

CA OPS/MVS® Event Management and Automation

Command and Function Reference

Release 12.1



This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the "Documentation") is for your informational purposes only and is subject to change or withdrawal by CA at any time. This Documentation is proprietary information of CA and may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA.

If you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with "Restricted Rights." Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2013 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

CA Technologies Product References

This document references the following CA products:

- CA 7™ Workload Automation (CA 7)
- CA ACF2™ for z/OS (CA ACF2)
- CA Automation Point
- CA Common Services for z/OS (CCS for z/OS)
- CA Jobtrac™ Job Management (CA Jobtrac)
- CA MIA Tape Sharing (CA MIA)
- CA MIC Message Sharing (CA MIC)
- CA MIM™ Resource Sharing (CA MIM)
- CA Netman™ (CA Netman)
- CA NetMaster™ Network Automation (CA NetMaster)
- CA NetSpy® Network Performance (CA NetSpy)
- CA NSM
- CA OPS/MVS® Event Management and Automation (CA OPS/MVS)
- CA Remote Console™ (CA Remote Console)
- CA Scheduler® Job Management (CA Scheduler)
- CA SYSVIEW® Performance Management (CA SYSVIEW)
- CA Top Secret® for z/OS (CA Top Secret)

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At <http://ca.com/support>, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at <http://ca.com/docs>.

Documentation Changes

The following documentation updates have been made since the last release of this documentation:

Note: In PDF format, page references identify the first page of the topic in which a change was made. The actual change may appear on a later page.

- Updated the [MQ GET—Retrieve Messages](#) (see page 184) section.
- Updated the [MQ PUT—Put Messages on the Queue](#) (see page 185) section.
- Updated the [ADDRESS MQ Return Codes](#) (see page 202) section.
- Added the [Indirect Destination IDs](#) (see page 252) section.
- Updated the [COF ACTIVATE—Add Transient Data Queue Names](#) (see page 252) section.
- Updated the [COF DEACTIVATE—Delete Transient Data Queue Names](#) (see page 252) section.
- Added [PPRCCMD Request Types](#) (see page 364) sections.
- Added the [PPRCCMD Request Examples](#) (see page 364) section.
- Updated the [COF DEFINE—Create Transient Data Queue Name List](#) (see page 256) section.
- Updated the [COF List Command Output](#) (see page 260) section.
- Updated the [ALTER TABLE Statement—Add a Table Column](#) (see page 418) section.
- Updated the [UPDATE Statement—Insert Values](#) (see page 454) section.
- Updated the [CLOSE Statement—End Cursor Operation](#) (see page 422) section.
- Updated the [CREATE TABLE Statement—Define Relational Table](#) (see page 424) section.
- Updated the [DECLARE CURSOR Statement—Define a Cursor](#) (see page 431) section.
- Updated the [DELETE FROM Statement—Delete a Row](#) (see page 433) section.
- Updated the [DROP TABLE Statement—Remove Table from DIV Data Set](#) (see page 436) section.
- Updated the [FETCH Statement—Retrieve Row Values](#) (see page 438) section.
- Updated the [INSERT Statement—Insert Table Rows](#) (see page 441) section.
- Updated the [OPEN Statement—Initiate Cursor Operation](#) (see page 444) section.
- Updated the [SELECT Statement—Extract Relational Table Data](#) (see page 447) section.
- Updated the [UPDATE Statement—Insert Values](#) (see page 454) section.

- Updated the [OPSCAWTO Function](#) (see page 492) section.
- Updated the [OPSCOLOR Function](#) (see page 496) section.
- Updated the [Types of OPSINFO Functions](#) (see page 534) section.
- Updated [OPSLOG Expanded Format](#) (see page 601) section.
- Added the [ADDRESS ALERTMON Host Environment](#) (see page 41) section.
- Updated the [OPSDEV Function](#) (see page 502) section.
- Updated the [Format of Device Information Returned](#) (see page 504) section.
- Added the [ADDRESS HWS Security](#) (see page 149) section.
- Added the [ADDRESS HWS Return and Reason Codes in REXX Programs](#) (see page 147) section.
- Added the [HWS GETATTR— Get the value of a system attribute](#) (see page 141) section.
- Added the [ADDRESS HWS Commands](#) (see page 139) section.
- Added the [HWS SETATTR— Update the value of a system attribute](#) (see page 143) section.
- Added the [ADDRESS HWS Command Format](#) (see page 139) section.
- Added the OPSINFO('UTC') section.
- Added the OPSINFO Parameters section.
- Updated the [OPSWTO Command Processor—Issue WTO and WTOR Messages](#) (see page 780) section.
- Added the [HWS SENDCMD—Sends command to HiSrv](#) (see page 140) section.
- Updated the [ADDRESS OPSCTL Host Environment](#) (see page 245) section.
- Updated the [QUERY DEVICES LOCAL DEVICE—Query Device ID](#) (see page 355) section.
- Updated the [List of POI Command Processors](#) (see page 690) section.
- Updated the [OPSWTO Command Processor—Issue WTO and WTOR Messages](#) (see page 780) section.
- Added the [PPRCMD FREEZE— Updates to the Secondary Subsystem](#) (see page 376) section.
- Added the [PPRCMD RUN—Resumes I/O on Primary Volumes](#) (see page 378) section.

Contents

Chapter 1: Using This Reference 21

Introduction.....	21
How You Interpret Syntax Diagrams.....	22

Chapter 2: Host Environment Commands 23

List of Host Environment Commands	24
Host Command Defined	32
Host Environments	32
External Host Environments	34
Output Returned from External Environments	34
How Host Command Delimiters Work	35
Responses from Host Commands	35
External Data Queue.....	35
Issue Commands from Rules without Host Command Responses	36
How OPS/REXX Gets Host Command Output	36
Return Codes from Host Commands	38
Security Considerations for Host Commands.....	38
REXX Instructions Against Host Commands	39
How Rules Process Host Commands	40
ADDRESS ALERTMON Host Environment	41
ADDRESS AOF Commands	42
ADDRESS AOF Command Format	43
How ADDRESS OPSCTL Routes AOF Commands to Other Systems	45
AOF Commands and the Test Facility	45
Command Execution When the Product Is Down	45
AOF COMPILE—Create Compiled Rules	46
AOF DELCOMP—Delete Compiled Rules	47
AOF DISABLE —Disable Rules.....	48
AOF ENABLE—Enable Rules	49
AOF INDEX—Return Rules Data Set Information	52
AOF LIST—Provide Rule and Rule Sets Statistics	52
AOF LISTCOMP—List Compiled Rule Statistics	56
AOF LISTINST—List Enabled Rule Statistics	60
AOF LISTSRC—List Source Code	67
AOF RESETAUTO—Prevent Automatic Enabling of Rule.....	68
AOF SETAUTO—Enable a Rule at Startup	69

