ASG-Manager Products™ REXX Interface User's Guide

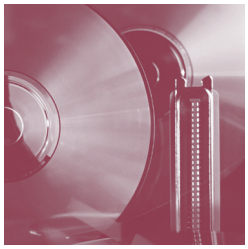
Version: 2.5.1

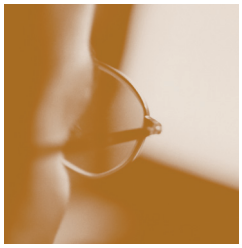
Publication Number: MPR0200-251-REXX

Publication Date: November 2001

The information contained herein is the confidential and proprietary information of Allen Systems Group, Inc. Unauthorized use of this information and disclosure to third parties is expressly prohibited. This technical publication may not be reproduced in whole or in part, by any means, without the express written consent of Allen Systems Group, Inc.

© 1998-2001 Allen Systems Group, Inc. All rights reserved.
All names and products contained herein are the trademarks or registered trademarks of their respective holders.





ASG Worldwide Headquarters Naples, Florida USA | asg.com
1333 Third Avenue South, Naples, Florida 34102 USA Tel: 941.435.2200 Fax: 941.263.3692 Toll Free: 1.800.932.5536

ASG Documentation/Product Enhancement Fax Form

Please FAX comments regarding ASG products and/or documentation to (941) 263-3692.

Company Name	Telephone Number	Site ID	Contact name

Product Name/Publication	Version #	Publication Date
Product:		
Publication:		
Tape VOLSER:		

Enhancement Request:

© 2001 Allen Systems Group, Inc.
All names and products are trademarks or registered trademarks of their respective holders.

ASG Support Numbers

ASG provides support throughout the world to resolve questions or problems regarding installation, operation, or use of our products. We provide all levels of support during normal business hours and emergency support during non-business hours. To expedite response time, please follow these procedures.

Please have this information ready:

- Product name, version number, and release number
- List of any fixes currently applied
- Any alphanumeric error codes or messages written precisely or displayed
- A description of the specific steps that immediately preceded the problem
- The severity code (ASG Support uses an escalated severity system to prioritize service to our clients. The severity codes and their meanings are listed below.)
- Verify whether you received an ASG Service Pack for this product. It may include information to help you resolve questions regarding installation of this ASG product. The Service Pack instructions are in a text file on the distribution media included with the Service Pack.

If You Receive a Voice Mail Message:

1. Follow the instructions to report a production-down or critical problem.
2. Leave a detailed message including your name and phone number. A Support representative will be paged and will return your call as soon as possible.
3. Please have the information described above ready for when you are contacted by the Support representative.

Severity Codes and Expected Support Response Times

Severity	Meaning	Expected Support Response Time
1	Production down, critical situation	Within 30 minutes
2	Major component of product disabled	Within 2 hours
3	Problem with the product, but customer has work-around solution	Within 4 hours
4	"How-to" questions and enhancement requests	Within 4 hours

ASG provides software products that run in a number of third-party vendor environments. Support for all non-ASG products is the responsibility of the respective vendor. In the event a vendor discontinues support for a hardware and/or software product, ASG cannot be held responsible for problems arising from the use of that unsupported version.

Business Hours Support

Your Location	Phone	Fax	E-mail
United States and Canada	800.354.3578 1.941.435.2201 **Secondary Numbers:** 800.227.7774 800.525.7775	941.263.2883	support@asg.com
Australia	61.2.9460.0411	61.2.9460.0280	support.au@asg.com
England	44.1727.736305	44.1727.812018	support.uk@asg.com
France	33.141.028590	33.141.028589	support.fr@asg.com
Germany	49.89.45716.300	49.89.45716.400	support.de@asg.com
Singapore	65.224.3080	65.224.8516	support.sg@asg.com
All other countries:	1.941.435.2201		support@asg.com

Non-Business Hours - Emergency Support

Your Location	Phone	Your Location	Phone
United States and Canada	800.354.3578 1.941.435.2201 **Secondary Numbers:** 800.227.7774 800.525.7775 **Fax:** 941.263.2883		
Asia	011.65.224.3080	**Japan/Telecom**	0041.800.9932.5536
Australia	0011.800.9932.5536	**New Zealand**	00.800.9932.5536
Denmark	00.800.9932.5536	**South Korea**	001.800.9932.5536
France	00.800.9932.5536	**Sweden/Telia**	009.800.9932.5536
Germany	00.800.9932.5536	**Switzerland**	00.800.9932.5536
Hong Kong	001.800.9932.5536	**Thailand**	001.800.9932.5536
Ireland	00.800.9932.5536	**United Kingdom**	00.800.9932.5536
Israel/Bezeq	014.800.9932.5536		
Japan/IDC	0061.800.9932.5536	**All other countries**	1.941.435.2201

ASG Web Site

Visit http://www.asg.com, ASG's World Wide Web site.

Submit all product and documentation suggestions to ASG's product management team at http://www.asg.com/products/suggestions.asp

If you do not have access to the web, FAX your suggestions to product management at (941) 263-3692. Please include your name, company, work phone, e-mail ID, and the name of the ASG product you are using. For documentation suggestions include the publication number located on the publication's front cover.

Contents

Preface iii

About this Publication iv

Publication Conventions v

1 The REXX Interface for the Manager Products Repository 1

2 Overview of the REXX Interface 3

Interface Modes 4
TEMPORARY Mode 4
PERMANENT Mode 5
RESIDENT Mode 5
SERVER Mode 5

Selecting Modes 6
Initializing with REXXMSPI 6
Activating SERVER Mode 9
Executing LOGON with the Initialization Call 10

3 Differences for ISPF (RESIDENT and PERMANENT Modes) 13

Using the SHUTDOWN Command 14

Starting REXX Procdures 15

4 Persistent Sessions (SERVER Mode) 17

Creating Persistent Sessions 17

Terminating and Resuming Applications 18

5 Syntax of the MPIO Call 21

6 REXX Interface Variables 23

7 The OUTMAP Function 25

8 Variable Services for the Communication with MMR/CMR 29

9 DACCESS from Within the REXX Application. 31

10 The SWAP Function 41

Appendix A
DCONTROL Area 45

Appendix B
Return Codes of MethodManager/ControlManager 47

Appendix C
Return Codes. 49

Index. 53

Preface

This *ASG-Manager Products REXX Interface User's Guide* provides information on using the REXX interface with ASG-Manager Products (herein called Manager Products).

Within the Manager Family of Program Products (excluding ASG-MethodManager, herein called MethodManager), ASG-ControlManager (herein called ControlManager) and ASG-DictionaryManager (herein called DictionaryManager) are co-requisites of each other. Both are environmental prerequisites (EPR) in as much as they must be at the latest version and release Level for each and every other Manager Product to execute correctly. This EPR rule applies to Manager Products in both Mainframe Environments (MFE) and Programmable Workstation Environments (PWSE). ControlManager and DictionaryManager complement each other in providing a gateway environment to Open Systems Interconnection (OSI) across information engineering techniques and dictionaries/directories/repositories from ASG and other vendors. Thus, ControlManager and DictionaryManager enable Manager Products users to position themselves to take full advantage of the Manager Products family in providing a Computer Aided Software Engineering (CASE) environment.

Allen Systems Group, Inc. (ASG) provides professional support to resolve any questions or concerns regarding the installation or use of any ASG product. Telephone technical support is available around the world, 24 hours a day, 7 days a week.

ASG welcomes your comments, as a preferred or prospective customer, on this publication or on any ASG product.

About this Publication

This publication consists of these chapters:

- Chapter 1, "The REXX Interface for the Manager Products Repository," provides an overview of how the REXX interface works with Manager Products repositories.
- Chapter 2, "Overview of the REXX Interface," describes the components the REXX interface uses.
- Chapter 3, "Differences for ISPF (RESIDENT and PERMANENT Modes)," describes the ISPF differences for RESIDENT/PERMANENT modes.
- Chapter 4, "Persistent Sessions (SERVER Mode)," describes the SERVER mode used to create persistent sessions.
- Chapter 5, "Syntax of the MPIO Call," describes the syntax and functions for the MPIO statement.
- Chapter 6, "REXX Interface Variables," describes the variables available in the REXX interface.
- Chapter 7, "The OUTMAP Function," describes using the output buffer function.
- Chapter 8, "Variable Services for the Communication with MMR/CMR," describes the variable interface for setting and reading DYR variables using the REXX interface.
- Chapter 9, "DACCESS from Within the REXX Application," describes the DACCESS function.
- Chapter 10, "The SWAP Function," describes the SWAP method used with MethodManager.

