=NO
```

2. In Netview, define a task to handle writing the the RXMQW trace records to DASD

   Add the following task definition to the DSIPARM member DSIDMN, or to the appropriate DSIDMN included member, such as DSIDMNB:

```
*************************************************************
*                                                           *
*  Netview Sequential Log Task RXMQWT                       *
*                                                           *
*  This task handles writing RXMQW trace records to DASD.   *
*                                                           *
*  RXMQW tracing is controlled by the RXMQW variable        *
*  RXMQWTRACETO.                                            *
*                                                           *
*  RXMQW is a Netview REXX function that interfaces to an   *
*  MQSeries Queue Manager.                                  *
*                                                           *
*  Note:  If INIT=N is used, the task is not automatically  *
*         started by Netview during Netview initialisation. *
*                                                           *
*************************************************************

   TASK     MOD=DSIZDST,TSKID=RXMQWT,MEM=RXMQWTD,PRI=2,INIT=N
```

3. Ensure that the Netview BSAM Sequential Logging facilities are available.

   The following Command Model statements are required in the DSIPARM member DSICMD, or in the appropriate DSICMD included member, such as DSICMDB:

```
*-------------------------------------------------------------*
* NOTE - THE FOLLOWING 2 CMDMDL STATEMENTS ARE NECESSARY FOR  *
* WRITING DATA TO A SEQUENTIAL LOG USING BSAM.                *
* IF SEQUENTIAL LOGGING IS BEING USED, UNCOMMENT THE FOLLOWING 2 *
* CMDMDL STATEMENTS.                                         *
*-------------------------------------------------------------*

DSIBSWCP CMDMDL   MOD=DSIBSWCP,TYPE=D
DSIZBSQW CMDMDL   MOD=DSIZBSQW,TYPE=RD,PARSE=N,RES=Y
```

4. Create RXMQW primary and secondary log datasets.

   The necessary dataset allocation JCL DD statements are as follows:

```
//*
//RXMQWTP  DD  DSNAME=<hlq>.RXMQWTP,
//             UNIT=<unitype>,
//             VOLUME=SER=<volser>,
//             DCB=(DSORG=PS,RECFM=VB,LRECL=4092,BLKSIZE=4096),
//             SPACE=(TRK,(5,5))
//             DISP=(NEW,CATLG,CATLG)
//*
//RXMQWTS  DD  DSNAME=<hlq>.RXMQWTS,
//             UNIT=<unitype>,
//             VOLUME=SER=<volser>,
//             DCB=(DSORG=PS,RECFM=VB,LRECL=4092,BLKSIZE=4096),
//             SPACE=(TRK,(5,5)),
//             DISP=(NEW,CATLG,CATLG)
//*
```

   You will need to supply appropriate DSNAME, UNIT and VOLUME information. You may need to increase the SPACE allocation.

5. Define the RXMQW primary and secondary log datasets to the NetView system.

   In the Netview startup procedure, such as CNMPROC, add the following MVS JCL DD statements:

```
//*
//*            *** RXMQW TRACE DATASETS ***
//*
//RXMQWTP  DD  DSNAME=<hlq>.RXMQWTP,DISP=SHR
//*
//RXMQWTS  DD  DSNAME=<hlq>.RXMQWTS,DISP=SHR
//*
```

   You will need to supply appropriate DSNAME information.

6. If the Netview RXMQWT log task is not automatically started by NetView during Netview initialisation (INIT=N used), then from a Netview NCCF user session start the RXMQWT log task with the following NCCF command:

   ```
   START TASK=RXMQWT
   ```

   To determine the current state of the RXMQWT log task use the NCCF command:

   ```
   LIST RXMQWT
   ```

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** 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 12).

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 (`RXMQW( ..., 'AGMO.', ... )`).

However, you are at liberty to use the normal Rexx substitutions on an interface call (so `Z = 'AGMO.' ; RXMQW...(..., 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 RXMQW('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 8 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 10) 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 RXMQW('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 1. Object Descriptor Mappings

Stem. Component	**MQOD Structure name**	**Input, Output or Both**	**Number or Text**
.OT	ObjectType	I	N
.ON	ObjectName	B	T
.OQM	ObjectQMgrName	B	T
.DQN	DynamicQueue	I	T
.AUID	AlternateUserid	I	T

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

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

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, RXMQW('GET') uses an Input and Output Stem variable to hold this information.

Table 3. Get Message Options Mappings

Stem. Component	MQGMO Structure name	Input, Output or Both	Number or Text
.OPT	Options	I	N
.WAIT	WaitInterval	I	N
.RQN	ResolvedQueueName	O	T

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, RXMQW('PUT') uses an Input and Output Stem variable to hold this information.

Table 4. Put Message Options Mappings

Stem. Component	MQPMO Structure name	Input, Output or Both	Number or Text
.OPT	Options	I	N
.TIME	Timeout	I	N
.CON	Context	I	T
.KDC	KnownDestCount	-	N
.UDC	UnKnownDestCount	-	N
.IDC	InvalidDestCount	-	N
.RQN	ResolvedQueueName	O	T
.RQMN	ResolvedQueueMgrName	O	T

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 RXMQW (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 RXMQW 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:

The Standard MQ functions:

The Extension functions:

Common Return Codes

These Return Codes can be commonly returned by RXMQW:

0 0 0 RXMQW Nothing happened

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

-99 0 0 RXMQW Incorrect number of Parms supplied

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

-98 0 0 RXMQW Globals not found (RXMQWG exposed?)

Explanation: The reserved Rexx Variable **RXMQWG** which contains information that lasts across the individual RXMQW call was not located. In an internal procedure, this probably means that you have not exposed RXMQWG on the procedure statement (see “Internal procedures” on page 9).

-97 0 0 RXMQW Globals look terrible (RXMQWG exposed?)!!

Explanation: The reserved Rexx Variable **RXMQWG** which contains information that lasts across the individual RXMQW 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 RXMQWG correctly (see “Internal procedures” on page 9).

-96 0 0 RXMQW Unknown request

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

-99 0 0 RXMQWxxxx UNKNOWN FAILURE

Explanation: Some unknown error has occurred in function RXMQW('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 RXMQW('INIT') call needs to be done **once** with the Exec.

Parameters

None

Call

```
rcc = RXMQW('INIT')
```

Additional Interface Return Codes and Messages

None

Example

```
rcc = RXMQW('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

```
rcc = RXMQW('CONS')
```

Additional Interface Return Codes and Messages

None

Example

```
rcc = RXMQW('CONS')
```

Termination

Description

This function simply removes the information which is saved across RXMQW executions (the things in the reserved Rexx Variable RXMQWG). It **does not** initiate MQ Termination processing. If a prior RXMQW('CLOSE') or a RXMQW('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 = RXMQW('TERM')
```

Additional Interface Return Codes and Messages

None

Example

```
rcc = RXMQW('TERM')
```

RXMQWCONN

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 Netview Rexx processor, in this case).

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

Owing to the above restriction, the Queue Manager Handle returned by the use of *MQCONN* within RXMQW 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 = RXMQW('CONN', QM )
```

Additional Interface Return Codes and Messages

-1 0 0 RXMQWCONN Bad number of parms

Explanation: You must specify only **one** parameter to RXMQW('CONN'); this parameter being the name of the Queue Manager to contact.

-2 0 0 RXMQWCONN Supplied QM name is too short

Explanation: The Queue Manager Name supplied was of Zero Length (ie: '').

-3 0 0 RXMQWCONN Supplied QM name too long

Explanation: The maximum length of a Valid Queue Manager Name is 48 bytes.

-4 0 0 RXMQWCONN QM already supplied

Explanation: The QM name has already been supplied to RXMQWG (an attempt to contact more than one QM is invalid).

-5 0 0 RXMQWCONN QM already connected

Explanation: A QM is already connected to RXMQW.

-6 0 0 RXMQWCONN Thread already connected

Explanation: The current TCB is already connected to a QM.

-7 0 0 RXMQWCONN Thread already connected to QM

Explanation: The current TCB is already connected to a QM.