AOF SUBSYS—Set Subsystem Identifier	70
ADDRESS AOF Return Codes	71
ADDRESS AP Commands	74
AP NMFind—Invoke Notification Manager	74
AP PPQ—Write to Queue	77
AP REXX—Execute a REXX EXEC	79
ADDRESS AP Return Codes	79
ADDRESS EPI Commands	82
Types of ADDRESS EPI Commands	82
ADDRESS EPI Return Codes in REXX Programs.....	84
EPI BIND—Dedicate a VTAM Session.....	85
EPI CHANGE—Change Virtual Terminal Definition	86
EPI DEBUG—Control VTAM Exit Debugging	89
EPI DEFINE—Define a Virtual Terminal to EPI	89
EPI DELETE—Delete Virtual Terminal Definitions.....	92
EPI DEQ—Release Virtual Terminal Ownership.....	93
EPI DISABLE Command—Disable Virtual Terminals.....	94
EPI ENABLE—Activate a Virtual Terminal	97
EPI ENQ—Enqueue a Virtual Terminal	98
EPI HELP—Returns Valid EPI Commands	99
EPI INQINPUT—Input Indicator	100
EPI LIST—Display Virtual Terminal Status.....	101
EPI LOGOFF—Log Off a Virtual Terminal.....	102
EPI LOGON—Log On a Virtual Terminal	103
EPI MSGID—Prefix Message IDs.....	104
EPI MVCURSOR—Move Cursor Position	105
EPI PEEK—Display Screen Buffer in Hexadecimal	106
EPI POKE—Modify Screen Buffer Contents	108
EPI RDCURSOR—Request Cursor Position.....	110
EPI RDSCREEN—Retrieve Virtual Terminal Screen	111
EPI RDSCRROW—Return Text from Specified Row.....	112
EPI SESSCMD—Issue Command to VTAM Session.....	113
ops--EPI SETMODEL—Set Model Number	120
EPI SETTERM—Set the Virtual Terminal.....	121
EPI SETUNAME—Set User-Defined Name	122
EPI SUBATTR—Return Attribute Characters.....	123
EPI SUBSYS—Direct Host Commands	124
EPI SUBUNPT—Set Substitute Character for Unprintable Characters.....	125
EPI TIMEOUT—Set Timeout Value	126
EPI TRACE—Set Trace Value.....	127
EPI TRIM—Remove Leading and Trailing Blanks	128
EPI TYPE—Simulate Typing	129

EPI TYPESEC—Simulate Typing without Displaying in OPSLOG	133
EPI TYPETEST—Test a TYPE Command	134
EPI UNBIND—Undo a Bind	136
EPI WAIT—Delay Rule Processing.....	137
EPI WAITTOD—Set Processing Delay Time.....	138
ADDRESS HWS Commands	139
ADDRESS HWS Command Format.....	139
HWS SENDCMD—Sends command to HiSrv	140
HWS GETATTR— Get the value of a system attribute	141
HWS SETATTR— Update the value of a system attribute	143
ADDRESS HWS Common Keywords	145
ADDRESS HWS Return and Reason Codes in REXX Programs	147
ADDRESS HWS Security	149
ADDRESS LXCON Commands.....	149
ADDRESS LXCON LXCMD—Send Commands To Linux System	150
ADDRESS LXCON VMCMD—Send Commands To VM System	151
ADDRESS LXCON LIST—Return VM And Linux System Properties	152
WTOR Records That OPSTATUS Returns	155
XML Encapsulation of Types 1 and 2 Responses	156
ADDRESS MIM Command	156
Common REXX Variables.....	157
Address MIM Formats	157
Common Keywords	158
Common Return and Reason Codes	159
ENV Function.....	161
QNAME Function	168
TAPE Function	174
ADDRESS MQ Commands	181
ADDRESS MQ Command Format	181
MQ CONNECT—Provide Queue Manager Connection.....	182
MQ OPEN—Open a Queue	182
MQ GET—Retrieve Messages	184
MQ PUT—Put Messages on the Queue	185
MQ SET—Set Attributes	188
MQ INQUIRE—Inquire Attributes	191
MQ CLOSE—Close Queue	199
MQ DISCONNECT—Disconnect Queue Manager	200
ADDRESS MQ Debug Parameter	200
How to Use ADDRESS MQ Commands	201
ADDRESS MQ Return Codes	202
ADDRESS NETMAN Host Environment	203
ADDRESS NETMASTR Host Environment	203

ADDRESS NETMASTR Command—Create and Work with Alerts	204
ADDRESS NETMASTR Return Codes	215
ADDRESS OPER Host Environment	217
Utilize the ADDRESS OPER Host Environment	218
Dispatch a Command and Continue Processing	219
Issue a Command and Process the Command Response	220
Usage of ADDRESS OPER	221
ADDRESS OPER Security With Audit Ability	230
ADDRESS OPER Keywords	231
Formats for ADDRESS OPER Commands	232
ADDRESS OPER Return Codes	242
IMS Type 2 Message Return and Reason Codes	245
ADDRESS OPSCTL Host Environment	245
ADDRESS OPSCTL Commands	245
ADDRESS OPSCTL Return Codes	249
ADDRESS OPSCTL Commands for the COF	250
Wildcards in OPSCTL COF Commands	250
Generate the DEFAULT List	250
Destination IDs	251
Indirect Destination IDs	252
COF ACTIVATE—Add Transient Data Queue Names	252
COF DEACTIVATE—Delete Transient Data Queue Names	254
COF DEFINE—Create Transient Data Queue Name List	256
COF DELETE—Delete Transient Data Queue Name List	258
COF LIST—Display Transient Data Destination List	259
OPSCTL COF Return Codes	262
Keywords Common to All ADDRESS OPSCTL COF Commands	263
ADDRESS OPSCTL Commands for the ECF	263
ECF LIST—List Console Users	264
ECFLIST Command Output	264
OPSCTL ECF LIST Return Codes	265
ADDRESS OPSCTL Commands for the MSF	266
MSF ACTIVATE—Activate a Session	266
MSF DEACTIVATE—End a Session	267
MSF DEFAULT—Specify System Name and Wait Defaults	268
MSF DEFINE—Define VTAM LU to LU Sessions	269
MSF DELETE—Delete Definitions	272
MSF LIST—Return Session Information	273
MSF START—Start MSF	277
MSF STOP—Stop MSF Sessions	278
OPSCTL MSF Return Codes	279
Keywords Common to ADDRESS OPSCTL MSF Commands	281

ADDRESS OPSCTL Commands for OPSLOG	282
OPSLOG ACTIVATE—Activates an OPSLOG	283
OPSLOG DEACTIVATE—Deactivate an OPSLOG	285
OPSLOG DEFINE—Define an OPSLOG	286
OPSLOG DELETE—Delete an OPSLOG	288
OPSLOG LIST—List All OPSLOG Definitions.....	288
OPSLOG RESET—Reset Active OPSLOG	291
OPSLOG SETLIVE—Make OPSLOG Live.....	292
OPSCTL OPSLOG Return Codes	293
Keywords Common to ADDRESS OPSCTL OPSLOG Commands	297
ADDRESS OPSCTL Commands for OSF.....	297
OSF EXECSTATS—Return Performance Statistics.....	298
OSF LIST—Return Active Server Statistics.....	300
OSF QUEUES—Display Server Queue Status	303
OSF RESETQ—Reset Execute Queue.....	304
OSF STOP—Stop the Server Address Space.....	305
Terminate a Server Immediately.....	305
OPSCTL OSF Return Codes	306
Keywords Common to All ADDRESS OPSCTL OSF Commands.....	306
ADDRESS OPSDYNAM Host Environment	308
OPSDYNAM Used as a TSO/REXX or OPS/REXX Function	308
ADDRESS OPSDYNAM Compared to TSO ALLOCATE	308
ADDRESS OPSDYNAM Commands	309
OPSDYNAM ALLOCATE—Allocate Data Set	309
OPSDYNAM CONCAT—Concatenate DDNames	320
OPSDYNAM DECONCAT—Deconcatenate DDNames	321
OPSDYNAM FREE—Free DD and Data Set Names.....	322
OPSDYNAM INFO—Display Data Set Information.....	324
OPSDYNAM Command Return Codes	325
Keywords Common to All ADDRESS OPSDYNAM Commands	326
ADDRESS OSF—Route Commands to TSO Server	328
ADDRESS OSF Return Codes	328
ADDRESS OSFTSL—Route Commands to TSL Server	329
ADDRESS OSFTSL Return Codes	329
ADDRESS OSFTSP—Route Commands to TSP Server	330
ADDRESS OSFTSP Return Codes	330
ADDRESS SERVDESK—Create Service Desk Requests.....	331
OPS/REXX Output Variables	333
ADDRESS SERVDESK Formats	334
Command Return and Reason Codes	337
ADDRESS SOF—Manage Devices.....	339
Execution Considerations	339