Publication Conventions

ASG uses these conventions in technical publications:

Convention	Represents
ALL CAPITALS	Directory, path, file, dataset, member, database, program, command, and parameter names.
Initial Capitals on Each Word	Window, field, field group, check box, button, panel (or screen), option names, and names of keys. A plus sign (+) is inserted for key combinations (e.g., Alt+Tab).
`lowercase italic monospace`	Information that you provide according to your particular situation. For example, you would replace *`filename`* with the actual name of the file.
`Monospace`	Characters you must type exactly as they are shown. Code, JCL, file listings, or command/statement syntax. Also used for denoting brief examples in a paragraph.
Vertical Separator Bar (|) with underline	Options available with the default value underlined (e.g., Y|N).

1 The REXX Interface for the Manager Products Repository

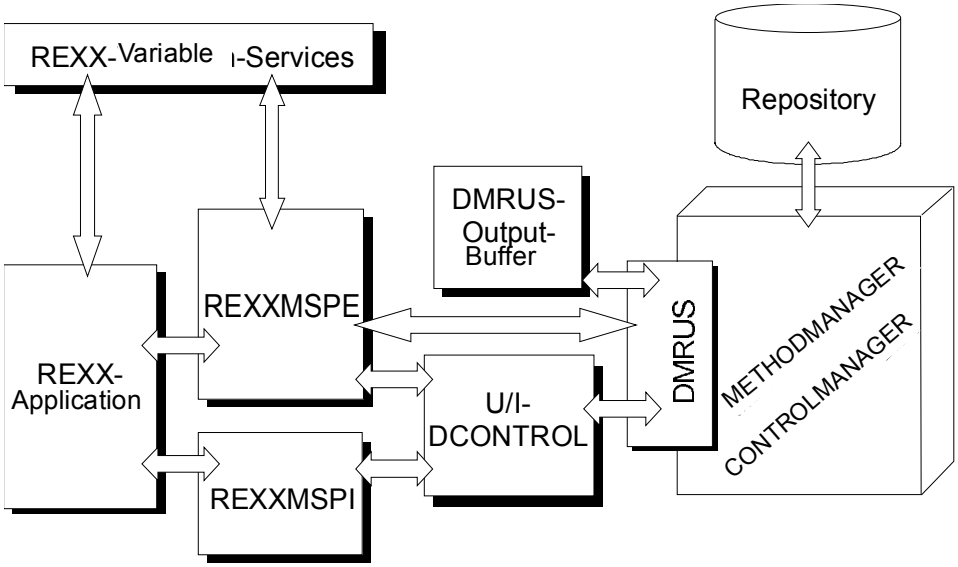
The purpose of the REXX interface is to extract and process Manager Products repository data from within a REXX application. You can easily change the information in the repository. To keep the data interface simple, the information is passed in the REXX environment. That is, information is exchanged in the Manager Products repository using the variables of the REXX application.

You can execute MethodManager and ControlManager functions from REXX in the MVS operating system. You can also use the REXX interface, MethodManager, and ControlManager in batch mode as long as no other part of the REXX application requires an ISPF environment.

The delivery tape includes the REXX interface and is identified by the FU-code RX01. You will need to customize MP.CORP to use the REXX interface DACCESS or SWAP function. You perform the customization by modifying and constructing the EXECUTIVE routines in the file MP.CORP. The other functions do not need customization.

2 Overview of the REXX Interface

This diagram demonstrates the relations between the various components of the REXX interface:



The DCONTROL area is created during the initialization of the application. This means, in ISPF, two split-screen applications can each work with their own area.

Initialize the interface *once* in the application that uses it. This is usually done during the early stages of the application.

Figure 1 • Initializing the interface

```
/* REXX ------- the application ------- */
  ...
         set some variables for initialization
  ...

  CALL REXXMSPI
    IF  MPIORC > 0  THEN  DO
    SAY '### Error in REXX-MSP interface, RC='MPIORC
    EXIT
  END
  ...
```

The initialization is usually done in the REXX procedure that contains the start of the application as the main procedure. When this procedure ends, the connection to MethodManager and ControlManager (herein called MMR/CMR) closes.

Interface Modes

The invocation of REXXMSPI is responsible for the internal initialization of the REXX interface. After the call completes successfully, the interface is in one of these modes:

- Temporary
- Permanent
- Resident
- Server

TEMPORARY Mode

The TEMPORARY mode reserves a temporary area for the REXX application. This area is released automatically when the application terminates.

An MMR/CMR session started via the LOGON command terminates automatically.

In an ISPF split screen environment, two REXX applications initialized in TEMPORARY mode have separate MMR/CMR sessions with their own data areas.

PERMANENT Mode

The PERMANENT mode calls REXXMSPI to activate a subtask that reserves up to 8 areas for an equal number of MMR/CMR sessions. Under ISPF, with a split screen, up to two subtasks of this type can be active in their own area.

Usually the parent-task in the REXX application terminates the subtask using a SHUTDOWN call. If this does not happen, MVS terminates the subtasks IKJMSPCT and IKJMSPWD with abend A03.

A normal panel dialog can run under ISPF between the start and the end of the subtask. This dialog enables an interactive use of the areas permanently available in the background (and their MMR/CMR sessions). In order to uncouple parallel activities, there is usually one REXX area assigned to each ISPF split screen application.

RESIDENT Mode

The RESIDENT mode reserves areas in the same way as for the PERMANENT mode, but differs in that the subtask is only activated once in the address space. Each additional invocation of REXXMSPI—independent of the ISPF split screen application that executes the call—uses this task and its areas.

This mode allows you to route both sides of the ISPF split screen into one area. In this case, it is the responsibility of the applications to synchronize the corresponding command execution and the usage of the output.

SERVER Mode

The SERVER mode is only available if ASG-Manager Products Server Facility (herein called MPSF) is active. Using this mode, a client REXX application can establish a persistent connection with MPSF and execute all Manager Products (MPR) commands via MPSF. You can activate persistency using a special terminating function in REXXMSPI.

The SERVER mode should not be confused with the RESIDENT and PERMANENT modes kept for compatibility. These modes allow the creation of persistent sessions within an ISPF address-space which keeps ISPF/TSO, the REXX-i/f tasks, the client REXX application, and the complete MPR software in that address-space.

Note:

RESIDENT and PERMANENT modes cannot be used to establish SERVER connections.

Selecting Modes

REXXMSPI evaluates the value of the variable MPIOMPFCT. The value of this variable determines one of these modes:

Mode	Value
TEMPORARY	This is the standard setting.
PERMANENT	In this mode, each ISPF screen application has its own fixed areas.
RESIDENT	In ISPF, all applications have a common pool of areas.
SERVER	The REXX application communicates its commands to a dedicated MPSF execution task.
RESUME	Under ISPF, a search for a PERMANENT and/or RESIDENT area, which may already exist, is requested. If such an area is not available, the TEMPORARY mode is activated. When a SERVER mode connection is established and the previous REXX client task terminates with the special termination call, use RESUME to continue using the MPSF session. Persistent sessions have to watch for time-outs of their MPSF tasks.
SHUTDOWN	A PERMANENT/RESIDENT/SERVER mode previously initialized by this application is terminated.

In batch mode (not ISPF in batch mode), the modes PERSISTENT or RESIDENT are equal to TEMPORARY as there is no REXX-i/f persistency mode available outside ISPF online REXX applications.

Always execute a LOGOFF and subsequent SHUTDOWN for PERMANENT and/or RESIDENT modes to assure a smooth transition. After SHUTDOWN, the interface can no longer be used from within the application.

Initializing with REXXMSPI

The initialization via REXXMSPI completes these tasks:

- Provides the central U/I DCONTROL area and additional input/output buffers.
- Activates the REXX language environment (by using ATTACH) in order to make the sub-function MPIO available. Use this function to execute all subsequent MMR/CMR functions until the last call, which is usually LOGOFF.

After the initialization completes, all function calls are executed via this new REXX command:

```
MPIO ... additional operands ...
```

REXX executes this function dynamically via a call to REXXMSPE.

The communication between the REXX application, the interface, and MMR/CMR is implemented via the REXX variable services. When specifying an MPIO function, REXX variables are either filled with input values or they are filled on return with output values.

The variables currently defined for the communication when executing commands are described in Chapter 3, "Differences for ISPF (RESIDENT and PERMANENT Modes)," on page 13.

Neither TSO nor ISPF is required to run the REXX interface. This enables you to use it in batch mode.