Example

```
rcc = RXMQW('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.

RXMQWDISC

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

```
rcc = RXMQW('DISC')
```

Additional Interface Return Codes and Messages

None

Example

```
rcc = RXMQW('DISC')
```

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

RXMQWOPEN

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

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

                    or

rcc = RXMQW('OPEN', QueueName       , OpenOptions, 'VarHandle', 'Stem.Output.OD.' )
```

Additional Interface Return Codes and Messages

-1 0 0 RXMQWOPEN Bad number of Parms

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

-2 0 0 RXMQWOPEN 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 RXMQWOPEN 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 RXMQWOPEN 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 RXMQWOPEN 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 RXMQWOPEN 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 RXMQWOPEN QM not connected

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

-8 0 0 RXMQWOPEN Thread not connected

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

-9 0 0 RXMQWOPEN Too many opened Objects

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

Example

```
opts =   MQOO_INQUIRE           + MQOO_INPUT_SHARED       ,
       + MQOO_BROWSE            + MQOO_SAVE_ALL_CONTEXT   ,
       + MQOO_FAIL_IF_QUIESCING

rcc  = RXMQW('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').

```
iod.OT = MQOT_Q
iod.ON = 'N1'

rcc = RXMQW('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.

RXMQWCLOSE

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 RXMQW('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

```
rcc = RXMQW('CLOSE', handle, CloseOptions )
```

Additional Interface Return Codes and Messages

-1 0 0 RXMQWCLOSE Bad number of Parms

Explanation: You **must** specify two parameters to the RXMQW('CLOSE') call.

-2 0 0 RXMQWCLOSE Handle not supplied

Explanation: No value has been keyed for the first parameter, the handle representing the MQ object.

-3 0 0 RXMQWCLOSE 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 RXMQWCLOSE 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 RXMQWCLOSE QM not connected

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

-6 0 0 RXMQWCLOSE Thread not connected

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

-7 0 0 RXMQWCLOSE 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 RXMQWCLOSE Handle invalid

Explanation: The handle specified does not relate to an accessed MQ Object.

Example

```
rcc = RXMQW('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.

RXMQWINQ

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 RXMQW('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 = RXMQW('INQ', handle, Attribute, 'VarAttributeValue' )
```

Additional Interface Return Codes and Messages

-1 0 0 RXMQWINQ Bad number of Parms

Explanation: You **must** specify three parameters to the RXMQW('INQ') call.

-2 0 0 RXMQWINQ Handle not supplied

Explanation: No value has been keyed for the first parameter, the handle representing the MQ object.

-3 0 0 RXMQWINQ 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 RXMQWINQ 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 RXMQWINQ 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 RXMQWINQ QM not connected

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

-7 0 0 RXMQWINQ Thread not connected

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

-8 0 0 RXMQWINQ Handle out of range

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

-9 0 0 RXMQWINQ Handle invalid

Explanation: The value of the handle supplied is not in the known range for a handle within the interface.

-10 0 0 RXMQWINQ 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 RXMQWINQ 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 = RXMQW('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.

RXMQWSET

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 RXMQW('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 = RXMQW('SET', handle, Attribute, AttributeSetting )
```

Additional Interface Return Codes and Messages

-1 0 0 RXMQWSET Bad number of Parms

Explanation: You **must** specify three parameters to the RXMQWSET call.

-2 0 0 RXMQWSET Handle not supplied

Explanation: No value has been keyed for the first parameter, the handle representing the MQ object.

-3 0 0 RXMQWSET 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 RXMQWSET 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 RXMQWSET Attribute Setting not supplied

Explanation: No value was supplied for the attribute under consideration.

-6 0 0 RXMQWSET QM not connected

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

-7 0 0 RXMQWSET Thread not connected

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

-8 0 0 RXMQWSET Handle out of range

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

-9 0 0 RXMQWSET Handle invalid

Explanation: The value of the handle supplied is not in the known range for a handle within the interface.

-10 0 0 RXMQWSET 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 RXMQWSET 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 RXMQWSET 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 = RXMQW('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.

RXMQWCMIT

Description

This verb will issue a *MQCMIT* verb. It syncpoints the current Queue Manager accesses. Note that this operation effects **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 = RXMQW('CMIT')
```

Additional Interface Return Codes and Messages

-1 0 0 RXMQWCMIT Bad number of Parms

Explanation: You cannot specify any parameters to this call.

-2 0 0 RXMQWCMIT Thread not connected to QM

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

Example

```
rcc = RXMQW('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.

RXMQWBACK

Description

This verb will issue a *MQBACK* verb. It rollsback the current Queue Manager accesses. Note that this operation effects **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 = RXMQW('BACK')
```

Additional Interface Return Codes and Messages

-1 0 0 RXMQWBACK Bad number of Parms

Explanation: You cannot specify any parameters to this call.

-2 0 0 RXMQWBACK Thread not connected to QM

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

Example

```
rcc = RXMQW('BACK')
```

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

RXMQWGET

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 “RXMQWBROWSE” on page 54. When using a Get.Signal operation, the ECB can only be tested by using the **RXMQW('QSIGNAL')** operation as described in “RXMQWQSIGNAL” on page 51.

Parameters

1. The Handle for the Queue obtained from a prior RXMQW('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 8 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

```
rcc = RXMQW('GET', handle, 'Stem.Message.' , 'Stem.Input.MD.' , 'Stem.Output.MD. ' ,
                                             'Stem.Input.GMO.', 'Stem.Output.GMO.' )
```

Additional Interface Return Codes and Messages

-1 0 0 RXMQWGET Bad number of Parms

Explanation: You **must** specify six parameters to the RXMQWGET call.

-2 0 0 RXMQWGET Handle not supplied

Explanation: No value has been keyed for the first parameter, the handle representing the MQ object.

-3 0 0 RXMQWGET 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 RXMQWGET 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 RXMQWGET 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 RXMQWGET 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 RXMQWGET 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 RXMQWGET QM not connected

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

-9 0 0 RXMQWGET Thread not connected

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

-10 0 0 RXMQWGET Handle out of range

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

-11 0 0 RXMQWGET Handle invalid

Explanation: The value of the handle supplied is not in the known range for a handle within the interface.

Example

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

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

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

rcc = RXMQW('GET', hn1, '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 upto 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.

RXMQWPUT

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 RXMQW('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

```
rcc = RXMQW('PUT', handle, 'Stem.Message.' , 'Stem.Input.MD.' , 'Stem.Output.MD. ' ,
                                             'Stem.Input.PMO.', 'Stem.Output.PMO.' )
```

Additional Interface Return Codes and Messages

-1 0 0 RXMQWPUT Bad number of Parms

Explanation: You **must** specify six parameters to the RXMQWPUT call.

-2 0 0 RXMQWPUT Handle not supplied

Explanation: No value has been keyed for the first parameter, the handle representing the MQ object.

-3 0 0 RXMQWPUT 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 RXMQWPUT 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 RXMQWPUT 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 RXMQWPUT 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 RXMQWPUT 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 RXMQWPUT QM not connected

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

-9 0 0 RXMQWPUT Thread not connected

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

-10 0 0 RXMQWPUT Handle out of range

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

-11 0 0 RXMQWPUT Handle invalid

Explanation: The value of the handle supplied is not in the known range for a handle within the interface.

Example

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

ipmo.opt  =   MQGMO_NO_SYNCPOINT      + MQPMO_NO_CONTEXT ,
            + MQPMO_FAIL_IF_QUIESCING

imd.MSG   = MQMT_DATAGRAM
imd.per   = MQPER_NOT_PERSISTENT

rcc = RXMQW('PUT', hn1, '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.

RXMQWQSIGNAL

Description

This call is used after a proceeding Get.Signal (via RXMQW('GET') with the Options specifying MQGMO_SET_SIGNAL). RXMQW('QSIGNAL') is used to test the ECB (which is not externalised outside of RXMQW 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 RXMQW('OPEN') call, whereat the object was opened for Input **and** the last input operation was a Get.Signal (RXMQW('GET') with the Options specifying MQGMO_SET_SIGNAL).

Call

```
rcc = RXMQW('QSIGNAL', handle )
```

Additional Interface Return Codes and Messages

-1 0 0 RXMQWQSIG Bad number of Parms

Explanation: You **must** specify one parameter to the RXMQW('QSIGNAL') call.

-2 0 0 RXMQWQSIG Handle not supplied

Explanation: No value has been keyed for the first parameter, the handle representing the MQ object.

-3 0 0 RXMQWQSIG QM not connected

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

-4 0 0 RXMQWQSIG Thread not connected

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

-5 0 0 RXMQWQSIG Handle out of range

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

-6 0 0 RXMQWQSIG Handle invalid

Explanation: The value of the handle supplied is not in the known range for a handle within the interface.