Common Keywords of ADDRESS SOF	340
Common REXX Variables.....	341
Sample Rules and Programs	341
Return and Reason Codes	341
Special Character Handling Considerations.....	343
Device ID String Composition.....	343
Non-QUERY SOF Request Types	344
QUERY SOF Request Types	348
PPRCMD Request Types	364
ADDRESS SOF Examples	380
ADDRESS SQL Commands	382
ADDRESS SYSVIEWE—Interface to CA SYSVIEW	383
ADDRESS SYSVIEWE Return Codes	384
ADDRESS TSO—Route Commands to TSO	384
ADDRESS TSO Return Codes	384
ADDRESS USS Commands	385
ADDRESS USS USSCMD—Issue UNIX Commands.....	386
ADDRESS USS CMD—Issue USS Commands	387
ADDRESS USS DOM—Remove Message from Console Display.....	388
ADDRESS USS PING—Ping CA Event Manager	389
ADDRESS USS REPLY—Reply to WTOR	390
ADDRESS USS WTO—Send Message to Event Manager	391
ADDRESS USS WTOR—Send and Receive Messages	392
API Keywords	393
Keywords Common to All ADDRESS USS Commands	397
ADDRESS USS Return Codes	398
USSCODE and USSREASON Variables	400
ADDRESS WTO—Issue WTO Messages	401
ADDRESS WTO Keywords.....	402
ADDRESS WTO and CA Automation Point	411
ADDRESS WTO Return Codes	411
Usage Notes for ADDRESS WTO	414

Chapter 3: Relational Data Framework Reference 415

SQL Statements	415
Expressions Used in SQL Statements	415
ALTER TABLE Statement—Add a Table Column	418
CLOSE Statement—End Cursor Operation.....	422
CREATE TABLE Statement—Define Relational Table.....	424
DECLARE CURSOR Statement—Define a Cursor.....	431
DELETE FROM Statement—Delete a Row	433

DROP TABLE Statement—Remove Table from DIV Data Set	436
FETCH Statement—Retrieve Row Values	438
INSERT Statement—Insert Table Rows	441
OPEN Statement—Initiate Cursor Operation.....	444
SELECT Statement—Extract Relational Table Data	447
UPDATE Statement—Insert Values	454
OPS/REXX Programs.....	457
READTBL Program—Read Backup Data Set.....	457
WRITETBL Program—Create Backup RDF Table.....	458
Relational Table Editor Batch API Functions	460
Non-zero Return Values	460
COPY Function—Copy Data Rows	461
DELETE Function—Remove Table	461
DROPCOLS Function—Remove Table Columns	461
FREE Function—Remove the Global Variable Enqueue	462
RENAME Function—Rename Tables.....	462
RENCOLS Function—Rename Column Names	463
TRANSFER Function—Copy Matching Column Names	463
UCCCOPY Function—Add Upper Case to Copied Rows	464
UCPKCOPY Function—Add Upper Case to Primary Keys	464

Chapter 4: OPS/REXX Built-in Functions 465

Overview.....	467
Error Handling For REXX Functions	467
Functions Not Supported in OPS/REXX	467
Usage Recommendation.....	468
DATE Function.....	468
Valid DATE Formats	469
Usage Notes for the DATE Function.....	469
FIND Function	469
INDEX Function.....	471
OPSARM Function	471
Values Returned	473
Additional Variable Output.....	474
Function Argument	475
ELEMNAME Argument	476
ELEMTYPE Argument	476
EVENTEXIT Argument	476
STARTTEXT Argument.....	476
TERMTYPE Argument	477
TIMEOUT Argument.....	477

TOKEN Argument	478
OPSARMST Function	478
Values Returned	480
Variable Output	480
OPSAUTO Function	485
Usage Notes for OPSAUTO	487
OPSBITS Function	487
OPSBITS Arguments	487
Interpreting Character Strings	487
OPSBITS and Variable Names	489
OPSDOM Function.....	490
OPSCA7 Function.....	490
Values OPSCA7 Returns	492
OPSCAWTO Function	492
Installation Considerations	495
Return Codes from OPSCAWTO	495
OPSCLEDQ Function.....	495
OPSCOLOR Function	496
OPSCPF Function.....	497
Records That OPSCPF Returns	498
OPSCPU Function	499
Records That OPSCPU Returns.....	499
OPSEDLV Function.....	501
OPSDEV Function.....	502
Format of Device Information Returned	504
OPSDOM Function	509
Values OPSDOM Returns	510
OPSDUMP Function	510
Values OPSDUMP Returns	511
OPSECURE Function.....	511
Extract Data from the ACEE.....	512
Fieldname Argument	512
Verify Data Set Access	517
Results of Access Queries	518
Excessive Access Queries	518
Retrieve a Logon ID Field	518
Conversion of Retrieved Fields	519
Request Security Product Data.....	519
Name Argument	520
Validate a Password.....	522
Request General Resource Data.....	523
Requestcode Values.....	523

OPSENG Function	524
OPSRC and OPSRS Variables	527
ENQ Information Returned in the EDQ	528
OPSGETVL Function	530
Namemask Argument	530
Keywords for OPSGETVL	531
Usage Notes for OPSGETVL	532
OPSHFI Function	532
Keywords for OPSHFI	533
OPSINFO Function	533
Environmental OPSINFO Functions for JES2	534
Types of OPSINFO Functions	534
OPSIPL Function	552
OPSIPL Returned Values	559
OPSJES2 Function	559
Record Fields for Inquiries	564
OPSJESX Function	574
Keywords for OPSJESX	575
OPSLIKE Function	598
OPSLOG Function	599
Basic Format	599
Expanded Format	601
OPSLOG Function Examples	608
OPSLOG Function Return Codes	609
Security for the OPSLOG Function	609
Sample Rules Programs	610
OPSMTRAP Function	610
Values Returned	611
OPSPDS Function	612
Values Returned	615
Variable Output	616
OPSPRM Function	616
Parmname Argument	617
Value Argument	617
Using OPSPRM to Display and Set Message Severity Codes	617
OPSTATUS Function	618
ASID Records That OPSTATUS Returns	621
IMS Records That OPSTATUS Returns	626
WTOR Records That OPSTATUS Returns	626
ASID Workload Manager Records That OPSTATUS Returns	627
WLM Returns and Descriptions	629
Sample Output from OPSTATUS	629

OPSTORE Function	631
OPSUBMIT Function.....	632
Usage Notes for OPSUBMIT	633
OPSUSS Function.....	633
Values Returned	634
Variable Output	634
Function Argument	639
Subfunction Argument	639
Entity Argument	640
Keyword Argument	640
Example: OPSUSS	641
OPSVLUE Function.....	641
Limits for Global Variable Stems	642
OPSVLUE and Cross-system Operations.....	643
Actions OPSVLUE Takes	643
Examples: OPSVLUE	653
OPSVASRV() Function	654
Utilization Environments	654
Installation Requirements	654
Coding Syntax.....	655
Generated Output.....	659
Sample Coding Statements	665
Debugging Guidelines.....	669
Return and Reason Codes	669
OPSVSAM Function	670
Arguments for OPSVSAM	670
Access Strategies.....	671
Output Variables	671
Types of OPSVSAM Functions	672
OPSWAIT Function	675
Values OPSWAIT Returns	676
Usage Notes for OPSWAIT	676
OPSWLM Function	677
OPSWLM Scheduling Examples	680
OPSWORD Function.....	680
OPSYSPLX Function	681
Record That OPSYSPLX Returns	682
OPSYSSYM Function.....	684
Records that OPSYSSYM Returns	686
TIME Function—Returns Current or Elapsed Time	686
TRACE Function.....	687