Figure 2 • Example job

```
//REXXBTCH EXEC PGM=IRXJCL,
//          PARM='REXXAPP parms'
//STEPLIB     DD   DISP=SHR,DSN=MP.LOADLIB
//SYSEXEC     DD   DISP=SHR,DSN=MP.REXX.APPS
//SYSTSPRT    DD   SYSOUT=*
//* -- for using METHODMANAGER
//MPAID       DD   DISP=SHR,DSN=Your.Mpaid
//...
//repos       DD   DISP=SHR,DSN=Your.Data.Repository.INDEX
//...
//MPIN        DD   DUMMY
//MPOUT       DD   SYSOUT=*
//MPRDIAG     DD   ...
```

Several variables are defined for controlling the REXXMSPI settings. Usually, the defaults defined within by the interface routines are sufficient.

For example, these commands activate the REXX interface for a specific REXX application:

```
MPIOMPFCT = 'mode'
...
CALL REXXMSPI
...
```

With the REXX interface you can create up to eight DCONTROL areas for the RESIDENT or PERMANENT modes. These areas are parallel within a session and all other modes are exactly one session per connection. Under ISPF split-screen, every ISPF REXX application has its own session or uses one common session depending on the chosen modes.

The first call to REXXMSPI initializes all U/I DCONTROL areas. The default setting for the number of areas is 1, but can be set to a higher value (for RESIDENT or PERMANENT) by setting the variable MPIOAREAS:

```
MPIOMPFCT     = 'RESIDENT'  or 'PERMANENT'
MPIOAREAS     = n
...
MPIOAREALINES = lines
MPIOOUTPAGES  = out-blocks
...
CALL REXXMSPI
...
```

where:

n is the number of areas. The maximum defined is 8, a value smaller than 1 is treated as an error.

`MPIOAREALINES` is an optional variable that defines the maximum number of command lines handled via these areas. Each line here has a maximum length of 255 characters. The default value is 250. This variable can be used in all modes.

`MPIOOUTPAGES` specifies the maximum size of the output area in 4K blocks. If for a standard query for members (e.g., LIST ONLY *xxx*) the output lines have a length of 72 characters, you can keep about 1137 lines using this variable:

```
MPIOAREALINES = 20
```

The default value is 8. For example, 8 x 4096 = 32768 bytes, about 430 lines.

This variable can also be used in any mode.

Each area managed by IKJMSPCT has its own unique ID and can only be accessed by subsequent REXX procedures when this ID is specified. The variable MPIOMPID sets the ID during the initialization:

```
MPIOMPFCT = 'mode'
...
MPIOMPID  = 'cccccccc0001'
...
CALL REXXMSPI
...
```

where *cccccccc* is the first 8 characters specified by the user as an arbitrary string (blanks are permitted, but can be confusing). The last four digits have to be specified in a predefined way, resulting in a string of 12 characters. This ID is assigned verbatim to the first DCONTROL, and then IKJMSPCT automatically continues by assigning to the second the value *cccccccc*0002 and so on.

Activating SERVER Mode

The SERVER mode is activated by using several special variables:

```
MPIOMPFCT = 'SERVER'
MPIOMPSRV = 'mpsf-luname'
MPIOMPHND = 0
...
CALL REXXMSPI
...
```

where *mpsf-luname* is a unique MPSF APPC name. Any MPSF task can be connected as SERVER. Connections unavailable to MPSF tasks produce an error from REXXMSPI. See the *ASG-Manager Products Server Facility User's Guide* for more information.

MPSF returns a *handle* (a numeric value) in the variable MPIOMPHND that uniquely identifies the active session. It is the task of the REXX application to keep this handle for a following REXX application to RESUME and re-establish the persistent connection

See Chapter 4, "Persistent Sessions (SERVER Mode)," on page 17 for additional information about SERVER mode sessions.

To resume a persistent session in another REXX-application

- Use this call:

```
MPIOMPFCT = 'RESUME'
MPIOMPSRV = 'mpsf-luname'
MPIOMPHND = previous-session-handle
...
CALL REXXMSPI
...
```

where:

mpsf-luname is the name which has been assigned as a unique MPSF APPC name.

MPIOMPHND contains a previously returned session handle. If the session has been terminated (time-out, MPSF shutdown, etc.), an error returns.

Executing LOGON with the Initialization Call

In all initializing modes it is possible to execute a LOGON together with the initialization call. The LOGON to ControlManager has to be handed over as first command after the initialization call.

For each area, you can specify a logon sequence with a maximum of 9 lines:

```
MPIOMPFCT     = 'mode'
...
MPIOLOGON.i.0 = m
MPIOLOGON.i.1 = 'first command line for the i-th area'
 ...
MPIOLOGON.i.m = '  last line for the i-th area'
...
CALL REXXMSPI
...
```

The first line MPIOLOGON.*i*.1 is usually the LOGON command, while the subsequent lines activate specific dictionaries.

If MPIOAREAS = 1 (this is the default), pass the start up LOGON by entering:

```
MPIOMPFCT    = 'mode'
...
MPIOLOGON.0  = m
MPIOLOGON.1  = 'first command line'
 ...
MPIOLOGON.m  = '  last line'
...
CALL REXXMSPI
...
```

If for one area the variable MPIOLOGON.0 = 0 or MPIOLOGON.i.0 = 0, no LOGON executes for this area. The first normal command may trigger an AUTOLOGON.

The call to REXXMSPI returns with some variables set to reflect the success or failure of the initialization. The main return code of the REXX interface is kept in the variable:

```
MPIOMPFCT     = 'mode'
...
CALL REXXMSPI
SAY '---- return from REXX i/f initialization, RC=' MPIORC
```

The variable MPIORC contains the return-code of the REXX interface for this call and for all subsequent calls. If the interface itself detects an erroneous condition, it gives a return code as stated in Appendix C, "Return Codes" on page 49. This table provides an overview:

Error Code	Meaning
MPIORC = 0	Execution of this request was successful and without any error.
MPIORC = 4...16	See meaning below.
MPIORC >= 32	A very severe error occurred during processing of the request in the REXX interface.

The return codes 4...16 are not return-codes of the REXX interface but are mapped into the variable MPIORC when these conditions are met:

- The REXX interface checks all parameters of the request and finds no error
- The command to execute was handed over via the interface to ControlManager (CMR) and executed there
- The result was received from CMR and included a CMR/DMR return code between 4 through 16

In this case, MPIORC contains the return code value from CMR.

After the successful initialization, the execution of MPR commands is done by a special REXX interface execution syntax:

```
MPIOMPFCT     = 'mode'
...
CALL REXXMSPI
...
...
MPIOCMD.0 = n  /* number of command lines to execute */
MPIOCMD.1 = 'first command line'
...
MPIOCMD.n = 'last command line'
...
ADDRESS ATTACH "MPIO COMMAND"
SAY '---- return from REXX i/f command execution, RC=' MPIORC
```

More forms of this MPIO function are given in Chapter 5, "Syntax of the MPIO Call," on page 21.

3 Differences for ISPF (RESIDENT and PERMANENT Modes)

In ISPF split screen mode you can use the variant MPIOMPFCT= 'RESIDENT' with this call sequence:

- Screen-1—REXX procedure after CALL REXXMSPI with 'RESIDENT'
- Screen-2—Calls REXXMSPI with 'RESIDENT' as well and uses from this point on the areas of screen-1
- Screen-1—Calls a procedure, which in turn calls other ISPF functions via the MPR command ISPF
- Screen-2—Calls procedures that use ISPF functions as well

This input is required:

```
Screen 1   MPIOMPFCT   = 'PERMANENT'
           MPIOMPID    = 'iiiiiiii0001'
           MPIOAREAS   = 1
           MPIOLOGON.0 = m
           ...
           CALL REXXMSPI

Screen 2   MPIOMPFCT   = 'PERMANENT'
           MPIOMPID    = 'kkkkkkkk0001'    different from above
           MPIOAREAS   = 1
           MPIOLOGON.0 = n
           ...
           CALL REXXMSPI
```

Both screens now operate separate areas and do not interfere with one another.

Both screens must also make a SHUTDOWN (see "Using the SHUTDOWN Command" on page 14) and it is necessary to ensure the setting of the LOCK-ID in MPIOMPID is correct in all call REXX procedures.

After calling REXXMSPI, these variables are available besides MPIORC = 0 (read only):

Variable	Value	Description
MPIOCTIS	RES	IKJMSPCT controls all areas RESIDENT for all users.
	TMP	The area has been created only temporarily for the calling REXX procedure and is removed when the procedure ends.
MPIOCTLOGON	NO	No LOGON executed yet.
	YES	A first command has been executed successfully. It is assumed that a valid session (with AUTOLOGON) is active.