0 0 0 RXMQWQSIG OK Signal not raised

Explanation: The Get.Signal has not yet completed

>0 0 0 RXMQWQSIG OK Signal raised

Explanation: The Get.Signal has completed for the reason given in the first word

Example

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

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

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

rcc = RXMQW('GET', hn1, 'message.', 'imd.', 'omd.', 'igmo.', 'ogmo.' )

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

do forever
   rcc = RXMQW('QSIGNAL', hn1 )
   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 = RXMQW('GET', hn1, '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 RXMQWQSIGNAL is used to permit processing whilst awaiting the arrival of a message.

RXMQWBROWSE

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 “RXMQWGET” on page 45.

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

Parameters

1. The Handle for the Queue obtained from a prior RXMQW('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 8 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

```
rcc = RXMQW('BROWSE', handle, 'Stem.Message.' )
```

Additional Interface Return Codes and Messages

-1 0 0 RXMQWBROWSE Bad number of Parms

Explanation: You **must** specify two parameters to the RXMQWBROWSE call.

'-2 0 0 RXMQWBROWSE Handle not supplied

Explanation: No value has been keyed for the first parameter, the handle representing the MQ object.

-3 0 0 RXMQWBROWSE 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 RXMQWBROWSE QM not connected

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

-5 0 0 RXMQWBROWSE Thread not connected

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

-6 0 0 RXMQWBROWSE Handle out of range

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

-7 0 0 RXMQWBROWSE Handle invalid

Explanation: The value of the handle supplied is not in the known range for a handle within the interface.

Example

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

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

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

message.0 = 100
message.1 = ''

rcc = RXMQW('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.

RXMQWHXT

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 8 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 8 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 = RXMQW('HXT', 'Stem.Message.', 'Stem.Splitup.' )
```

Additional Interface Return Codes and Messages

-1 0 0 RXMQWHXT Bad number of Parms

Explanation: You **must** specify two parameters to the RXMQWHXT call.

-2 0 0 RXMQWHXT 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 RXMQWHXT 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 RXMQWHXT No Data for Header Extraction

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

-5 0 0 RXMQWHXT Message is too short for an Header

Explanation: The input Stem.0 was <= 3, indicating no header in the message

-6 0 0 RXMQWHXT 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 RXMQWHXT 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 RXMQWHXT 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

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

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

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

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 = RXMQW('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.

RXMQWEVENT

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 8 for a warning about truncated messages used with this function. This message will have come from a prior RXMQW('BROWSE') or RXMQW('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 **RXMQW('GET')** rather than the RXMQW('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 10). 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 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 Event Queue. See “Message Lengths” on page 8 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, 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 = RXMQW('EVENT', 'Stem.Message.', 'Stem.Splitup.' )
```

Usage Notes

Bear in mind the following when using RXMQWEVENT:

- 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 2 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 RXMQWEVENT Bad number of Parms

Explanation: You **must** specify two parameters to the RXMQWEVENT call.

-2 0 0 RXMQWEVENT 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 RXMQWEVENT 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 RXMQWEVENT No Data for Event Extraction

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

-5 0 0 RXMQWEVENT 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 RXMQWEVENT 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 RXMQWEVENT 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 RXMQWEVENT 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 RXMQWEVENT 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 7. Event Names

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

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

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

Table 8 (Page 2 of 6). Event Names	
.item	**PCF Parameter Name**
PNS	MQCACF_PROCESS_NAMES
QNS	MQCACF_Q_NAMES
RECCNS	MQCACF_RECEIVER_CHANNEL_NAMES
REMQNS	MQCACF_REMOTE_Q_NAMES
REQCNS	MQCACF_REQUESTER_CHANNEL_NAMES
SENDCNS	MQCACF_SENDER_CHANNEL_NAMES
SERVCNS	MQCACF_SERVER_CHANNEL_NAMES
TCN	MQCACF_TO_CHANNEL_NAME
TPN	MQCACF_TO_PROCESS_NAME
TQN	MQCACF_TO_Q_NAME
UID	MQCACF_USER_IDENTIFIER
CN	MQCACH_CHANNEL_NAME
CNS	MQCACH_CHANNEL_NAMES
CSD	MQCACH_CHANNEL_START_DATE
CST	MQCACH_CHANNEL_START_TIME
CONN	MQCACH_CONNECTION_NAME
CLUWID	MQCACH_CURRENT_LUWID
DESC	MQCACH_DESC
FORMAT	MQCACH_FORMAT_NAME
LLUWID	MQCACH_LAST_LUWID
LMSGD	MQCACH_LAST_MSG_DATE
LMSGT	MQCACH_LAST_MSG_TIME
MCAJN	MQCACH_MCA_JOB_NAME
MCAN	MQCACH_MCA_NAME
MCAUID	MQCACH_MCA_USER_ID
MCAMODE	MQCACH_MODE_NAME
MRXITN	MQCACH_MR_EXIT_NAME
MRXITUD	MQCACH_MR_EXIT_USER_DATA
MSGXN	MQCACH_MSG_EXIT_NAME
MSGXUD	MQCACH_MSG_EXIT_USER_DATA
PASSWORD	MQCACH_PASSWORD
RCVXN	MQCACH_RCV_EXIT_NAME
RCVXUD	MQCACH_RCV_EXIT_USER_DATA
SECXN	MQCACH_SEC_EXIT_NAME
SECXUD	MQCACH_SEC_EXIT_USER_DATA
SENDXN	MQCACH_SEND_EXIT_NAME
SENDXUD	MQCACH_SEND_EXIT_USER_DATA

Table 8 (Page 3 of 6). Event Names

.item	PCF Parameter Name
TPNAME	MQCACH_TP_NAME
USERID	MQCACH_USER_ID
XQN	MQCACH_XMIT_Q_NAME
AT	MQIA_APPL_TYPE
AE	MQIA_AUTHORITY_EVENT
BT	MQIA_BACKOUT_THRESHOLD
CCSID	MQIA_CODED_CHAR_SET_ID
CLEV	MQIA_COMMAND_LEVEL
CPILEV	MQIA_CPI_LEVEL
CURDEPTH	MQIA_CURRENT_Q_DEPTH
DEFIO	MQIA_DEF_INPUT_OPEN_OPTION
DEFPER	MQIA_DEF_PERSISTENCE
DEFPRI	MQIA_DEF_PRIORITY
DEFTYPE	MQIA_DEFINITION_TYPE
HARDENGB	MQIA_HARDEN_GET_BACKOUT
HQD	MQIA_HIGH_Q_DEPTH
IE	MQIA_INHIBIT_EVENT
IGET	MQIA_INHIBIT_GET
IPUT	MQIA_INHIBIT_PUT
LOCEV	MQIA_LOCAL_EVENT
MAXH	MQIA_MAX_HANDLES
MAXMSG	MQIA_MAX_MSG_LENGTH
MAXPRI	MQIA_MAX_PRIORITY
MAXDEPTH	MQIA_MAX_Q_DEPTH
NUMUNC	MQIA_MAX_UNCOMMITTED_MSGS
DELSEQ	MQIA_MSG_DELIVERY_SEQUENCE
MDC	MQIA_MSG_DEQ_COUNT
MEC	MQIA_MSG_ENQ_COUNT
NAMEC	MQIA_NAME_COUNT
IOC	MQIA_OPEN_INPUT_COUNT
OOC	MQIA_OPEN_OUTPUT_COUNT
PEV	MQIA_PERFORMANCE_EVENT
PLATFORM	MQIA_PLATFORM
QDHEV	MQIA_Q_DEPTH_HIGH_EVENT
QDHLIM	MQIA_Q_DEPTH_HIGH_LIMIT
QDLEV	MQIA_Q_DEPTH_LOW_EVENT
QDLLIM	MQIA_Q_DEPTH_LOW_LIMIT