Chapter 5: POI Command Processors

689

List of POI Command Processors.....	690
Benefits of POI.....	692
Security Considerations for POI.....	693
Specify a Subsystem ID on a POI Command Processor	693
Change the Default Subsystem ID.....	694
Specify an MSF System ID on a POI Command Processor	694
How to Invoke POI Command Processors	695
Issue POI Commands in Batch Mode.....	696
Abbreviations in Command Processor Text	696
Using OPS/REXX Efficiently.....	697
How to Use POI Command Processors.....	698
Run POI Commands in UNIX.....	699
OPSBRW Command Processor—View Automation Events	701
OPSCMD Command Processor—Issue z/OS, JES, VM, and IMS Commands	702
OPSCMD Security With Audit Ability.....	706
Enter Subcommand Mode.....	706
Guidelines for Issuing OPSCMD	706
Output Variables Generated by OPSCMD	707
IMS Type 2 Message Considerations	708
Command Characters for Subsystem Commands.....	708
OPSCMD Command Characters for IMS Operation	709
Issue VM Commands Through OPSCMD	709
Issue the OPSCMD from CLISTs	709
Use OPSCMD Subcommand Mode	710
Use OPSCMD with z/OS REPLY Command	711
How OPSCMD Retrieves Command Output	712
CLIST Variables That TSO/E Creates	713
CLIST Variables That OPSCMD Creates.....	713
OPSCMD Return Codes	714
OPSDLV Command Processor—Delete Global Variables	717
OPSDLV Return Codes	720
OPSDOM Command Processor—Delete Operator Messages	720
Return Codes.....	721
OPSEXEC Command Processor—Run a Specified Program.....	722
OPSGTV Command Processor—Retrieve Global Variable Value.....	722
Return Codes.....	725
OPSGTVL Command Processor—Get Global Variable Names	726
CLIST or REXX Variables Generated by OPSGETVL	730
Return Codes.....	731
OPSHFI Command Processor—Store Global Variable Values	732

Variable Characteristics.....	737
OPSHFI Return Codes	738
OPSIMEX Command Processor—Executes a Member as a REXX Program	739
OPSPARM Command Processor	740
OPSPARM For mat—Change Parameters	740
OPSPARM For mat—Display Parameters.....	741
Using OPSPARM to Display and Set Message Severity Codes	743
OPSPARM Return Codes	743
OPSQL Command Processor—Invoke SQL Statements	745
OPSREPLY Command Processor—Compare WTOR Messages.....	746
How OPSREPLY Determines Which WTORs Receive Replies.....	752
Considerations for Issuing OPSREPLY	753
OPSREPLY Return Codes	754
OPSREQ Command Processor—Invoke AOF Request Rules	755
Security and the OPSREQ Command Processor	756
OPSREQ Return Codes	757
OPSRMT Command Processor—Issue TSO Commands to Other Systems	758
Using OPSRMT in Subcommand Mode	760
OPSRMT Return Codes	761
OPSSETV Command Processor—Create or Update Global Variable Values.....	762
OPSSMTBL Command Processor—Maintain the Directory Table.....	766
OPSSMTBL Invoked as a REXX Function.....	771
OPSSMTBL Return Codes	771
OPSVASRV Command Processor	773
Definition and Usage	773
Installation Requirements	773
Utilization Environments	773
Coding Syntax.....	774
Sample Coding Statements.....	775
Return and Reason Codes.....	776
OPSVIEW Command Processor—Access OPSVIEW.....	776
OPSWAIT Command Processor—Suspend CLIST or REXX Processing.....	777
OPSWAIT Used Without Event Keywords.....	779
OPSWAIT Return Codes	779
OPSWTO Command Processor—Issue WTO and WTOR Messages	780
STATESET Program—Set State Values.....	788
STATESET Return Codes	791
 Appendix A: REXX/EDQ Output from IMS Type 2 Message Responses	 793
XML Encapsulation of Types 1 and 2 Responses	793
Type 1 Response.....	794

Type 2 Response.....	795
Standard REXX and EDQ Output.....	796

Chapter 1: Using This Reference

This section contains the following topics:

[Introduction](#) (see page 21)

[How You Interpret Syntax Diagrams](#) (see page 22)

Introduction

This guide describes the following features of CA OPS/MVS:

- The commands OPS/REXX programs or AOF rules can issue to various host environments
- Reference information for the Relational Data Framework (RDF) facility
- The built-in functions of OPS/REXX
- The command processors that the Programmable Operations Interface (POI) uses to interact with the console facilities of z/OS

How You Interpret Syntax Diagrams

The following legend has been provided to help you read the syntax diagrams provided for each command:

ALL UPPERCASE or all lowercase

Identifies control verbs, object identifiers, and arguments that must be exactly as shown.

Note: Whether you use all uppercase or all lowercase depends on which platform you are documenting.

MIXed Case

Identifies abbreviations. The uppercase letters are the minimum abbreviation; lowercase letters are optional.

lowercase italics

Identifies a value that you must supply.

[] Brackets

Identifies optional arguments. Brackets are put around each argument. Multiple arguments are separated by a space.

{ } Braces

Identifies required arguments among multiple arguments. Braces are put around each argument. Multiple arguments are separated by a space.

Underlined Text

Indicates default values.

|

Separates alternative arguments (choose one).

...

Indicates that the preceding items or group of items can be repeated.

Chapter 2: Host Environment Commands

This chapter describes the commands OPS/REXX programs or AOF rules can issue to various host environments, including facilities of CA OPS/MVS such as POI and MSF, and system facilities such as TSO and ISPF.

This section contains the following topics:

- [List of Host Environment Commands](#) (see page 24)
- [Host Command Defined](#) (see page 32)
- [Host Environments](#) (see page 32)
- [Responses from Host Commands](#) (see page 35)
- [REXX Instructions Against Host Commands](#) (see page 39)
- [How Rules Process Host Commands](#) (see page 40)
- [ADDRESS ALERTMON Host Environment](#) (see page 41)
- [ADDRESS AOF Commands](#) (see page 42)
- [ADDRESS AP Commands](#) (see page 74)
- [ADDRESS EPI Commands](#) (see page 82)
- [ADDRESS HWS Commands](#) (see page 139)
- [ADDRESS LXCON Commands](#) (see page 149)
- [WTOR Records That OPSTATUS Returns](#) (see page 155)
- [XML Encapsulation of Types 1 and 2 Responses](#) (see page 156)
- [ADDRESS MIM Command](#) (see page 156)
- [ADDRESS MQ Commands](#) (see page 181)
- [ADDRESS NETMAN Host Environment](#) (see page 203)
- [ADDRESS NETMASTR Host Environment](#) (see page 203)
- [ADDRESS OPER Host Environment](#) (see page 217)
- [ADDRESS OPSCTL Host Environment](#) (see page 245)
- [ADDRESS OPSCTL Commands for the COF](#) (see page 250)
- [ADDRESS OPSCTL Commands for the ECF](#) (see page 263)
- [ADDRESS OPSCTL Commands for the MSF](#) (see page 266)
- [ADDRESS OPSCTL Commands for OPSLOG](#) (see page 282)
- [ADDRESS OPSCTL Commands for OSF](#) (see page 297)
- [ADDRESS OPSDYNAM Host Environment](#) (see page 308)
- [ADDRESS OPSDYNAM Commands](#) (see page 309)
- [ADDRESS OSF—Route Commands to TSO Server](#) (see page 328)
- [ADDRESS OSFTSL—Route Commands to TSL Server](#) (see page 329)
- [ADDRESS OSFTSP—Route Commands to TSP Server](#) (see page 330)
- [ADDRESS SERVDESK—Create Service Desk Requests](#) (see page 331)
- [ADDRESS SOF—Manage Devices](#) (see page 339)
- [ADDRESS SQL Commands](#) (see page 382)
- [ADDRESS SYSVIEWE—Interface to CA SYSVIEW](#) (see page 383)
- [ADDRESS TSO—Route Commands to TSO](#) (see page 384)
- [ADDRESS USS Commands](#) (see page 385)
- [ADDRESS WTO—Issue WTO Messages](#) (see page 401)