Enter this to reuse an area initialized as resident by other REXX procedures:

```
MPIOMPFCT = 'RESUME'
MPIOMPID  = 'cccccccc000n'
CALL REXXMSPI
IF MPIORC > 0  then ...
...
```

where:

cccccccc is the 8-character ID of the area to be accessed, as it was specified during initialization.

000*n* is the internal position of the area '0001' up to a maximum of '0008'.

Using the SHUTDOWN Command

The resident areas continue to exist until either the task that has executed the initialization call REXXMSPI (with 'RESIDENT' or 'PERMANENT') terminates or until you enter the SHUTDOWN command to have the REXX procedure explicitly terminate:

```
MPIOMPFCT = 'SHUTDOWN'
CALL REXXMSPI
```

Starting REXX Procdures

All REXX procedures may start with this code if the initialization is executed under ISPF:

```
MPIOMPFCT = 'RESIDENT'  or 'PERMANENT'
MPIOMPID  = 'cccccccc0001'
...
CALL REXXMSPI
...
```

The procedure executed first also carries out the initialization. All of the subsequent procedures work internally with MPIOMPFCT= 'RESUME'.

4 Persistent Sessions (SERVER Mode)

You can create *persistent* sessions using the mode MPIOMPFCT= 'SERVER'. These sessions allow the client applications to terminate without the MPSF task terminating. The MPSF task waits for another client application that resumes that pending session. The persistent SERVER mode is equivalent to the old PERMANENT and RESIDENT modes except it is not restricted to ISPF only, and MPSF controls the session.

Creating Persistent Sessions

MPSF creates persistent sessions and assigns them a unique *handle* (numerical value) for this session. The creating application has to request this handle during its initialization call:

```
MPIOMPFCT = 'SERVER'
MPIOMPSRV = 'mpsf-luname'
MPIOMPHND = 0
...
CALL REXXMSPI
...
IF  MPIORC = 0  THEN  DO
  SAY "--- session handle is now = " MPIOMPHND
  ...
END
...
```

You must set the variable MPIOMPHND to 0 for this call. If the session was established, the REXX interface returns the new handle value in this variable. You can also keep this handle for applications that follow. For ISPF you may enter:

```
...
MPIOHND = MPIOMPHND
"ISPEXEC VPUT (MPIOHND) PROFILE"
...
```

An application that does not want to establish a persistent session, may simply ignore the value of MPIOMPHND as it returns. However, every application that wants to start a new session with MPSF has to use the above given command sequence.

Terminating and Resuming Applications

If an application wants to terminate but wants to allow subsequent applications to resume this session, it must terminate with this command:

```
...
MPIOSCOD = ''
ADDRESS ATTACH "MPIO COMMAND '$$DEALLOCATE$$'"
MPIOMPFCT = 'SHUTDOWN'
CALL REXXMSPI
...
```

If this code executes correctly, the next application resumes with that pending session (until its time-out terminates it otherwise).

Use the initialization call to resume a pending session:

```
...
  /* try to find a session handle to resume ............  */
  /* for ISPF:  */
 "ISPEXEC VGET (MPIOHND) PROFILE"
...
MPIOMPFCT = 'RESUME'
MPIOMPSRV = 'mpsf-luname'
MPIOMPHND = previous-session-handle
...
CALL REXXMSPI
...
IF  MPIORC = 0  THEN  DO
  SAY "--- session handle reused = " MPIOMPHND
  ...
END
...
```

After this call to REXXMSPI, the value of MPIOMPHND gives the handle and it should be the same value as the *previous-session-handle*. If not, MPSF creates a new session (because the previous one did not exist anymore).

Enter this call if you want the application to terminate a (non-)persistent session and the MPSF task:

```
...
MPIOSCOD = ''
ADDRESS ATTACH "MPIO COMMAND 'LOGOFF.'"
MPIOMPFCT = 'SHUTDOWN'
CALL REXXMSPI
...
  /* plus for example for ISPF: */
  MPIOHND = 0
  "ISPEXEC VPUT (MPIOHND) PROFILE"
...
```

The LOGOFF command terminates the MPSF session, freeing all its resources. The final SHUTDOWN clears all REXX interface resources.

5 Syntax of the MPIO Call

The MPIO statement design is similar to that used by the REXX function MVS/EXECIO:

```
MPIO  sub-function   [Operands according to function]
```

where *sub-function* consists of these commands:

```
{ COMMAND  }   [ { STEM  MPIOCMD.          ... } ]
{ EXECUTE  }     { "a one-line-command"      }
{ EXECUTD  }
```

These are the basic functions of the REXX interface for executing commands:

COMMAND. The command (or the command sequence) is passed to MMR/CMR, processed, and the output is immediately made available to the REXX application in the variable MPIOOUT.

The next call executes a new command.

EXECUTE. As with COMMAND, the command is handed over and executed. In contrast with COMMAND, the output of MMR/CMR remains in the internal interface buffers. Only the MPIO*xxx* variables, which specify the number of available output lines, are filled.

Subsequent OUTMAP calls can retrieve this data.

EXECUTD. This command is only available in PERMANENT or RESIDENT mode.

Here, the commands are passed to MMR/CMR as well, but contrary to EXECUTE, control is immediately returned to the REXX application.

Subsequent WAIT calls are necessary to determine the end of the asynchronous execution. Until the execution has ended, no additional commands can be executed via the busy subtask IKJMSPCT (in PERMANENT mode, several tasks can be available).

OUTMAP. Special mapping for MPIOOUT onto variables.

WAIT. Waits for the EXECUTD commands to complete.

CONTROL. DCONTROL control parameter via DOPTION.1 .. 9 and DBUFFLEN (default = 8192 Bytes)

Without operands:

The variables MPIODOPTION.1 to MPIODOPTION.9 and MPIODBUFFLEN are used.

Operand = *cccccccccc*: The parameter is stored in OPTION1..9, up to the maximum of 9 characters.

Operand = RESET: The default values are reset.

DGET. Reads GLOBAL variables from MethodManager and making them available in REXX variables.

DPUT. Writes REXX variables into GLOBAL variables in MMR/CMR.

DACCESS. Reads (all) member attributes.

SWAP. Switches to interactive mode in MethodManager. This function is only available to MethodManager customers.

6 REXX Interface Variables

All variables available in the REXX interface start with the prefix MPIO.

After a successful call of REXXMSPI for initialization, the variable MPIOVERSION contains the valid version of the REXX interface. This information, among other things, is used for troubleshooting. If necessary, additional internal debugging variables are available. The included examples contain several small routines that can log this additional data if needed. In your applications, this group of variables is not needed.

Passing a command stream via the interface to the U/I, and then to MMR/CMR, is executed similar to MVS-EXECIO via a STEM variable with the standard name MPIOCMD.

Alternatively, you can use the operand STEM *xxx*. to specify a different name for handing over the commands. You can use these standard names:

Variable	Description
MPIOCMD.0.	Numeric, specifies the number of following elements, the maximum depends on the setting of MPIOAREALINES.
MPIOCMD.1.	The command(s) as string lines with a maximum of 80 characters.
MPIOCMD.*n*	

After verifying the parameters, the contents of MPIOCMD.1 ... MPIOCMD.*n* is handed over to MMR/CMR. If the parameters are incorrect, the variable MPIORC is set to return code >0 and the function is immediately terminated.

Appendix C, "Return Codes," on page 49 lists all the return codes currently used by the REXX interface.

After calling MMR/CMR, the variables are set to these results:

Result	Description
MPIORC	Return code of the interface and MPR command status.

If the REXX interface does not execute any MPR command the MPIORCMPR variable has the default value -1.

If MPR commands are executed, the variable will contain the same value as MPIORC. The MPIORC variables can have these values:

MPIODRETURN	A 4-character string from DRETURN,,.DSEVRITY. See Appendix C, "Return Codes," on page 49.
MPIODOPT	A 9-character string with the OPTION1..9 values.
MPIOTOTL	Numeric value of DOUTTOTL.
MPIOTOTM	Numeric value of DOUTTOTM.
MPIOWAIT	Numeric value of DOUTWAIT.
MPIOOLNS	Numeric value of all output lines.

If the function COMMAND is used, all output lines are returned to the calling REXX procedure (according to the setting of the DOPTION flags). If the lines are processed only at a later stage, ASG recommends starting the command with EXECUTE. In this case, the output is kept in an internal output buffer and can be retrieved later with a subsequent OUTMAP call.