Table 8 (Page 4 of 6). Event Names

.item	PCF Parameter Name
QDMEV	MQIA_Q_DEPTH_MAX_EVENT
SI	MQIA_Q_SERVICE_INTERVAL
SIEV	MQIA_Q_SERVICE_INTERVAL_EVENT
QT	MQIA_Q_TYPE
REV	MQIA_REMOTE_EVENT
RETINT	MQIA_RETENTION_INTERVAL
SCOPE	MQIA_SCOPE
SHARE	MQIA_SHAREABILITY
SSEV	MQIA_START_STOP_EVENT
SYNC	MQIA_SYNCPOINT
TSR	MQIA_TIME_SINCE_RESET
TRIGC	MQIA_TRIGGER_CONTROL
TRIGD	MQIA_TRIGGER_DEPTH
TRIGI	MQIA_TRIGGER_INTERVAL
MSGPRI	MQIA_TRIGGER_MSG_PRIORITY
TRIT	MQIA_TRIGGER_TYPE
USAGE	MQIA_USAGE
ALL	MQIACF_ALL
AED1	MQIACF_AUX_ERROR_DATA_INT_1
AED2	MQIACF_AUX_ERROR_DATA_INT_2
BRTYPE	MQIACF_BRIDGE_TYPE
CATTR	MQIACF_CHANNEL_ATTRS
COM	MQIACF_COMMAND
CONVRC	MQIACF_CONV_REASON_CODE
EID	MQIACF_ERROR_ID
ERRORID	MQIACF_ERROR_IDENTIFIER
ERROROF	MQIACF_ERROR_OFFSET
ESCTYPE	MQIACF_ESCAPE_TYPE
FORCE	MQIACF_FORCE
OOPTS	MQIACF_OPEN_OPTIONS
PARMID	MQIACF_PARAMETER_ID
PROCATTR	MQIACF_PROCESS_ATTRS
PURGE	MQIACF_PURGE
QATTR	MQIACF_Q_ATTRS
QMATTR	MQIACF_Q_MGR_ATTRS
QUIESCE	MQIACF_QUIESCE
RQUAL	MQIACF_REASON_QUALIFIER

Table 8 (Page 5 of 6). Event Names

.item	PCF Parameter Name
REPLACE	MQIACF_REPLACE
SELECTOR	MQIACF_SELECTOR
BATCHSZ	MQIACH_BATCH_SIZE
BATCHES	MQIACH_BATCHES
NBUFRECV	MQIACH_BUFFERS_RCVD
NBUFRECV	MQIACH_BUFFERS_RECEIVED
NBUFSENT	MQIACH_BUFFERS_SENT
NBYTERECV	MQIACH_BYTES_RCVD
NBYTESENT	MQIACH_BYTES_RECEIVED
NBYTERECV	MQIACH_BYTES_SENT
CED	MQIACH_CHANNEL_ERROR_DATA
CIATTR	MQIACH_CHANNEL_INSTANCE_ATTRS
CITYPE	MQIACH_CHANNEL_INSTANCE_TYPE
CSTATUS	MQIACH_CHANNEL_STATUS
CTABLE	MQIACH_CHANNEL_TABLE
CTYPE	MQIACH_CHANNEL_TYPE
CURMSG	MQIACH_CURRENT_MSGS
CURSEQ	MQIACH_CURRENT_SEQ_NUMBER
CURSEQ	MQIACH_CURRENT_SEQUENCE_NUMBER
CURDATACV	MQIACH_DATA_CONVERSION
CURDATAIN	MQIACH_DATA_COUNT
CURDISCIN	MQIACH_DISC_INTERVAL
IND	MQIACH_IN_DOUBT
INDSEQ	MQIACH_INDOUBT_STATUS
LASTSEQ	MQIACH_LAST_SEQUENCE_NUMBER
LRTRYL	MQIACH_LONG_RETRIES_LEFT
LTRY	MQIACH_LONG_RETRY
LTIME	MQIACH_LONG_TIMER
MAXMSGL	MQIACH_MAX_MSG_LENGTH
MCAST	MQIACH_MCA_STATUS
MCATYPE	MQIACH_MCA_TYPE
MRCOUNT	MQIACH_MR_COUNT
MRINT	MQIACH_MR_INTERVAL
MSGSEQNO	MQIACH_MSG_SEQUENCE_NUMBER
MSGS	MQIACH_MSGS
PUTAUTH	MQIACH_PUT_AUTHORITY
SEQNWRAP	MQIACH_SEQUENCE_NUMBER_WRAP

Table 8 (Page 6 of 6). Event Names

.item	PCF Parameter Name
SRTRYL	MQIACH_SHORT_RETRIES_LEFT
SRTRY	MQIACH_SHORT_RETRY
STIME	MQIACH_SHORT_TIMER
STOPREQ	MQIACH_STOP_REQUESTED
XPT	MQIACH_XMIT_PROTOCOL_TYPE
NA	MQIAV_NOT_APPLICABLE

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	INGET	INPUT	LLAQT	LLABQ	LLUON	CHCONV	CHSTRT	CHSTOP	PFQDH	PFQDL	PFQFU	PFQSH	PFQSO	RMDXQT	RMDXQU	RMQUTY	RMRQNA	RMXQTY	RMXQUS	RMUDXQ	RMURQM	RRUXQN	NAAUT	CHACT	CHNACT	BRSTRT	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

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

message.0 = n
message.1 = <EVENT Header><Event Data>

/* Clear the result variable */

drop x.

/* Split the message */

rcc = RXMQW('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            /* C4TO36N    */
say x.XQN           /* T4TO36N    */
```

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

See Figure 1 on page 11 for an example using ZLIST processing to cope with the variable format component names.

```
/* Explurge an Event */

message.0 = n
message.1 = <EVENT Header><Event Data>
drop x.
rcc = RXMQW('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 say x.qn   /* works - returned comp */
if ( x.BQN <> 'X.BQN' ) then say 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:

```
message.0 = n
message.1 = <EVENT Header><Event Data>
drop x.
rcc = RXMQW('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 2. Removing funny event data

RXMQWTM

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 RXMQWTM 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 8 for a warning about truncated messages used with this function.

  This is called Message Mode.

- If it does **not end** in a dot, then RXMQWTM 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 10) is provided so that the various extant components can be determined.

In Message Mode (a Trigger Message provided to RXMQWTM 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 RXMQWTM 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 3 on page 78), and Data Mode permits the use of Rexx Execs as Triggered Processes (see Figure 4 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 8 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 = RXMQW('TM', 'Stem.Message.', 'Stem.Splitup.' )


Data    Mode:

   rcc = RXMQW('TM', MQTMC2_data    , 'Stem.Splitup.' )
```

Additional Interface Return Codes and Messages

-1 0 0 RXMQWTMM Bad number of Parms

Explanation: You **must** specify two parameters to the RXMQWTM call.

-2 0 0 RXMQWTMM 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 or data representing a MQTMC2 structure to be parsed.

-3 0 0 RXMQWTMM 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 RXMQWTMM 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 data.

-5 0 0 RXMQWTMM No Data for TM Extraction

Explanation: The input Stem.0 was zero, indicating no message to process (message mode)

-6 0 0 RXMQWTMM 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 RXMQWTMM 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 RXMQWTMM 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 splitup (message mode)

-9 0 0 RXMQWTMM No Data for TM Extraction

Explanation: The input Stem.0 was zero, indicating no message to process (data mode)

-10 0 0 RXMQWTMM 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 RXMQWTMM 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 RXMQWTMM 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 splitup (data mode)

Trigger information

Table 10. Trigger Components

Stem. Component	MQTM/MQTMC2 Structure name	Number or Text
.QN	QName	T
.PN	ProcessName	T
.TD	TriggerData	T
.AT	ApplType	N
.AID	ApplId	T
.ED	EnvData	T
.UD	UserData	T
.QM	QMgrName	T
.PL	MQTMC2 parameter	T

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.0 = 684
message.1 = <MQTM>

/* Clear the result variable */

drop t.

/* Split the message */

rcc = RXMQW('TM', 'message.', 't.' )

/* on return, the following are set */

say t.QN            /* L3N1       */
say t.PN            /* P3TO46N    */

/* 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 3. A Trigger Monitor

This example shows how a message obtained from an Initiation Queue is splitup, 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 ones own Trigger Monitor.

```
/* Get the parm */

parse arg parm

/* Clear the result variable */

drop p.

/* Split the parm */

rcc = RXMQW('TM', parm, 'p.' )

/* on return, the following are set */

say p.QM            /* RAH1       */
say p.QN            /* L3N1       */
say p.PN            /* P3TO46N    */

/* Truncate areas for usage */

do j=1 to words(p.zlist)
   item    = word(p.zlist,j)
   p.item  = strip(p.item,'B')
   end
```

Figure 4. A Rexx Triggered Process

This example shows how a Rexx Exec being initiated via a Trigger Monitor accesses its passed data (as a Trigger Monitor for Netview has to be hand-coded, I'm assuming that you use the above exec).