List of Host Environment Commands

The following CA OPS/MVS host environment commands are described in this chapter.

ADDRESS AOF commands

Sends commands to the AOF.

ADDRESS AOF COMPILE

Creates compiled versions of rules.

ADDRESS AOF DELCOMP

Deletes compiled versions of rules.

ADDRESS AOF DISABLE

Makes rules ineligible to run.

ADDRESS AOF ENABLE

Makes rules eligible to run.

ADDRESS AOF INDEX

Allows an OPS/REXX program to find the rules CA OPS/MVS is using.

ADDRESS AOF LIST

Fetches statistics about rules and rule sets.

ADDRESS AOF LISTCOMP

Fetches statistics about compiled rules and rule sets.

ADDRESS AOF LISTINST

Fetches statistics about enabled rules and rule sets, regardless of whether the rules reside on DASD. Returns information about dynamic rules.

ADDRESS AOF LISTSRC

Lists the source code of an enabled rule.

ADDRESS AOF RESETAUTO

Prevents a rule or rule-set from being automatically enabled at CA OPS/MVS startup.

ADDRESS AOF SETAUTO

Automatically enables a rule or rule-set at the next CA OPS/MVS startup.

ADDRESS AOF SUBSYS

Sets the z/OS SSID of the CA OPS/MVS product that processes subsequent ADDRESS AOF commands.

ADDRESS AP commands

Allows CA OPS/MVS to communicate with an MSF-connected Unicenter Automation Point system.

ADDRESS AP NMFIND

Invokes Notification Manager on an MSF-connected Unicenter Automation Point system.

ADDRESS AP PPQ

Writes one or more items to a specified queue on an MSF-connected Unicenter Automation Point system.

ADDRESS AP REXX

Sends a command to an MSF-controlled Unicenter Automation Point system to execute a REXX EXEC.

ADDRESS EPI commands

Allow OPS/REXX programs to define and operate virtual terminals that interact with VTAM applications through the EPI.

ADDRESS EPI BIND

Temporarily dedicates a VTAM application session to a specific CA OPS/MVS TSO user, CLIST, or REXX EXEC.

ADDRESS EPI CHANGE

Changes an existing definition of a virtual terminal.

ADDRESS EPI DEBUG

Turns on or off VTAM exit debugging.

ADDRESS EPI DEFINE

Defines a virtual terminal to the EPI.

ADDRESS EPI DELETE

Deletes one or more virtual terminal definitions from the EI.

ADDRESS EPI DEQ

Releases ownership of the virtual terminal.

ADDRESS EPI DISABLE

Disables one or more virtual terminals.

ADDRESS EPI ENABLE

Activates a virtual terminal.

ADDRESS EPI ENQ

Enqueues a virtual terminal or tests for terminal ownership.

ADDRESS EPI GETSCRN

Fetches the current screen image from a CA OPS/MVS -monitored VTAM session and displays it on the terminal.

ADDRESS EPI HELP

Returns a list of valid EPI host commands to the external data queue.

ADDRESS EPI INQINPUT

Checks to see whether the virtual keyboard can accept input.

ADDRESS EPI LIST

Displays the status of one or more virtual terminals.

ADDRESS EPI LOGOFF

Logs off a virtual terminal from an external product.

ADDRESS EPI LOGON

Logs a virtual terminal on to an external product.

ADDRESS EPI MSGID

Determines whether output lines that all EPI host commands return to the EPI external data queue have a prefix.

ADDRESS EPI MVCURSOR

Moves the cursor to the named row and column position on the virtual terminal screen.

ADDRESS EPI PEEK

Displays the contents of the virtual terminal screen buffer in hexadecimal format.

ADDRESS EPI POKE

Modifies the contents of the virtual terminal screen buffer.

ADDRESS EPI RDCURSOR

Returns the cursor position, which is specified by row and column number.

ADDRESS EPI RDSCREEN

Returns the entire virtual terminal screen as a set of output lines

ADDRESS EPI RDSCRROW

Returns a specified row as one output line.

ADDRESS EPI SESSCMD

Enables you to issue a command to a VTAM application session.

ADDRESS EPI SETMODEL

Sets the model number for the virtual terminal.

ADDRESS EPI SETTERM

Sets the current virtual terminal.

ADDRESS EPI SETUNAME

Sets the user-defined name for a virtual terminal.

ADDRESS EPI SUBATTR

Affects how the RDSCREEN and RDSCRROW commands return host attribute characters.

ADDRESS EPI SUBSYS

Directs EPI host commands from the current OPS/REXX program to an EPI other than the one using the default z/OS subsystem ID, which is OPSS.

ADDRESS EPI SUBUNPT

Affects how the RDSCREEN and RDSCRROW commands return unprintable characters.

ADDRESS EPI TIMEOUT

Determines how long EPI commands wait before timing out.

ADDRESS EPI TRACE

Turns tracing of terminal activity on and off.

ADDRESS EPI TRIM

Turns on and off trimming.

ADDRESS EPI TYPE

Simulates typing at a 3278 keyboard.

ADDRESS EPI TYPESEC

Simulates typing at a 3278 keyboard without writing the typed text to OPSLOG.

ADDRESS EPI TYPETEST

Test-runs a TYPE command.

ADDRESS EPI UNBIND

Reverses the effect of a BIND command and releases the indicated session for access by other sources.

ADDRESS EPI WAIT

Delays processing of a request rule or REXX program by a number of seconds.

ADDRESS EPI WAITTOD

Delays processing of a request rule or REXX program until a specific time of day is reached.

ADDRESS ISPEXEC

Sends commands to ISPF, which must be active to receive them.

ADDRESS LINKMVS

Enable an OPS/REXX program to invoke user-written and standard z/OS utility programs that support only the standard z/OS batch parameter list.

ADDRESS LINKPGM

Enable an OPS/REXX program to invoke user-written and standard z/OS utility programs that support a standard z/OS parameter list.

ADDRESS LXCON LXCMD | VMCMD

Sends VM or Linux commands to the USS server for execution by the Linux Connector started task.

ADDRESS LXCON LIST

Displays properties of VM or Linux systems that are connected to the Linux Connector started task.

ADDRESS MESSAGE

Enable you to debug host commands in rules or REXX programs without actually issuing any live commands.

ADDRESS MIM ENV

Contains environment information about the MIM driver.

ADDRESS MIM QNAME

Contains statistics and status of the system ENQ/DEQ and RESERVE/RELEASE activity.

ADDRESS MIM TAPE

Contains status information relating to the system tape activity.