Note:

The next COMMAND or EXECUTE sequence overwrites this buffer with new data.

If the executed function has created output lines, they are retrieved and stored in the STEM variable MPIOOUT:

Variable	Description
MPIOOUT.0	Numeric, number of output lines (= value of MPIOOLNS)
MPIOOUT.1 MPIOOUT.*m*	The output lines with a maximum length of 256

Some commands (e.g., LOGOFF) do not create output (except the return codes).

7 The OUTMAP Function

After COMMAND or EXECUTE, you can read the output buffer as often as you want (until the next COMMAND/EXECUTE call). The output can be requested in three different forms:

- Direct by handing over the output lines with a maximum line length of 256, as they are stored in the output buffer:

  ```
  "MPIO  OUTMAP"
  ```

- Splitting the output into columns, which are defined by the specification of the start column and the column width:

  ```
  "MPIO  OUTMAP  FIELDS var-1.  start-1 length-1  [var-2 ..]"
  ```

  For example:

Col-1	**32**	**45**	**55**	**58**
DE-ITEM-1	ITEM	SCE ENC		OWNER
DE-ITEM-2	ITEM	...		...

- Splitting the lines into words and storing them in column variables:

  ```
  "MPIO  OUTMAP  WORDS  stem-var-1.  [  stem-var-2 ...]"
  ```

  For splitting the lines into words, it may be useful to ignore certain parts of the output lines. For this purpose, you can insert two special definitions between the variables receiving the individual values:

  ```
  ...  SKIP.nn  ...     and     ...  TO.ccc  ...
  ```

  where:

  nn specifies a number greater than zero of words to skip in each line.

  ccc specifies the start column of the word search. With TO.*ccc*, you can assign a column area multiple times.

For example:

```
"... WORDS  VAR1.  SKIP.1  VAR.2  TO.50  VAR3. .. "
```

results in:

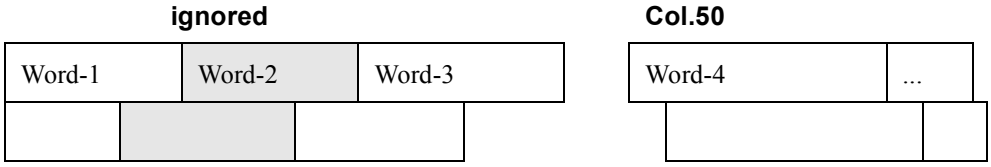


Figure 3 • Initialization example

```
ADDRESS ATTACH

"MPIO COMMAND 'LOGON ADMIN PASSWORD ADMIN NO-PROFILE;'
SAY MPIORC

MPIOCMD.0 = 2
MPIOCMD.1 = 'DICTIONARY DEMO;'
MPIOCMD.2 = 'AUTHORITY  DEMO;'
"MPIO COMMAND MPIOCMD."

MPIOCMD.0 = 1
MPIOCMD.1 = 'LIST ONLY DE-HAW;'
"MPIO COMMAND MPIOCMD."
DO  i = 1  TO  MPIOOUT.0
SAY RIGHT(i,3) MPIOOUT.i
END

"MPIO COMMAND 'LOGOFF'"
ADDRESS MVS
```

After the initialization—which has been omitted here—these commands are passed to the REXX interface:

Command	Description
MPIO COMMAND "LOGON ...	The first command to be executed is a LOGON to start the MMR/CMR session.
MPIOCMD.*i* = *t*... and	The variable MPIOCMD. is used to pass two commands as block.
MPIO COMMAND MPIOCMD.	
MPIOCMD.1 = ‘LIST ONLY ...	The first actual command.
MPIO ... LOGOFF	The end for now.

Figure 4 • Using external files

```
        ADDRESS TSO "ALLOC FI(MPIN) DA('a-file')"
        cmds.0 = 4
        cmds.1 = 'REPLACE member-xy;'
        cmds.2 = 'ITEM'
        cmds.3 = '   HELD-AS CHAR 30'
        cmds.4 = ';'
        "EXECIO * DISKW MPIN (STEM cmds. FINIS"
         ...

         ADDRESS ATTACH
              "MPIO CONTROL '01'"
      ...     "MPIO COMMAND"
   .....      "MPIO CONTROL '00'"
        ...
```

The MPIO-CONTROL redirects the input of the REXX interface from MPIOCMD. to the external file that has been allocated under the DD name MPIN. The subsequent call without the specification of additional operands after COMMAND automatically switches to MPIN and returns the output via MPIOOUT.

Before executing the next normal command, you should switch back with an ADDRESS ATTACH "MPIO CONTROL '00' ".

For an example, see the sub-procedure MPRY in REXXCMD.

8 Variable Services for the Communication with MMR/CMR

A simple variable interface for setting and reading DYR variables is available with the REXX Version 2.2.0 interface.

This sub-function requires the existence of the procedures EX-REXX-***, which should be available in the MPAID with the name MPREXD****. Customizable routines are delivered on the release tape in file MP.CORP.

Only GLOBAL variables can be accessed, since the procedures used here have no memory for other pools.

DYR GLOBAL variables are read with:

```
ADDRESS ATTACH "MPIO DGET rexx-var msp-var"
```

where:

rexx-var is the STEM name of the REXX variables that should receive the value(s).

msp-var is the name of the MSP variable, which may also be an ARRAY.

On execution, these cases are distinguished:

- The variable (MPIORC > 0) is not defined.
- The variable contains only a single value.

  The REXX variable with the name rexx-var contains the value.
- The variable consists of an array (ARRAYHI > 1).

  The REXX variable rexx-var.0 contains the number of array elements, the REXX variables rexx-var.1 ... rexx-var.n correspond to the values in the DYR array. If the array is not filled completely, the corresponding elements are missing in REXX.

Use this command to set the PROCL GLOBAL variables:

```
ADDRESS ATTACH "MPIO DPUT rexx-var msp-var"
```

where:

rexx-var is the STEM name of the REXX variables that contain the value(s).

msp-var is the name of the DYR variable of type GLOBAL, which should receive the values from REXX.

If a STEM is specified as a REXX variable (XYZ. ..., e.g.), the complete array with XYZ.0 elements is stored in a real array.

A special variant of DPUT is

```
ADDRESS ATTACH "MPIO DPUT rexx-var msp-member MEMBER"
```

where:

rexx-var is the name STEM of the REXX variable that contains the complete source of a member definition.

msp-member (if there is an active repository for the application at that time) is the name of the contents encoded in the current repository under the member name .

9 DACCESS from Within the REXX Application

The DACCESS function largely corresponds to the function used in the Manager Products procedure language. Use the following to call it from REXX:

```
ADDRESS ATTACH "MPIO DACCESS msp-member  [ options ]"
```

where *msp-member* is the name of a member in the current repository.

Internally, the procedure EX-REXX-QMTS is called (in the MPAID MPREXDQMTS), which branches to a member type specific sub-procedure after a DACCESS to the corresponding member, where all (or at least some) attributes of the member are processed and passed to the REXX interface.

Since member types usually can have an arbitrary number of attributes, a fixed access function for attributes is not useful. For this reason, the user has to execute a generation for the required member types before DACCESS is used for the first time from within REXX. This generation is executed in a MethodManager session with:

```
REXX-ACGEN  member-type  [ xxxx ] [ options ] ;
```

where:

member-type is the complete ENCODE-KEYWORD of the desired member type. It is indicated for DACCESS for a member of this type in the variable MEMBER_TYPE.

xxxx is a unique name with a maximum of 4 characters, which identifies the generated MethodManager DYR procedure EX-REXX-QMTS-*xxxx* (in the MPAID: MPREXD*xxxx*). If another generation is executed after changes in the member type, this short name may be omitted.

Examples:

```
REXX-ACGEN  EXECUTIVE-ROUTINE  EXEC ;
REXX-ACGEN  DV-FELD  ITEM ;
```

These items are required to execute REXX-ACGEN:

- The execution takes place in the repository with the UDS that contains the desired member types
- The member type EXECUTIVE-ROUTINE is available in this repository

The routines EX-REXX-QMTS and EX-REXX-QMTS-*encode-keyword* generated by REXX-ACGEN are only available for the REXX interface. The data interface used by these routines is not available for customer uses.