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 MA1DTEST JCL file.

```
/* MA1DTEST Exec - a Rexx/MQ/Netview Example */

/* Initialise the interface */

RXMQWTRACE = ''

rcc= RXMQW('INIT')
say 'rc=' rcc

/* Connect to Queue Manager - VRH1 */

RXMQWTRACE = ''
rcc = RXMQW('CONN', 'VRH1')
say 'RC=' rcc

/* Open Queue N1 for Inquire Access Only, tracing Object Descriptor accesses */

iod.on = 'N1'
iod.ot = MQOT_Q

RXMQWTRACE = 'BOD MOD'
rcc = RXMQW('OPEN', 'iod.', mqoo_inquire, 'h1', 'ood.')
say 'RC=' rcc  'H=' h1

/* Open Queue P1 for Output and Browse Access, plus Attribute manipulation */

RXMQWTRACE = ''
oo  = mqoo_inquire+mqoo_output+mqoo_browse+mqoo_set
rcc = RXMQW('OPEN', 'P1', oo , 'h2', 'ood.')
say 'RC=' rcc  'H=' h2

/* Write a Persistent Message, within UOW, to Queue P1; trace everything */

RXMQWTRACE = 'PUT BMD MMD MPO BPO'
d.0      = 20
d.1      = time() '0123456789'
imd.PER  = MQPER_PERSISTENT
ipmo.opt = MQPMO_SYNCPOINT
rcc      = RXMQWPUT( h2,'d.','imd.','omd.','ipmo.','opmo.')
say 'RC=' rcc
```

Figure 5 (Part 1 of 4). Interface example

```
/* Inquire upon the number of Messages now in Queue P1 */

RXMQWTRACE = ''
atrin = mqia_current_q_depth
atrou = ''
rcc   = RXMQW('INQ', h2, atrin, 'atrou' )
say 'RC=' rcc 'Atr' atrin 'Setting <'atrou'>'

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

RXMQWTRACE = ''
atrin = mqca_q_name
atrou = ''
rcc   = RXMQW('INQ', h1, atrin, 'atrou' )
say 'RC=' rcc 'Atr' atrin 'Setting <'atrou'>'
/* Toggle the GETtability of a Queue, providing a change each time */

RXMQWTRACE = ''
atrsn = MQIA_INHIBIT_GET
atrsd = MQQA_GET_INHIBITED
rcc   = RXMQW('SET', h2, atrsn, atrsd)
say 'RC=' rcc

RXMQWTRACE = ''
atrin = mqia_inhibit_get ; atrou = ''
rcc   = RXMQW('INQ', h2, atrin, 'atrou' )
say 'RC=' rcc 'Atr' atrin 'Setting <'atrou'>'

RXMQWTRACE = ''
atrsn = MQIA_INHIBIT_GET
atrsd = MQQA_GET_ALLOWED
rcc   = RXMQW('SET', h2, atrsn, atrsd)
say 'RC=' rcc

RXMQWTRACE = ''
atrin = MQIA_INHIBIT_GET ; atrou = ''
rcc = RXMQW('INQ', h2, atrin, 'atrou' )
say 'RC=' rcc
say 'Atr' atrin 'Setting <'atrou'>'
```

Figure 5 (Part 2 of 4). Interface example

```
/* Set the Trigger Data for a Queue ... */

RXMQWTRACE = ''
atrsn = MQCA_TRIGGER_DATA
atrsd = 'RAH Trigger Data'
rcc   = RXMQW('SET', h2, atrsn, atrsd)
say 'RC=' rcc

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

RXMQWTRACE = ''
atrin = mqca_trigger_data
atrou = ''
rcc = RXMQW('INQ', h2, atrin, 'atrou' )
say 'RC=' rcc 'Atr' atrin 'Setting <'atrou'>'

/* Syncpoint all accesses to the QM */

RXMQWTRACE = ''
rcc = RXMQW('CMIT')
say 'RC=' rcc

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

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

/* Rollback a Unit of Work (empty in this case) */
RXMQWTRACE = ''
rcc = RXMQW('BACK')
say 'RC=' rcc

/* Stop access to a Queue */

RXMQWTRACE = ''
rcc = RXMQW('CLOSE', h2, mqco_none)
say 'RC=' rcc
```

Figure 5 (Part 3 of 4). Interface example

```
/* Re-open the P1 Queue for Browse only access */

RXMQWTRACE = ''
rcc = RXMQW('OPEN', 'P1', mqoo_browse, 'h3', 'ood.')
say 'RC=' rcc  'H=' h3

/* Browse the Queue using the Extension function */
RXMQWTRACE = ''
do i=1
   g.0 = 200
   g.1 = ''
   rcc = RXMQW('BROWSE', h3,'g.')
   say 'RC=' rcc
   say '@@@@@@@@@@@@@@@' i 'data <'g.1'> length' g.0
   if ( word(rcc,1) <> 0 ) then leave
   end

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


/* Issue a Bad command to show effect of -ve RC */
rcc = RXMQW('OPEN')
say 'Last Message <'RXMQW.LASTMSG'> Last call 'RXMQW.LASTOP,
    ' which ended with RC('RXMQW.LASTRC') and MQCC('RXMQW.LASTCC')',
    ' MQRC('RXMQW.LASTAC')'

/* Stop access to the Queue */

RXMQWTRACE = ''
rcc = RXMQW('CLOSE', h3, mqco_none)
say 'RC=' rcc

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

RXMQWTRACE = ''
rcc = RXMQW('DISC')
say 'RC=' rcc

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

RXMQWTRACE = 'TERM'
rcc = RXMQW('TERM')
say 'RC=' rcc

/* ... but leave the MQ_ constants around */
say 'MQPER_PERSISTENT' MQPER_PERSISTENT 'RC(2048) is' RXMQW.RCMAP.2048

/* End of MA1DTEST exec */
```

Figure 5 (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_MFS_MAP_NAME_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
MQAT_AIX
MQAT_CICS
MQAT_CICS_VSE
MQAT_DEFAULT
MQAT_DOS
MQAT_GUARDIAN
MQAT_IMS
MQAT_IMS_BRIDGE
MQAT_MVS
MQAT_NO_CONTEXT
MQAT_OS2
MQAT_OS400
MQAT_QMGR
MQAT_UNIX
MQAT_UNKNOWN
MQAT_USER_FIRST
MQAT_USER_LAST
MQAT_VMS
MQAT_VOS
MQAT_WINDOWS
MQAT_WINDOWS_NT
MQAT_XCF
MQBT_OTMA
MQCA_APPL_ID
MQCA_BACKOUT_REQ_Q_NAME
MQCA_BASE_Q_NAME
MQCA_COMMAND_INPUT_Q_NAME
MQCA_CREATION_DATE
MQCA_CREATION_TIME
MQCA_DEAD_LETTER_Q_NAME
MQCA_DEF_XMIT_Q_NAME
MQCA_ENV_DATA
MQCA_FIRST
MQCA_INITIATION_Q_NAME
MQCA_LAST
MQCA_LAST_USED
MQCA_NAMELIST_DESC
MQCA_NAMELIST_NAME
MQCA_NAMES
MQCA_PROCESS_DESC
MQCA_PROCESS_NAME
MQCA_Q_DESC
MQCA_Q_MGR_DESC
MQCA_Q_MGR_NAME
MQCA_Q_NAME
MQCA_REMOTE_Q_MGR_NAME
MQCA_REMOTE_Q_NAME
MQCA_STORAGE_CLASS
MQCA_TRIGGER_DATA
MQCA_USER_DATA
MQCA_XMIT_Q_NAME
MQCACF_ALIAS_Q_NAMES
MQCACF_APPL_NAME
MQCACF_AUX_ERROR_DATA_STR_1
MQCACF_AUX_ERROR_DATA_STR_2
MQCACF_AUX_ERROR_DATA_STR_3
MQCACF_BRIDGE_NAME
MQCACF_ESCAPE_TEXT
MQCACF_FIRST
MQCACF_FROM_CHANNEL_NAME
MQCACF_FROM_PROCESS_NAME
MQCACF_FROM_Q_NAME
MQCACF_LAST_USED
MQCACF_LOCAL_Q_NAMES
MQCACF_MODEL_Q_NAMES
MQCACF_OBJECT_Q_MGR_NAME
MQCACF_PROCESS_NAMES
MQCACF_Q_NAMES
MQCACF_RECEIVER_CHANNEL_NAMES
MQCACF_REMOTE_Q_NAMES
MQCACF_REQUESTER_CHANNEL_NAMES
MQCACF_SENDER_CHANNEL_NAMES
MQCACF_SERVER_CHANNEL_NAMES
MQCACF_TO_CHANNEL_NAME
MQCACF_TO_PROCESS_NAME
MQCACF_TO_Q_NAME
MQCACF_USER_IDENTIFIER
MQCACH_CHANNEL_NAME
MQCACH_CHANNEL_NAMES