ADDRESS NETMAN

Enables you to open, update, or close records in CA Netman.

ADDRESS NETMASTR

Creates alerts on the CA NetMaster NM for SNA Alert Monitor screen.

ADDRESS OPER

Permits you to issue operator commands (z/OS, JES, MVS/JES3, and VM) from an OPS/REXX program or AOF rule.

ADDRESS OPSCTL commands

Control the CA OPS/MVS COF, ECF, OSF, and MSF components.

ADDRESS OPSCTL COF ACTIVATE

Adds transient data queue names to a destination list or lists.

ADDRESS OPSCTL COF DEACTIVATE

Deletes transient data queue names from a destination list or lists.

ADDRESS OPSCTL COF DEFINE

Creates a list of CICS transient data queue names.

ADDRESS OPSCTL COF DELETE

Deletes one or more defined or default transient data queue name lists.

ADDRESS OPSCTL COF LIST

Displays the contents of a transient data destination list or lists.

ADDRESS OPSCTL ECF LIST

Lists information about console users that are logged on to the ECF.

ADDRESS OPSCTL MSF ACTIVATE

Activates a session or sessions.

ADDRESS OPSCTL MSF DEACTIVATE

Deactivates a session or sessions.

ADDRESS OPSCTL MSF DEFAULT

Specifies a default system name and system wait value for the currently executing REXX program or rule.

ADDRESS OPSCTL MSF DEFINE

Defines the VTAM LU-LU sessions that the MSF establishes and maintains with copies of CA OPS/MVS running on other systems.

ADDRESS OPSCTL MSF DELETE

Deletes definitions for either the specified system or for all MSF resources.

ADDRESS OPSCTL MSF LIST

Returns information about defined MSF sessions to the external data queue.

ADDRESS OPSCTL MSF START

Starts the MSF on the local system.

ADDRESS OPSCTL MSF STOP

Causes CA OPS/MVS to end its sessions with all other copies of the MSF on other systems.

ADDRESS OPSCTL OPSLOG DEFINE

Define a new OPSLOG for future use.

ADDRESS OPSCTL OPSLOG ACTIVATE

Activate a defined OPSLOG entry

ADDRESS OPSCTL OPSLOG SETLIVE

Make an active OPSLOG the live OPSLOG.

ADDRESS OPSCTL OPSLOG RESET

Empty an active (not live) OPSLOG.

ADDRESS OPSCTL OPSLOG LIST

List all of the OPSLOGs

ADDRESS OPSCTL OPSLOG DEACTIVATE

Deactivate an active OPSLOG

ADDRESS OPSCTL OPSLOG DELETE

Delete an OPSLOG definition

ADDRESS OPSCTL OSF EXECSTATS

Returns information about the performance of the OSF to the external data queue.

ADDRESS OPSCTL OSF LIST

Returns information about active OSF servers to the external data queue.

ADDRESS OPSCTL OSF QUEUES

Displays the status and history of the OSF server queue.

ADDRESS OPSCTL OSF RESETQ

All pending commands waiting on the OSF execute queue are immediately discarded.

ADDRESS OPSCTL OSF STOP

Stops a server.

ADDRESS OPSDYNAM commands

Allow you to issue the ADDRESS OPSDYNAM commands for dynamic allocation, concatenation, deconcatenation, and information retrieval functions.

ADDRESS OPSDYNAM ALLOCATE

Allocates the following information to a z/OS ddname with selected attributes:

- An existing or new data set
- A list of existing data sets
- A dummy data set
- A subsystem data set

ADDRESS OPSDYNAM CONCAT

Either permanently or temporarily concatenates a list of ddnames to the first ddname specified.

ADDRESS OPSDYNAM DECONCAT

Deconcatenates a nonpermanent, existing concatenation back to the originally allocated ddnames of the concatenated data sets.

ADDRESS OPSDYNAM FREE

Frees a list of ddnames, data set names, or both.

ADDRESS OPSDYNAM INFO

Returns the data set name, ddname, and data set organization of a currently allocated data set or ddname.

ADDRESS OSF

Routes commands to CA OPS/MVS OSF TSO server address spaces.

ADDRESS OSFTSL

Routes commands to CA OPS/MVS OSF TSL server address spaces.

ADDRESS OSFTSP

Routes commands to CA OPS/MVS OSF TSP server address spaces.

ADDRESS SERVDESK

Creates new requests on CA Service Desk.

ADDRESS SOF

Interfaces with the IBM I/O subsystem to provide advanced management capabilities for devices that are attached through the ESCON directors and FICON switches.

ADDRESS SQL

Enables you to create and maintain relational tables.

ADDRESS SYSVIEWE

Sends commands to CA SYSVIEW.

ADDRESS TSO

Routes commands to TSO.

ADDRESS USS commands

Sends UNIX or CA Common Services (CCS) for z/OS API commands to USS servers.

ADDRESS USS DOM

Removes a message from the held message area of the CA Event Manager console display.

ADDRESS USS CMD

Issues a USS command or any internal command that is executed by the CA Event Manager.

ADDRESS USS PING

Tests whether the CA Event Manager is active and reachable on a specified node name.

ADDRESS USS REPLY

Replies to an outstanding WTOR in the CA Event Manager.

ADDRESS USS USSCMD

Issues any UNIX command that can be executed from a shell environment.

ADDRESS USS WTO

Issues a message to the CA Event Manager.

ADDRESS USS WTOR

Issues a message to and then waits for a reply from the CA Event Manager.

ADDRESS WTO

Issues singleline and multiline WTO messages.

Host Command Defined

A *host command* is a command that an OPS/REXX program or an AOF rule sends to a non-REXX environment for execution. You use an ADDRESS instruction in a rule or OPS/REXX program to pass the command to the host. To find out which of the host environments are supported by each rule type, see the *AOF Rules User Guide*.

Host Environments

OPS/REXX supports these host environments:

- ADDRESS AOF instructions send commands to the CA OPS/MVS Automated Operations Facility.
- ADDRESS AP instructions allow you to send commands to Unicenter Automation Point.
- ADDRESS EPI instructions send commands to VTAM applications accessed by the CA OPS/MVS EPI facility.
- ADDRESS ISPEXEC instructions issue commands controlling ISPF dialogs.
- ADDRESS LINKMVS instructions enable an OPS/REXX program to invoke user-written and standard z/OS utility programs that support only the standard z/OS batch parameter list. Rules cannot use the LINKMVS environment. Invoke this environment without running under the TSO TMP.

- ADDRESS LINKPGM instructions enable an OPS/REXX program to invoke user-written and standard z/OS utility programs that support a standard z/OS parameter list. Rules cannot use the LINKPGM environment. You can invoke this environment without running under the TSO TMP.
- ADDRESS LXCON instructions enable an OPS/REXX program to issue commands to VM and Linux systems that are connected to the local system by the Linux Connector component.
- ADDRESS MIM interfaces with the MIM API to retrieve environment, qname, and tape information that CA MIM collects and maintains.
- ADDRESS MESSAGE instructions enable you to debug host commands in rules or REXX programs without actually issuing any live commands.
- ADDRESS NETMAN instructions enable you to open, update, or close records in CA Netman.
- ADDRESS NETMASTR instructions enable you to create alerts on the CA NetMaster NM for SNA Alert Monitor screen.
- ADDRESS OPER instructions issue operator commands (system, JES, or VM/CP commands).
- ADDRESS OPSCTL instructions issue commands to control CA OPS/MVS facilities.
- ADDRESS OPSDYNAM instructions provide dynamic allocation, unallocation, concatenation, deconcatenation, and information retrieval functions for data sets in the current address space.
- ADDRESS OSF instructions enable REXX programs and rules to issue TSO commands and CLIST commands to an OSF TSO server.