The result of a REXX call of MPIO DACCESS *msp-member* is a set of REXX variables, which contain with the same name as their counterparts in the MPR procedure language, the corresponding attribute values (e.g., after DACCESS EX-REXX-QUVR):

```
ACCESS_MEMBER    "EX-REXX-QUVR"
MEMBER_CONDITION "ENCODED"
MEMBER_TYPE  "EXECUTIVE-ROUTINE"
...
EXECUTIVE_LEVEL  "unknown"
COUNT_CONTENTS   41
```

With regard to the last variable from the example above, consider these items:

- If there is a generated access routine for the member type of the member *msp-member*, DACCESS passes the 2 variables ACCESS_MEMBER and MEMBER_CONDITION filled with *msp-member* and ENCODED, respectively.
- DACCESS without further operands only makes the base attributes of the member type available.
- The command DACCESS *msp-member* WITH COUNTS creates the variables COUNT_xyz, which contain the number of available attribute values for each repeated attribute.
- With DACCESS *msp-member* COUNTS, only the COUNT variables are created.
- With DACCESS *msp-member* WITH rep-attr (further attr), you can access these attributes themselves. Their values can be found in the corresponding STEM variables with the number in rexx-var.0.

  An access to a *rep-attr* can fill several variables if the corresponding clause has several sub-values.
- With DACCESS *msp-member* ONLY attribute, only the requested attribute is handed over.
- DACCESS *msp-member* ALL returns all attributes of the member (that are filled).
- With DACCESS *msp-member* SOURCE, the complete source is returned in the REXX STEM variable SOURCE. This also works for members that are not encoded.

The REXX variables largely correspond to the names that can be checked with:

```
SHOW MEMBER-TYPE  VARIABLES  FOR  MEMBER-TYPE  member-type ;
```

Here are some remarks regarding the host command REXX-ACGEN:

Each MPIO DACCESS *member* call of the REXX interface corresponds on the host side to the execution of the procedure MPREXDQMTS in this form:

```
MPREXDQMTS  member  [ options ] ;
```

The procedure MPREXDQMTS executes these sub-functions:

- Executing a DACCESS MEMBER member WITH SITUATION.

  If the return code of this function is not 0, DUMMY, SOURCE-ONLY, and non-existing members receive special treatment.

- For the current repository (determined via the variable &DICT), the member type of the member (in the DACCESS variable MEMBER_TYPE) is mapped to a generated access routine MPREXD*xxxx*.

  If no routine is found, the routine MPREXDQUDF is called, which notifies the REXX interface of this error condition. This usually returns to the application with MPIORC = 56/57.

- The localized access routine MPREXD*xxxx* is called. It generates the transfer data for the desired attributes for the member type. The REXX interface receives this data.

- The interface generates from the transfer data, the filled REXX variables and additional control information, which can be retrieved by the application.

Construct the procedure EX-REXX-QMTS-PROTOTYP during the installation of the REXX interface. It generates in the MPAID first, empty frame procedure MPREXDQMTS, which does not yet know a user repository and the member type.

With each subsequent REXX-ACGEN call:

- A member type specific routine EX-REXX-QMTS-*encode-keyword* is saved in the corresponding repository (for this reason, each repository has to contain the member type EXECUTIVE-ROUTINE). This routine is stored in the MPAID with the name MPREXD*xxxx*, where *xxxx* is a suffix with a maximum of 4 characters that can be chosen freely by the customer, as long as it does not correspond to one of the names used by the REXX interface (e.g., QMTS itself, QUVR, STVR, etc.).
- The routine EX-REXX-QMTS is completed via the current version MPREXDQMTS in the MPAID and constructed again. At the same time, queries to the repository and the member type are inserted automatically.

Caution! When generating existing routines again: MPREXDQMTS and all access routines are retained.

If a new generation is required, the procedure EX-REXX-QMTS should not be modified manually. A CONSTRUCT should be done for the member EX-REXX-QMTS-PROTOTYP from the ADMIN repository, with subsequent REXX-ACGEN calls for all required member types.

Figure 5 • Example for the automation of the generation

```
MPXX LITERAL=^
PARSE ARG hierarchy data-uds data-dict
     data-auth data-status

DICT ADMIN;
AUTH hihi;
STA  data-uds;

ARRAYGEN ^list 'WHICH MEMBER-TYPES CONST UH-hierarchy'^;

COMMAND MBRS MBRK

cnt = 0
DO over Member-type-member in "list"

  abbrev = SUBSTR(member-name,1,4)
  IF in METHODMANAGER THEN DO
    DACCESS MEMBER member-name;
    DRETRIEVE FIRST ALIAS;
    abbrev = ALIAS_NAME(1)
    DRELEASE MEMBER member-name;
  END

  cnt = cnt+1
  MBRS(cnt) = member-name
  MBRK(cnt) = abbrev

END

DICT data-dict;
AUTH data-auth;
STA  data-status;
```

```
DO FOR ARRAYHI(^MBRS^)
  m = FDO(^DFOR^)

  CEXEC REXX-ACGEN MBRS(m) MBRK(m)
          [ options ];

END
```

The procedure REXX-ACGEN allows some customization via additional parameters:

Parameter	Description
NOUDR	User defined relationships are ignored.
UDR	Additional variables are generated for the UDRS and SUDRS. UDR and NOUDR exclude one another.
ARRAY	All attribute variables generated by the REXX interface are transferred with: var.0 = Number of variable values var.1 ... The values of the attribute var.*n* Without this option, the elementary attributes are stored without "var.0" and for single values only with var = value.
COUNTS	The variables COUNT_var are generated. Without this option, these variables are only created for a call DACCESS member WITH COUNTS. COUNT and ARRAY exclude one another.
REF	All references departing from the member are created as well. In this case variables are: COUNT_REFERENCES REF_NAME.*x* Name of member REF_RELATIONSHIP.*x* Relationship keyword REF_MEMBER_TYPE.*x* Member type

The settings described above can also be made in the Exit procedure EX-REXX-ACGEN-USEREXIT.

In this user exit, the variable SITCATR has been defined. The variable names defined there are created as REXX variables after the internal DACCESS MEMBER *xxxx* WITH SITUATION. By default, BASE_MEMBER_TYPE, MEMBER_CONDITION, and MEMBER_ER_INTEGRITY are defined.

An additional exit, EX-REXX-ACGEN-GENEXIT is available. Use this exit to complete MPREXDQMTS or MPREXD*xxxx* in the source. This exit is activated if the command uses the parameter GENEXIT*xxxx*, where *xxxx* is identical with the MPAID name in the procedure EX-REXX-ACGEN-GENEXIT (default = MPREXD$GNX). The exit is called immediately after the creation of the encode keyword EXECUTIVE. It can then create additional clauses. After it has returned, the MPAID-NAME clause and the CONTENTS clause are generated.

Among the attributes created for each member type, the COMMON-ATTRIBUTES play a special role, since they are created or ignored in REXX-ACGEN via a filter function. Internally, a SHOW MEM VAR FOR COMMON is executed for the generation process, with subsequent filtering to omit undesired attributes.

The filter consists of a procedure variable DROPATTR in the procedure EX-REXX-ACGEN-USEREXIT that suppresses by default the attributes QUERY, FREQUENCY, SECURITY-CLASSIFICATION, EFFECTIVE-DATE, OBSOLETE-DATE, and ACCESS-AUTHORITY.

If you make changes to the procedure, you should construct it again.

In general, the names of the variables provided by the REXX interface to the application correspond to the attributes, with these exceptions:

- Attributes having a dash in their name create REXX variables with the underscore character (_) instead of the dash (-).
- Repeated attributes that have the suffix *var*-NAME in the SHOW MEM VAR ... are created as REXX variables without this suffix. Other suffixes automatically suppressed are -ARE, -IS, and -TO.
- The CATALOGUE attribute is generated as the REXX variable CATALOG.
- The sub-attribute SEE-QUALIFICATION becomes the REXX variable SEE_FOR.
- Sub-attributes of attributes are mapped as STEM variables with the corresponding qualifier name parts in REXX. The rules applied for this mapping have been defined for the MPR base member types and, due to their complexity, will not be explained here. If you need more information, please contact the ASG Service Desk.

In the process of correcting errors in this area, these naming rules might lead to new forms of REXX variables for complex member types . To reduce the complexity in customer applications, ASG recommends using the REXX interface additional STEM variable MPIODACCESS.*xx*. This variable is provided after DACCESS.

MPIODACCESS.*xx* makes the attribute name structure more comprehensible. This table defines the variable options:

Variable	Description
MPIODACCESS.0	Number of all variables returned by DACCESS = number of attributes.
MPIODACCESS.*i*	Name of the REXX variable assigned to the corresponding attribute (e.g., ACCESS_MEMBER, NOTE, SEE).
MPIODACCESS.*i*.TYP	character = C character value, N numerical value, or U the value is missing (optional) character = * values are in an array
MPIODACCESS.*i*.CNT	Number of values if the attribute is type C or N.