MQCACH_CHANNEL_START_DATE
MQCACH_CHANNEL_START_TIME
MQCACH_CONNECTION_NAME
MQCACH_CURRENT_LUWID
MQCACH_DESC
MQCACH_FIRST
MQCACH_FORMAT_NAME
MQCACH_LAST_LUWID
MQCACH_LAST_MSG_DATE
MQCACH_LAST_MSG_TIME
MQCACH_LAST_USED
MQCACH_MCA_JOB_NAME
MQCACH_MCA_NAME
MQCACH_MCA_USER_ID
MQCACH_MODE_NAME
MQCACH_MR_EXIT_NAME
MQCACH_MR_EXIT_USER_DATA
MQCACH_MSG_EXIT_NAME
MQCACH_MSG_EXIT_USER_DATA
MQCACH_PASSWORD
MQCACH_RCV_EXIT_NAME
MQCACH_RCV_EXIT_USER_DATA
MQCACH_SEC_EXIT_NAME
MQCACH_SEC_EXIT_USER_DATA
MQCACH_SEND_EXIT_NAME
MQCACH_SEND_EXIT_USER_DATA
MQCACH_TP_NAME
MQCACH_USER_ID
MQCACH_XMIT_Q_NAME
MQCC_FAILED
MQCC_OK
MQCC_UNKNOWN
MQCC_WARNING
MQCCSI_DEFAULT
MQCCSI_EMBEDDED
MQCCSI_Q_MGR
MQCFC_LAST
MQCFC_NOT_LAST
MQCFH_STRUC_LENGTH
MQCFH_VERSION_1
MQCFIL_STRUC_LENGTH_FIXED
MQCFIN_STRUC_LENGTH
MQCFSL_STRUC_LENGTH_FIXED
MQCFST_STRUC_LENGTH_FIXED
MQCFT_COMMAND
MQCFT_EVENT
MQCFT_INTEGER
MQCFT_INTEGER_LIST
MQCFT_RESPONSE
MQCFT_STRING
MQCFT_STRING_LIST
MQCHIDS_INDOUBT
MQCHIDS_NOT_INDOUBT
MQCHS_BINDING
MQCHS_PAUSED
MQCHS_REQUESTING
MQCHS_RETRYING
MQCHS_RUNNING
MQCHS_STARTING
MQCHS_STOPPED
MQCHS_STOPPING
MQCHSR_STOP_NOT_REQUESTED
MQCHSR_STOP_REQUESTED
MQCHTAB_CLNTCONN
MQCHTAB_Q_MGR
MQCMD_CHANGE_CHANNEL
MQCMD_CHANGE_PROCESS
MQCMD_CHANGE_Q
MQCMD_CHANGE_Q_MGR
MQCMD_CHANNEL_EVENT
MQCMD_CLEAR_Q
MQCMD_COPY_CHANNEL
MQCMD_COPY_PROCESS
MQCMD_COPY_Q
MQCMD_CREATE_CHANNEL
MQCMD_CREATE_PROCESS
MQCMD_CREATE_Q
MQCMD_DELETE_CHANNEL
MQCMD_DELETE_PROCESS
MQCMD_DELETE_Q
MQCMD_ESCAPE
MQCMD_INQUIRE_CHANNEL
MQCMD_INQUIRE_CHANNEL_NAMES
MQCMD_INQUIRE_CHANNEL_STATUS
MQCMD_INQUIRE_PROCESS
MQCMD_INQUIRE_PROCESS_NAMES
MQCMD_INQUIRE_Q
MQCMD_INQUIRE_Q_MGR
MQCMD_INQUIRE_Q_NAMES
MQCMD_PERFM_EVENT
MQCMD_PING_CHANNEL
MQCMD_PING_Q_MGR
MQCMD_Q_MGR_EVENT
MQCMD_RESET_CHANNEL
MQCMD_RESET_Q_STATS
MQCMD_RESOLVE_CHANNEL
MQCMD_START_CHANNEL
MQCMD_START_CHANNEL_INIT
MQCMD_START_CHANNEL_LISTENER
MQCMD_STOP_CHANNEL
MQCMDL_LEVEL_1
MQCMDL_LEVEL_114
MQCMDL_LEVEL_200
MQCMDL_LEVEL_201
MQCMDL_LEVEL_221
MQCMDL_LEVEL_320
MQCO_DELETE
MQCO_DELETE_PURGE
MQCO_NONE
MQDLH_VERSION_1
MQEC_CONNECTION_QUIESCING
MQEC_MSG_ARRIVED
MQEC_Q_MGR_QUIESCING
MQEC_WAIT_CANCELED
MQEC_WAIT_INTERVAL_EXPIRED
MQEI_UNLIMITED
MQENC_DECIMAL_MASK
MQENC_DECIMAL_NORMAL
MQENC_DECIMAL_REVERSED
MQENC_DECIMAL_UNDEFINED
MQENC_FLOAT_IEEE_NORMAL
MQENC_FLOAT_IEEE_REVERSED
MQENC_FLOAT_MASK
MQENC_FLOAT_S390
MQENC_FLOAT_UNDEFINED
MQENC_INTEGER_MASK
MQENC_INTEGER_NORMAL
MQENC_INTEGER_REVERSED
MQENC_INTEGER_UNDEFINED
MQENC_NATIVE
MQENC_RESERVED_MASK
MQET_MQSC
MQEVR_DISABLED
MQEVR_ENABLED
MQFB_APPL_CANNOT_BE_STARTED
MQFB_APPL_FIRST

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_NONE
MQFMT_PCF
MQFMT_STRING
MQFMT_TRIGGER
MQFMT_XMIT_Q_HEADER
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_SET_SIGNAL
MQGMO_SYNCPOINT
MQGMO_VERSION_1
MQGMO_WAIT
MQHC_DEF_HCONN
MQIA_APPL_TYPE
MQIA_AUTHORITY_EVENT
MQIA_BACKOUT_THRESHOLD
MQIA_CODED_CHAR_SET_ID
MQIA_COMMAND_LEVEL
MQIA_CPI_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_DEQ_COUNT
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_HIGH_LIMIT
MQIA_Q_DEPTH_LOW_EVENT
MQIA_Q_DEPTH_LOW_LIMIT
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_QUIESCE
MQIACF_REASON_QUALIFIER
MQIACF_REPLACE
MQIACF_SELECTOR
MQIACH_BATCH_SIZE
MQIACH_BATCHES
MQIACH_BUFFERS_RCVD