Note: Use the ADDRESS OSF environment instead of the ADDRESS TSO environment, except in request rules. This request makes certain that the command is sent to an OSF TSO server for execution. Using ADDRESS TSO, the environment to which the command is sent is dependent upon the environment in which ADDRESS TSO is executing.

- ADDRESS OSFTSL instructions enable REXX programs and rules to issue TSO commands and CLIST commands to an OSF TSL server. TSL servers are intended for work that runs for relatively long periods of time.
- ADDRESS OSFTSP instructions enable REXX programs and rules to issue TSO commands and CLIST commands to an OSF TSP server. TSP servers are intended for high priority work.
- ADDRESS SERVDESK instructions enable you to create new requests using CA Service Desk.
- ADDRESS SOF interfaces with the IBM I/O subsystem to provide advanced management capabilities for devices that are attached through the ESCON directors and FICON switches.

- ADDRESS SQL instructions issue SQL commands to create, modify, and delete the relational tables where the CA OPS/MVS Relational Data Framework stores information about your system. For detailed descriptions of ADDRESS SQL instructions and their syntax, see the chapter “Relational Data Framework Reference.”
- ADDRESS SYSVIEWE instructions issue commands directly to CA SYSVIEW and return the results of the commands. Rules cannot use the SYSVIEWE Command.
- ADDRESS TSO instructions issue TSO commands and invoke CLISTs.
- ADDRESS USS instructions send UNIX or CCS for z/OS API commands to USS servers.
- ADDRESS WTO instructions issue WTO messages synchronously.

OPS/REXX always issues host commands to the host environment named in the last ADDRESS instruction that executed. For instance, in the following example, OPS/REXX issues the D A,L command to the operating system and the LISTBC command to TSO:

```
address OPER  
"D A,L"  
address TSO  
"LISTBC"
```

External Host Environments

OPS/REXX programs, but not AOF rules, can send host commands to host environments outside the CA OPS/MVS product. When an ADDRESS statement references a host environment that the CA OPS/MVS product does not recognize as one of its built-in environments (those described in the previous section), CA OPS/MVS passes the request to the CA GSS software. CA GSS is part of CCS for z/OS.

You must have installed and started the CA GSS address space and any CA product (such as CA Jobtrac or CA Scheduler) to which you are passing host commands. For more information, see the CCS for z/OS documentation.

Output Returned from External Environments

CA GSS returns output from a host command forwarded by CA OPS/MVS to the external data queue of the OPS/REXX program that issued the command. The RC variable receives one of these values:

- -20, if CA GSS is not active or it fails
- -3, if CA GSS also does not recognize the named host environment
- The return code from the host command, in all other cases

How Host Command Delimiters Work

To ensure that OPS/REXX parses host commands properly, use keywords and quotation marks correctly in host commands and understand how OPS/REXX interprets these commands.

To use keywords and quotation marks, follow these guidelines:

- Enclose the complete text of a host command in double quotes. If that command contains a text string, enclose the string in single quotes.

Example: Text string enclosed in single quotes

```
ADDRESS WTO "TEXT(VTAM IS DOWN)"
```

- Enclose everything except for variable names in double quotation marks.

Example: The first statement sets an OPS/REXX variable called JOBNAME. The command text being sent to TSO through the OPSCMD command references this variable, so two sets of quotes delimit the text outside the variable name.

```
JOBNAME = MSG.JOBNAME  
ADDRESS TSO  
"OPSCMD COM($C "jobname")"
```

- When the text of a host command is too large to fit on one line:
 - Use the || symbol and a comma to indicate that the command text breaks and continues onto the next line.
 - Use double quotes to indicate the beginning and end of each piece of the host command.

Example: Host command text that spans two or more lines

```
ADDRESS TSO  
"OPSWTO TEXT("This message is so"  
"long that I couldn't pos"  
"sibly fit it on one line")"
```

Responses from Host Commands

This section discusses external data queues, rules and host command responses, OPS/REXX host command output, return codes, and security considerations for host commands.

External Data Queue

OPS/REXX places responses from host commands in the external data queue. You can read responses line by line from this queue using the PULL instruction. To find how many lines remain in the external data queue, use the QUEUED function.

Issue Commands from Rules without Host Command Responses

Some types of AOF rules do not process responses from host commands. For example, message rules run synchronously in the z/OS subsystem exit, so they do not wait for commands to generate output.

To issue a command from a rule that does not process command responses:

1. Write the command as an asynchronous command procedure (using either TSO/E REXX or OPS/REXX).
2. Write your rule so that it runs the procedure in a server address space.

Example: Issue Commands from Rules

The following OPS/REXX program issues a 'D A,L' operator command and displays the responses at the terminal of the user:

```
address OPER
"D A,L"      /* responses placed in external data queue */
do while queued() > 0
  parse pull x /* retrieve a line from the queue */
  say x       /* display it at the terminal */
end
```

If a rule needs to issue a command, and then act upon the command response, make the rule a request rule and invoke it from a different command procedure.

How OPS/REXX Gets Host Command Output

How OPS/REXX obtains the messages that host commands generate depends on the environment to which the commands were addressed:

Environment	How OPS/REXX Fetches the Response
AOF	Host commands sent through the ADDRESS AOF instruction are directives to the internal CA OPS/MVS AOF control facilities. For descriptions of these commands and the output they send to the external data queue, see ADDRESS AOF Commands in this chapter.
AP	The ADDRESS AP instruction routes commands to MSF-connected CA Automation Point systems. CA Automation Point uses CAICCI as a communication protocol and acts as a limited-function remote MSF system. For more information on using ADDRESS AP, see ADDRESS AP Commands in this chapter.

Environment	How OPS/REXX Fetches the Response
EPI	The ADDRESS EPI instruction routes commands to the CA OPS/MVS External Product Interface. These commands allow users to control and interact with VTAM virtual terminals. Output from EPI host commands is returned in the external data queue. For more information on using ADDRESS EPI, see ADDRESS EPI Commands in this chapter.
ISPEXEC	The ADDRESS ISPEXEC instruction sends commands to ISPF, which must be active to receive them.
LINKMVS	For information about this host environment, see the IBM documentation.
LINKPGM	For information about this host environment, see the IBM documentation.
MESSAGE	MESSAGE is the default host environment for AOFTTEST. An ADDRESS MESSAGE instruction displays commands as messages. The text of the command becomes the text of a WTO message. You can use ADDRESS MESSAGE during debugging to inhibit live commands. For more information, see the descriptions of the CA OPS/MVS parameters AOFDEFAULTADDRESS and REXXDEFAULTADDRESS in the <i>Parameter Reference</i> .
OSF	The ADDRESS OSF instruction routes commands to CA OPS/MVS OSF TSO server address spaces. Output from TSO commands directed to the OSF host environment does not return to the OPS/REXX program or to rules. Host commands sent to a server through ADDRESS OSF can contain no more than 256 bytes.
OSFTSL	The ADDRESS OSFTSL instruction routes commands to CA OPS/MVS OSF TSL server address spaces. Output from TSO commands directed to the OSFTSL host environment does not return to the OPS/REXX program or to rules. Host commands sent to an OSF TSL server through ADDRESS OSFTSL can contain no more than 256 bytes.
OSFTSP	The ADDRESS OSFTSP instruction routes commands to CA OPS/MVS OSF TSP server address spaces. Output from TSO commands directed to the OSFTSP host environment does not return to the OPS/REXX program or to rules. Host commands sent to an OSF TSP server through ADDRESS OSFTSP can contain no more than 256 bytes.