Figure 6 is an example for accessing complex attributes for DACCESS.

Figure 6 • The member GR-BOOK

```
GROUP
CATALOG
          'MMR'
NOTE      'Customizing the panel interface'
SEE       GR-MMR-COMMON   FOR 'Original member'
CONTAINS
          MDG_TABLE_FIELD_CHAR
        , MDG_COMMAND_LINE_CHAR   IF  FE-1 EQ 'A'
    ELSE  MDG_BOOK                IF  FE-1 EQ 'B'
        , MDG_LINE_COMMAND_CHAR
      ...
        , MDG_STMAX               IF  FE-2 EQ '1'
    ELSE  MDG_BOOK_2              IF  FE-2 EQ '2'
      ...
        , MDG_MATRIX_SIZE_ONLINE
        , MDG_MATRIX_SIZE_BATCH
```

Returns for MPIO DACCESS GR-BOOK ALL:

```
ACCESS_MEMBER                          = GR-BOOK
MEMBER_TYPE                            = GROUP
CATALOG.0                              = 3
CATALOG.1                              = MMR
CATALOG.2                              = GR2
CATALOG.3                              = MR0
NOTE.0                                 = 1
NOTE.1                                 = Customizing ....
CONTAINS.0                             = 9
CONTAINS.1                             = MDG_TABLE_FIELD_CHAR
CONTAINS.1.INDEXED_BY.0                = 0
CONTAINS.1.CONDITION.0                 = 0
CONTAINS.1.ELSE.0                      = 0
CONTAINS.2                             = MDG_COMMAND_LINE_CHAR
CONTAINS.2.INDEXED_BY.0                = 0
CONTAINS.2.CONDITION.0                 = 1
CONTAINS.2.CONDITION.1                 = IF
CONTAINS.2.CONDITION.COMPARATOR        = FE-1
CONTAINS.2.CONDITION.OPERATOR          = EQ
CONTAINS.2.CONDITION.COMPARAND         = A
CONTAINS.2.ELSE.0                      = 1
CONTAINS.2.ELSE.1.CONTAINS.1           = MDG_BOOK
           ....
CONTAINS.8                             = MDG_MATRIX_SIZE_ONLINE
CONTAINS.8.INDEXED_BY.0                = 0
CONTAINS.8.CONDITION.0                 = 0
CONTAINS.8.ELSE.0                      = 0
CONTAINS.9                             = MDG_MATRIX_SIZE_BATCH
CONTAINS.9.INDEXED_BY.0                = 0
CONTAINS.9.CONDITION.0                 = 0
CONTAINS.9.ELSE.0                      = 0
```

To supplement DACCESS, the sub-function DEXPAND has been added:

```
ADDRESS ATTACH "MPIO DEXPAND member  [ options ]"
```

The available options are the DEXPAND operands documented in the *ASG-Manager Products Procedures Language*:

Operand	Option	Description
FOR	*language*	Language for the expanded form (must be available as a Functional Unit)
USE		
USING	*form*	HELD-AS etc.
VERSION	*vvv*	Version for items

Operand	Option	Description
GIVING	KNOWN-AS	Additional creation of the KNOWN-AS names
ALIAS		
WITH-ALIAS	*alias-type*	Use the ALIAS

The operands can be abbreviated.

After the call, the variables EXPANDED_MEMBER = *member* of the DEXPAND call and the important variables are available. You can use MPIODACCESS.*i* to get their names.

10 The SWAP Function

The SWAP function is another method of the REXX interface available for MethodManager customers:

```
MPIO   SWAP    [ profile-exec  | # ]
```

This function is used for switching to the interactive MethodManager mode from within a REXX application. This switching requires that

- The REXX application is running in an interactive environment (TSO, ISPF, or similar)
- Some additional preparations are made with regard to MethodManager

SWAP always switches to the current interactively displayable mode of the current MethodManager session. For example, the currently valid TSS or LCS panel displays and the dialog is continued from this point on as user dialog. The REXX application is from this point on deactivated.

The interactive execution—also indirectly via procedures—of the command MMR-SWAP ends this MethodManager mode and reactivates the sleeping REXX application. The application can continue the processing, but needs to take into account that the MethodManager environment may have changed during the interactive phase.

To minimize the side-effects of unintentional interactive activities, each entry via MPIO SWAP is executed via a profile routine, which is executed each time before activating the interactive mode. This routine has the predefined MPAID name MMRSWP0000.

This is an example routine supplied as EX-MMRSWP0000:

```
MPXX LITERAL=:
  GLOBAL MDG_MMR_SWAP
  IF  MDG_MMR_SWAP EQ :ON:               OR -
      MDG_MMR_SWAP EQ :AUPD:                -
  THEN  DO
      MPR :NOPR SET PF02 IMMED MMR-SWAP;:
      IF  &PVAL NE ::  THEN  DO
          MPR &PVAL ;
      END
  END
```

When creating this profile procedure, make sure that the global variable MDG_MMR_SWAP is defined by the REXX interface and expected within the Manager Products software at certain points with these predefined contents:

OFF	Normal mode, the REXX application executes commands
ON	Interactive mode called by REXX application with MPIO SWAP
AUPD	Interactive mode, automatic entry directly into AUPD (for usage, see below)
undefined	No processing from within REXX

MPIO SWAP always acts on these additional pieces of information:

Operand after SWAP	Can be the character #, in this case the standard profile MMRSWP0000 is executed. If a procedure name is specified instead of '#', this procedure is executed instead of MMRSWP0000. A SWAP without profile procedure is not possible.
Value in the REXX variable MPIOCMD.1	A regular MethodManager command with operands as far as they are specified.
MPIOCMD.0 and MPIOCMD.2 ... are ignored	This command is also passed to the profile routine, which usually after its initialization and test parts executes this command via the variable &PVAL.

Note:

The contents of MPIOCMD.1 are always evaluated as direct command. To avoid problems, assign an empty string to this variable before invoking MPIO SWAP, if only the switching is desired.

The supplied test procedure REXXCMD contains a small sub-procedure MPRZ that demonstrates the internal call of MPIO SWAP.

A typical case for switching via MPIO SWAP is if you want to execute the MethodManager function AUPD from within a REXX application. You can implement this as follows:

```
/* REXX --------------------- starting AUPD --------------- */
 ...
CALL REXXMSPI
 ...
MPIOCMD.1 = 'AUPD '!!member-name
ADDRESS ATTACH "MPIO SWAP #"
ADDRESS ISPEXEC "VGET (MMRAUPD) SHARED)"
 ...
```

For AUPD, a new additional EXIT has been introduced in MethodManager, which permits, together with the MethodManager variable MDG_MMR_SWAP and the ISPF variable MMRAUPD, extended control over the interactive mode via MPIO SWAP.

If AUPD is terminated, control is automatically returned to the REXX application, without the necessity to enter an MMR-SWAP manually.

If AUPD is terminated in a regular way with F3, three sub-cases have to be distinguished:

- The edited member has not been changed. In this case, the variable MMRAUPD contains on return the value *member-name* UNCHANGED
- The edited member has been saved and encoded. The variable MMRAUPD then contains the value *member-name* ENCODED
- If an error occurs during the encoding, the user receives the corresponding messages and can decide how to proceed with the AUPD

If AUPD is aborted with cancel, control is returned to the REXX application as well. The variable MMRAUPD then contains the value *member-name* CANCEL.

The storing of a value in the variable MMRAUPD occurs in the new exit EC9949. This exit is supplied together with the software as an example. In this case its only task is to assign a value to the ISPF variable MMRAUPD by means of the two call parameters and to save it as SHARED variable. If the REXX application does not need the information about how the AUPD was terminated, the exit is not required in this form.

Currently, the exit is called by the two command members MPEAC1100 and MPEAC1020, which handle the two cases mentioned above. At the same time it represents an AUPD-Clean-Up exit, which may also be useful in other cases.

If, from within an AUPD session activated with MPIO, SWAP activities are started by the user superseding the AUPD, or which bring the dialog into different MethodManager functions, an automatic return is no longer useful, since the dialog status may be undefined.

Appendix A
DCONTROL Area

These are the default settings for the DCONTROL area:

Note:
The shaded fields must not be changed.