MQIACH_BUFFERS_RECEIVED
MQIACH_BUFFERS_SENT
MQIACH_BYTES_RCVD
MQIACH_BYTES_RECEIVED
MQIACH_BYTES_SENT
MQIACH_CHANNEL_ERROR_DATA
MQIACH_CHANNEL_INSTANCE_ATTRS
MQIACH_CHANNEL_INSTANCE_TYPE
MQIACH_CHANNEL_STATUS
MQIACH_CHANNEL_TABLE
MQIACH_CHANNEL_TYPE
MQIACH_CURRENT_MSGS
MQIACH_CURRENT_SEQ_NUMBER
MQIACH_CURRENT_SEQUENCE_NUMBER
MQIACH_DATA_CONVERSION
MQIACH_DATA_COUNT
MQIACH_DISC_INTERVAL
MQIACH_FIRST
MQIACH_IN_DOUBT
MQIACH_INDOUBT_STATUS
MQIACH_LAST_SEQ_NUMBER
MQIACH_LAST_SEQUENCE_NUMBER
MQIACH_LAST_USED
MQIACH_LONG_RETRIES_LEFT
MQIACH_LONG_RETRY
MQIACH_LONG_TIMER
MQIACH_MAX_MSG_LENGTH
MQIACH_MCA_STATUS
MQIACH_MCA_TYPE
MQIACH_MR_COUNT
MQIACH_MR_INTERVAL
MQIACH_MSG_SEQUENCE_NUMBER
MQIACH_MSGS
MQIACH_PUT_AUTHORITY
MQIACH_SEQUENCE_NUMBER_WRAP
MQIACH_SHORT_RETRIES_LEFT
MQIACH_SHORT_RETRY
MQIACH_SHORT_TIMER
MQIACH_STOP_REQUESTED
MQIACH_XMIT_PROTOCOL_TYPE
MQIAUT_NONE
MQIAV_NOT_APPLICABLE
MQICM_COMMIT_THEN_SEND
MQICM_SEND_THEN_COMMIT
MQIDO_BACKOUT
MQIDO_COMMIT
MQIIH_LENGTH_1
MQIIH_NONE
MQIIH_VERSION_1
MQISS_CHECK
MQISS_FULL
MQITS_IN_CONVERSATION
MQITS_NOT_IN_CONVERSATION
MQMCAS_RUNNING
MQMCAS_STOPPED
MQMD_VERSION_1
MQMDS_FIFO
MQMDS_PRIORITY
MQMT_APPL_FIRST
MQMT_APPL_LAST
MQMT_DATAGRAM
MQMT_REPLY
MQMT_REPORT
MQMT_REQUEST
MQMT_SYSTEM_FIRST
MQMT_SYSTEM_LAST
MQOD_VERSION_1
MQOO_ALTERNATE_USER_AUTHORITY
MQOO_BROWSE
MQOO_FAIL_IF_QUIESCING
MQOO_INPUT_AS_Q_DEF
MQOO_INPUT_EXCLUSIVE
MQOO_INPUT_SHARED
MQOO_INQUIRE
MQOO_OUTPUT
MQOO_PASS_ALL_CONTEXT
MQOO_PASS_IDENTITY_CONTEXT
MQOO_SAVE_ALL_CONTEXT
MQOO_SET
MQOO_SET_ALL_CONTEXT
MQOO_SET_IDENTITY_CONTEXT
MQOT_ALIAS_Q
MQOT_ALL
MQOT_CHANNEL
MQOT_CURRENT_CHANNEL
MQOT_LOCAL_Q
MQOT_MODEL_Q
MQOT_NAMELIST
MQOT_PROCESS
MQOT_Q
MQOT_Q_MGR
MQOT_RECEIVER_CHANNEL
MQOT_REMOTE_Q
MQOT_REQUESTER_CHANNEL
MQOT_RESERVED_1
MQOT_SAVED_CHANNEL
MQOT_SENDER_CHANNEL
MQOT_SERVER_CHANNEL
MQPER_NOT_PERSISTENT
MQPER_PERSISTENCE_AS_Q_DEF
MQPER_PERSISTENT
MQPL_AIX
MQPL_MVS
MQPL_NATIVE
MQPL_OS2
MQPL_OS400
MQPL_UNIX
MQPL_WINDOWS_NT
MQPMO_ALTERNATE_USER_AUTHORITY
MQPMO_DEFAULT_CONTEXT
MQPMO_FAIL_IF_QUIESCING
MQPMO_NO_CONTEXT
MQPMO_NO_SYNCPOINT
MQPMO_NONE
MQPMO_PASS_ALL_CONTEXT
MQPMO_PASS_IDENTITY_CONTEXT
MQPMO_SET_ALL_CONTEXT
MQPMO_SET_IDENTITY_CONTEXT
MQPMO_SYNCPOINT
MQPMO_VERSION_1
MQPO_NO
MQPO_YES
MQPRI_PRIORITY_AS_Q_DEF
MQQA_BACKOUT_HARDENED
MQQA_BACKOUT_NOT_HARDENED
MQQA_GET_ALLOWED
MQQA_GET_INHIBITED
MQQA_NOT_SHAREABLE
MQQA_PUT_ALLOWED
MQQA_PUT_INHIBITED
MQQA_SHAREABLE
MQQDT_PERMANENT_DYNAMIC
MQQDT_PREDEFINED
MQQDT_TEMPORARY_DYNAMIC
MQQO_NO
MQQO_YES

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_DEAD_LETTER_Q
MQRO_DISCARD_MSG
MQRO_EXCEPTION
MQRO_EXCEPTION_WITH_DATA
MQRO_EXCEPTION_WITH_FULL_DATA
MQRO_EXPIRATION
MQRO_EXPIRATION_WITH_DATA
MQRO_EXPIRATION_WITH_FULL_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_NOT_AUTHORIZED
MQRQ_CMD_NOT_AUTHORIZED
MQRQ_CONN_NOT_AUTHORIZED
MQRQ_OPEN_NOT_AUTHORIZED
MQRQ_Q_MGR_QUIESCING
MQRQ_Q_MGR_STOPPING
MQSCO_CELL
MQSCO_Q_MGR
MQSP_AVAILABLE
MQSP_NOT_AVAILABLE
MQTC_OFF
MQTC_ON
MQTM_VERSION_1
MQTT_DEPTH
MQTT_EVERY
MQTT_FIRST
MQTT_NONE
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_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_ATTR_COUNT_ERROR
MQRC_INT_ATTR_COUNT_TOO_SMALL
MQRC_INT_ATTRS_ARRAY_ERROR
MQRC_MAX_CONNS_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_TOO_BIG_FOR_Q_MGR
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_OUTPUT
MQRC_NOT_OPEN_FOR_PASS_ALL
MQRC_NOT_OPEN_FOR_PASS_IDENT
MQRC_NOT_OPEN_FOR_SET
MQRC_NOT_OPEN_FOR_SET_ALL
MQRC_NOT_OPEN_FOR_SET_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_PAGESET_ERROR
MQRC_PAGESET_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
MQRC_RESOURCE_PROBLEM
MQRC_SECOND_MARK_NOT_ALLOWED