DDMR	CM00	Not changeable
DBUFFLEN	8192	Via CONTROL sub-function
DINPLEN	19	Computed from the initialized input buffer
DINPLREC	255	Fixed
DOUTLEN	256	Fixed
DOPTION1	0	0 neither INPUT nor OUTPUT created 1 OUTPUT in output file, INPUT acc. DOPTION2
DOPTION2	3	0 Input from DINPUT buffer or parameter (DOPTION8) 1 Input from file 2 Input from file, print if DOPTION1 = 1 3 Input like 0, print if DOPTION1 = 1 4 Input like 0, then input from file, for EOF DOPTION2 =1 set 5 Input like 4, print if DOPTION1 = 1
DOPTION3	1	0 DINPLEN is size of INPUT buffer 1 DINPLEN is number of input lines
DOPTION4	0	0 Return to caller if all input lines processed 1 Return after each line
DOPTION5	2	0 Only output lines in DOUTPUT buffer 1 Only messages 2 Output lines and messages

DOPTION6	1	0 Output lines in format of POST/MAIL 1 In normal print output format
DOPTION7	1	0 Access Call Work File (ACWF) if necessary 1 no ACWF
DOPTION8	1	0 Input is in DINPUT buffer 1 Input is in 3rd parameter of UI call
DOPTION9	2	0 Execution in Batch 1 Execution in Online (then Break-Detection) 2 Execution Online without Break.

Appendix B

Return Codes of MethodManager/ControlManager

This appendix describes the MethodManager/ControlManager return codes as documented in the *ASG-DataManager User Interface*. In case of inconsistencies in this documentation, the *ASG-DataManager User Interface* takes precedence.

DRETURN	0	The output contains data
	1	The output contains messages
	2	No output created
	3	No output and the MethodManager/ControlManager session has ended
DRETURN1	0	The work file was used for storing results
	1	The work file was not required
	2	The work file was required for storing, but could not be opened
DRETURN2	0	The input has been processed
	1	EOF in the input (command not terminated properly?)
	2	The input file could not be opened (not possible in REXX interface)
	3	A severe error has interrupted the processing
	4	The processing was interrupted with the BREAK key
	5	Unable to connect to MPSF
	6	Client timeout occured
	7	Server task has abended
	8	Server shutdown was done
		The values 1 or 2 are only possible if DOPTION2 = 1,2,4, or 5 was set. Should not normally occur in the REXX interface.

DSEVRITY	( )	No messages
	I	One or more information messages
	W	One or more warning messages
	E	One or more error messages
	S	One or more severe error messages
	C	One or more critical error messages

Appendix C
Return Codes

These are the return codes of the REXX interface:

```
0         all actions executed. If MPR commands were executed and DRETURN2
           = 0, a maximum of I messages were detected.
4         all actions executed. If MPR commands were executed and DRETURN2
           = 0, a maximum of W messages were detected.

8         all actions executed. If MPR commands were executed and DRETURN2
           = 0, a maximum of E messages were detected.

12        all actions executed. If MPR commands were executed and DRETURN2
           = 0, a maximum of S messages were detected.

16        all actions executed. If MPR commands were executed and DRETURN2
           = 0, a maximum of C messages were detected.

32        all actions executed. If MPR commands were executed and DRETURN2
           = 0, other terminating conditions were detected.

36        DRETURN2 = 1: End-Of-Data on command input

37        DRETURN2 = 2: MPIN missing or invalid

38        DRETURN2 = 3: MPR failed (typically abended)

39        DRETURN2 = 4: BREAK-KEY (only for normal online UI)

40        DRETURN2 = 5: Unable to connect to MPSF

41        DRETURN2 = 6: Client timeout occured

42        DRETURN2 = 7: Server task has abended

43        DRETURN2 = 8: Server shutdown has occured

50    ### SEVERE   ERROR IN DPUT-DACCESS SAVE
                  the variable MPIODACCESS.xxx could not
                  be saved
51    ### SEVERE   ERROR IN DPUT-DACCESS SAVE
                  the variable MPIODACCESS.xxx.TYP could not
                  be saved
```

```
52     ### SEVERE  ERROR IN DPUT-DACCESS SAVE
                   the variable MPIODACCESS.0 could not
                   be saved
                   the variable attribute.0 could not
                   be saved
53     ### SEVERE  ERROR IN DPUT-DACCESS SAVE
                   the variable attribute.xxx could not
                   be saved
54     ### SEVERE  ERROR IN DPUT-DACCESS SAVE
                   reserved
55     ### SEVERE  ERROR IN DPUT-        ASSIGN
                   reserved
56     ### SEVERE  ERROR IN DPUT-DACCESS SAVE
                   the variable MPIODACCESS.xxx.CNT could not
                   be saved
                   the variable MPIODACCESS.xxx.LOW could not
                   be saved
57     ### SEVERE  ERROR IN DPUT-DACCESS SAVE
                   the number of values of an attribute is
                   invalid, please check MPREXDQMTS ....
58     ### SEVERE  ERROR IN DPUT-DACCESS SAVE
                   the specification Low:High for an attribute is
                   invalid, please check MPREXDQMTS ...
59     ### SEVERE  ERROR IN DPUT-DACCESS SAVE
                   the data type specification for an attribute is
                   invalid, please check MPREXDQMTS ...
60     ### SEVERE  ERROR IN DPUT-DACCESS SAVE
                   The data type of an attribute needs to be
                   followed by either the Low:High specification
                   or directly by the value assignment.
                   Please check MPREXDQMTS ... output
63     ### SEVERE  ERROR IN DPUT-DACCESS SAVE
                   The name of an attribute is >= 80 Bytes.
                   Please check MPREXDQMTS ... output
64     ### SEVERE  MISSING/INVALID FUNCTION
68     ### SEVERE  INVALID DURING PENDING OPERATION
69     ### SEVERE  INVALID SUB-FUNCTION OF MPIO
70     ### SEVERE  INVALID DCONTROL PARAMETER
71     ### SEVERE  BAD # OF COMMAND LINES
72     ### SEVERE  INVALID OUTMAP PARAMETERS
73     ### SEVERE  MISSING 1ST OPERAND IN DGET
74     ### SEVERE  MISSING 2ND OPERAND IN DGET
75     ### SEVERE  ASSIGNMENT ARRAY RANGE INVALID
76     ### SEVERE  ASSIGNMENT VALUE TOO LARGE
77     ### SEVERE  ASSIGNMENT ERROR
80     ### SEVERE  NO DCONTROL AREA
81     ### SEVERE  NOT LOCKED TO THIS TASK
82     ### SEVERE  INVALID LENGTH
83     ### SEVERE  HEADER IS INVALID
90     ### SEVERE  REXXMSPI: Invalid MPIOAREA Count ( > 8)
91     ### SEVERE  REXXMSPI: Invalid MPIOLOGON.0 Count
92     ### SEVERE  REXXMSPI: Too many MPIOLOGON.xxx lines
93     ### SEVERE  REXXMSPI: Invalid MPIOAREALINES Value

94     ### SEVERE  REXXMSPE: IKJMSPCT abended due to CMR-Abend
95     ### SEVERE  REXXMSPI: termination after IKJMSPCT-Abend
```

```
    For sub-functions not available:

99     ### SEVERE  NOT YET IMPLEMENTED


    Internal errors that ASG should be notified of:

100    ### SEVERE  REXXMSPI: SHUTDOWN for non-resident area
101    ### SEVERE  REXXMSPI: SHUTDOWN and Subtask is not active
102    ### SEVERE  REXXMSPI: TCB-addresses missmatch
103    ### SEVERE  REXXMSPI: IKJMSPCT is locked
104    ### SEVERE  REXXMSPI: IKJMSPCT locked denied
105    ### SEVERE  REXXMSPI: IKJMSPCT not properly released
106    ### SEVERE  REXXMSPI: IKJMSPWD not properly released
```

Index

A
applications
    resuming 18
    terminating 18

C
components 3
conventions page v

D
DACCESS function 31

I
initializing
    REXXMSPI 6–9
initializing the interface 4
interface modes 4
ISPF split screen mode 13

L
LOGON executing 10

M
modes
    ISPF split screen 13
    PERMANENT 5
    RESIDENT 5
    SERVER 5
    TEMPORARY 4
MPIO syntax 21
MPIOMPFCT variable 6

O
output buffer 25

P
parameters
    customization 35
PERMANENT mode 5
persistent sessions 17
    creating 17

R
RESIDENT mode 5
resuming applications 18
return codes 49
    MMR/CMR 47
REXX procedures
    starting 15

S
SERVER mode 5
    activating 9
SWAP function 41

T
TEMPORARY mode 4
terminating applications 18

V
variables 23