MQRC_SECURITY_ERROR
MQRC_SELECTOR_COUNT_ERROR
MQRC_SELECTOR_ERROR
MQRC_SELECTOR_LIMIT_EXCEEDED
MQRC_SELECTOR_NOT_FOR_TYPE
MQRC_SERVICE_ERROR
MQRC_SERVICE_NOT_AVAILABLE
MQRC_SIGNAL_OUTSTANDING
MQRC_SIGNAL_REQUEST_ACCEPTED
MQRC_SIGNAL1_ERROR
MQRC_SOURCE_BUFFER_ERROR
MQRC_SOURCE_CCSID_ERROR
MQRC_SOURCE_DECIMAL_ENC_ERROR
MQRC_SOURCE_FLOAT_ENC_ERROR
MQRC_SOURCE_INTEGER_ENC_ERROR
MQRC_SOURCE_LENGTH_ERROR
MQRC_STORAGE_CLASS_ERROR
MQRC_STORAGE_NOT_AVAILABLE
MQRC_SUPPRESSED_BY_EXIT
MQRC_SYNCPOINT_LIMIT_REACHED
MQRC_SYNCPOINT_NOT_AVAILABLE
MQRC_TARGET_BUFFER_ERROR
MQRC_TARGET_CCSID_ERROR
MQRC_TARGET_DECIMAL_ENC_ERROR
MQRC_TARGET_FLOAT_ENC_ERROR
MQRC_TARGET_INTEGER_ENC_ERROR
MQRC_TARGET_LENGTH_ERROR
MQRC_TERMINATION_FAILED
MQRC_TRIGGER_CONTROL_ERROR
MQRC_TRIGGER_DEPTH_ERROR
MQRC_TRIGGER_MSG_PRIORITY_ERR
MQRC_TRIGGER_TYPE_ERROR
MQRC_TRUNCATED
MQRC_TRUNCATED_MSG_ACCEPTED
MQRC_TRUNCATED_MSG_FAILED
MQRC_UNEXPECTED_ERROR
MQRC_UNKNOWN_ALIAS_BASE_Q
MQRC_UNKNOWN_AUTH_ENTITY
MQRC_UNKNOWN_DEF_XMIT_Q
MQRC_UNKNOWN_ENTITY
MQRC_UNKNOWN_OBJECT_NAME
MQRC_UNKNOWN_OBJECT_Q_MGR
MQRC_UNKNOWN_Q_NAME
MQRC_UNKNOWN_REF_OBJECT
MQRC_UNKNOWN_REMOTE_Q_MGR
MQRC_UNKNOWN_REPORT_OPTION
MQRC_UNKNOWN_XMIT_Q
MQRC_USER_ID_NOT_AVAILABLE
MQRC_WAIT_INTERVAL_ERROR
MQRC_XMIT_Q_TYPE_ERROR
MQRC_XMIT_Q_USAGE_ERROR
MQRC_XWAIT_CANCELED
MQRC_XWAIT_ERROR
MQRCCF_ALLOCATE_FAILED
MQRCCF_ATTR_VALUE_ERROR
MQRCCF_BATCH_SIZE_ERROR
MQRCCF_BIND_FAILED
MQRCCF_CCSID_ERROR
MQRCCF_CELL_DIR_NOT_AVAILABLE
MQRCCF_CFH_COMMAND_ERROR
MQRCCF_CFH_CONTROL_ERROR
MQRCCF_CFH_LENGTH_ERROR
MQRCCF_CFH_MSG_SEQ_NUMBER_ERR
MQRCCF_CFH_PARM_COUNT_ERROR
MQRCCF_CFH_TYPE_ERROR
MQRCCF_CFH_VERSION_ERROR
MQRCCF_CFIL_COUNT_ERROR
MQRCCF_CFIL_DUPLICATE_VALUE
MQRCCF_CFIL_LENGTH_ERROR
MQRCCF_CFIL_PARM_ID_ERROR
MQRCCF_CFIN_DUPLICATE_PARM
MQRCCF_CFIN_LENGTH_ERROR
MQRCCF_CFIN_PARM_ID_ERROR
MQRCCF_CFST_DUPLICATE_PARM
MQRCCF_CFST_LENGTH_ERROR
MQRCCF_CFST_PARM_ID_ERROR
MQRCCF_CFST_STRING_LENGTH_ERR
MQRCCF_CHANNEL_ALREADY_EXISTS
MQRCCF_CHANNEL_DISABLED
MQRCCF_CHANNEL_IN_USE
MQRCCF_CHANNEL_INDOUBT
MQRCCF_CHANNEL_NAME_ERROR
MQRCCF_CHANNEL_NOT_ACTIVE
MQRCCF_CHANNEL_NOT_FOUND
MQRCCF_CHANNEL_TABLE_ERROR
MQRCCF_CHANNEL_TYPE_ERROR
MQRCCF_CHL_INST_TYPE_ERROR
MQRCCF_CHL_STATUS_NOT_FOUND
MQRCCF_COMMAND_FAILED
MQRCCF_COMMIT_FAILED
MQRCCF_CONFIGURATION_ERROR
MQRCCF_CONN_NAME_ERROR
MQRCCF_CONNECTION_CLOSED
MQRCCF_CONNECTION_REFUSED
MQRCCF_DATA_CONV_VALUE_ERROR
MQRCCF_DATA_TOO_LARGE
MQRCCF_DISC_INT_ERROR
MQRCCF_DISC_INT_WRONG_TYPE
MQRCCF_DYNAMIC_Q_SCOPE_ERROR
MQRCCF_ENCODING_ERROR
MQRCCF_ENTRY_ERROR
MQRCCF_ESCAPE_TYPE_ERROR
MQRCCF_FORCE_VALUE_ERROR
MQRCCF_HOST_NOT_AVAILABLE
MQRCCF_INDOUBT_VALUE_ERROR
MQRCCF_LIKE_OBJECT_WRONG_TYPE
MQRCCF_LISTENER_NOT_STARTED
MQRCCF_LONG_RETRY_ERROR
MQRCCF_LONG_RETRY_WRONG_TYPE
MQRCCF_LONG_TIMER_ERROR
MQRCCF_LONG_TIMER_WRONG_TYPE
MQRCCF_MAX_MSG_LENGTH_ERROR
MQRCCF_MCA_NAME_ERROR
MQRCCF_MCA_NAME_WRONG_TYPE
MQRCCF_MCA_TYPE_ERROR
MQRCCF_MD_FORMAT_ERROR
MQRCCF_MISSING_CONN_NAME
MQRCCF_MQCONN_FAILED
MQRCCF_MQGET_FAILED
MQRCCF_MQINQ_FAILED
MQRCCF_MQOPEN_FAILED
MQRCCF_MQPUT_FAILED
MQRCCF_MQSET_FAILED
MQRCCF_MR_COUNT_ERROR
MQRCCF_MR_COUNT_WRONG_TYPE
MQRCCF_MR_EXIT_NAME_ERROR
MQRCCF_MR_EXIT_NAME_WRONG_TYPE
MQRCCF_MR_INTERVAL_ERROR
MQRCCF_MR_INTERVAL_WRONG_TYPE
MQRCCF_MSG_EXIT_NAME_ERROR
MQRCCF_MSG_LENGTH_ERROR
MQRCCF_MSG_SEQ_NUMBER_ERROR
MQRCCF_MSG_TRUNCATED
MQRCCF_NO_COMMS_MANAGER
MQRCCF_NO_STORAGE
MQRCCF_NOT_XMIT_Q

MQRCCF_OBJECT_ALREADY_EXISTS
MQRCCF_OBJECT_NAME_ERROR
MQRCCF_OBJECT_OPEN
MQRCCF_OBJECT_WRONG_TYPE
MQRCCF_PARM_COUNT_TOO_BIG
MQRCCF_PARM_COUNT_TOO_SMALL
MQRCCF_PARM_SEQUENCE_ERROR
MQRCCF_PING_DATA_COMPARE_ERROR
MQRCCF_PING_DATA_COUNT_ERROR
MQRCCF_PING_ERROR
MQRCCF_PURGE_VALUE_ERROR
MQRCCF_PUT_AUTH_ERROR
MQRCCF_PUT_AUTH_WRONG_TYPE
MQRCCF_Q_ALREADY_IN_CELL
MQRCCF_Q_TYPE_ERROR
MQRCCF_Q_WRONG_TYPE
MQRCCF_QUIESCE_VALUE_ERROR
MQRCCF_RCV_EXIT_NAME_ERROR
MQRCCF_RECEIVE_FAILED
MQRCCF_RECEIVED_DATA_ERROR
MQRCCF_REMOTE_QM_TERMINATING
MQRCCF_REMOTE_QM_UNAVAILABLE
MQRCCF_REPLACE_VALUE_ERROR
MQRCCF_SEC_EXIT_NAME_ERROR
MQRCCF_SEND_EXIT_NAME_ERROR
MQRCCF_SEND_FAILED
MQRCCF_SEQ_NUMBER_WRAP_ERROR
MQRCCF_SHORT_RETRY_ERROR
MQRCCF_SHORT_RETRY_WRONG_TYPE
MQRCCF_SHORT_TIMER_ERROR
MQRCCF_SHORT_TIMER_WRONG_TYPE
MQRCCF_STRUCTURE_TYPE_ERROR
MQRCCF_TERMINATED_BY_SEC_EXIT
MQRCCF_UNKNOWN_Q_MGR
MQRCCF_UNKNOWN_REMOTE_CHANNEL
MQRCCF_USER_EXIT_NOT_AVAILABLE
MQRCCF_XMIT_PROTOCOL_TYPE_ERR
MQRCCF_XMIT_Q_NAME_ERROR
MQRCCF_XMIT_Q_NAME_WRONG_TYPE

(end of document)

Sending your comments to IBM

MA1D: A Netview Rexx interface for MQSeries for MVS/ESA
Version 2.0

MA1D 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

MA1D: A Netview Rexx interface for MQSeries for MVS/ESA
Version 2.0

MA1D 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.).

1 Cut along this line

2 Fold along this line

By air mail
Par avion

IBRS/CCRI NUMBER: PHQ - D/1348/SO

NE PAS AFFRANCHIR
NO STAMP REQUIRED

IBM

REPONSE PAYEE
GRANDE-BRETAGNE

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

3 Fold along this line

From: Name _____
Company or Organization _____
Address _____
_____
EMAIL _____
Telephone _____

1 Cut along this line

4 Fasten here with adhesive tape