Return Codes from Host Commands

When a host command executes, OPS/REXX sets the variable RC to a value indicating whether the command executed without *errors* or *failures*. In general, *errors* are problems that the calling program can probably recover from, and *failures* are unrecoverable errors. When an abend occurs as a host command executes, OPS/REXX considers the abend to be a failure.

- If the value of RC is zero, the host command detected no error or failure.
- If RC has a positive value, the host command detected an error.
- If the RC value is negative, the host command detected a failure.

Security Considerations for Host Commands

Host commands issued from AOF rules are considered authorized from a CA OPS/MVS product perspective and are not subject to CA OPS/MVS security processing. When used in an OPS/REXX program in an insecure environment (for example, the TSO address space of a user or in a server), the use of any of these host command environments results in the creation of CA OPS/MVS product security events:

- AOF
- EPI
- OPER
- OPSCTL
- SQL
- WTO

Note: For related information, see the discussion of security rules and the OPUSEX security exit in the *User Guide*. Additional information about the OPUSEX exit appears in the chapter “Technical Notes” in the *Administrator Guide*.

REXX Instructions Against Host Commands

In OPS/REXX, *instructions* provide logic structures (IF ... THEN ... ELSE ..., DO WHILE ... END) and define variables, while *host commands* allow you to issue actions (TSO commands, ISPEXEC commands, z/OS operator commands, JES commands, and VM/CP commands). In general, when OPS/REXX encounters an instruction consisting solely of an *expression* (a string by itself is a valid expression), it assumes that the instruction is a host command.

OPS/REXX does not interpret the syntax of host commands. To OPS/REXX, a host command is a meaningless string of characters. For instance, if you make a coding mistake such as placing an OPS/REXX function on a line by itself without a variable and an equal sign, OPS/REXX will probably interpret it as a host command and send it to the environment specified in the last ADDRESS statement. Conversely, instructions have a specific meaning to OPS/REXX.

Examples: OPS/REXX Instructions and Host Commands

OPS/REXX program demonstrating instructions and host commands:

```
if today = "Friday" then
  say "TGF !!"
else
  do
    address OPER "D A,L"
    address TSO "LISTBC"
  end
```

In the above example, the strings D A,L and LISTBC are host commands. All other constructs are OPS/REXX instructions. The specific instructions are:

- if ... then ... else ...
- say ...
- do ... end
- address ...

The data structures shown are:

- Variables (today)
- String constants ("Friday", "TGF !!", "OPER", "D A,L", "TSO", and "LISTBC")
- Expressions (today = "Friday")

How Rules Process Host Commands

This section lists the types of host commands and explains how rule sections and types of rules process them.

ADDRESS AOF

In the)INIT and)TERM sections of all rules and the)PROC sections of all rules *except* command rules:

- OPS/REXX sends commands to the AOF for execution.
- Command output cannot be retrieved from the external data queue.
- OPS/REXX routes command output to the destination specified with the AOFDEST parameter.

In the)PROC section of command rules:

- OPS/REXX sends commands to the AOF for execution.
- Command output cannot be retrieved from the external data queue.
- Command output goes to the console where the command was issued.

ADDRESS EPI

In the)INIT and)TERM sections of all rules and the)PROC sections of all rules *except* request rules ADDRESS EPI commands are subject to NOWAIT considerations.

In the)PROC section of request rules command output can be retrieved from the external data queue.

ADDRESS ISPEXEC

In the)INIT and)TERM sections of all rules and the)PROC sections of all rules *except* request rules ADDRESS ISPEXEC commands are not supported.

In the)PROC section of request rules:

- ADDRESS IPSEXEC commands can execute synchronously if the OPSREQ command is issued from in an ISPF environment.
- In a server, the rules for batch execution of ISPF apply. For more information, see the IBM documentation.

ADDRESS OPER

In the)INIT and)TERM sections of all rules:

- ADDRESS OPER instructions can issue operator commands.
- Command output cannot be retrieved from the external data queue.
- OPS/REXX routes command output to either the z/OS master console or the JES3 master console.

In the)PROC section of command, DOM, EOM, global variable, message, and OMEGAMON rules:

- ADDRESS OPER instructions can issue operator commands.
- OPS/REXX cannot retrieve command output from the external data queue.
- OPS/REXX routes command output to the z/OS master console.

In the)PROC section of request and time-of-day rules:

- ADDRESS OPER instructions can issue operator commands.
- OPS/REXX can retrieve command output from the external data queue.

ADDRESS OPSCTL

You can use ADDRESS OPSCTL instructions in any section of any type of rule, but you cannot always retrieve the command output.

ADDRESS SOF

You can use ADDRESS SOF to communicate between the ADDRESS SOF host command and the CA SOF server through the CA CCI interface.

ADDRESS SQL

You can use ADDRESS SQL instructions in any section of any type of rule.

ADDRESS WTO

You can use ADDRESS WTO instructions in any section of any type of rule.

In any type of rule except for time-of-day (TOD) and request (REQ) rules, CA OPS/MVS converts to an OPSWTO command processor any ADDRESS WTO instruction that contains one of these keywords: REPLY, WAIT, or DELAY. CA OPS/MVS performs this conversion because it does not delay rule processing.

When a TOD or REQ rule contains an ADDRESS WTO command that specifies the DELAY, WAIT, or REPLY keywords, CA OPS/MVS permits the ADDRESS WTO command to execute synchronously. The TOD or REQ rule waits until the ADDRESS WTO command completes before it resumes processing.

Note: The use of the REPLY, WAIT, or DELAY keyword in a TOD rule may delay the execution of other TOD rules.

ADDRESS ALERTMON Host Environment

See [ADDRESS NETMASTR alert function](#) (see page 203) description.

ADDRESS AOF Commands

The ADDRESS AOF host environment sends commands to the Automated Operations Facility (AOF). The AOF uses rules (compact OPS/REXX programs) to respond automatically when events occur on your system, relieving system operators from having to identify each event and react to it. For a description of AOF rules and how to use them, see the *AOF Rules User Guide*.

AOF rules and OPS/REXX programs can issue ADDRESS AOF commands. All of these commands place some information in the external data queue, and all set the RC variable to the return code from the task that the command performs.

Some ADDRESS AOF commands require you to specify a rule set and rule. To do so, you need to know the current values of three parameters:

- **RULEALTFIX** specifies the highest level (first) qualifier of the name of the alternate rules data set.
- **RULEPREFIX** specifies the highest level qualifier of the current rules data set.
- **RULESUFFIX** specifies the lowest level (last) qualifier of the current rules data set.

Note: For descriptions of these parameters, see the *Parameter Reference*.

ADDRESS AOF Command Format

Most ADDRESS AOF commands act on a ruleset or a rule in a rule set, and conform to the following syntax. There are some exceptions; complete syntax and descriptions of all ADDRESS AOF commands appear in the sections that follow.

```
ADDRESS AOF "commandverb keywords"  
[ruleset{ruleset.rule}]  
[SYSTEM(ALL|EXT){sysnames}]
```

In the syntax shown:

commandverb

Specifies one of the following AOF commands:

- COMPILE
- DELCOM
- DISABLE
- ENABLE
- INDEX
- LIST
- LISTCOMP
- LISTINST
- LISTSRC
- RESETAUTO
- SETAUTO
- SUBSYS

ruleset

(Optional) Specifies the middle qualifier of the current rules data set-the qualifier that lies between the qualifiers specified by the RULEPREFIX or RULEALTFIX and RULESUFFIX parameters.

For example, if the name of the current rules data set is SYS1.OPS.IPL.RULES, the RULEPREFIX is SYS1.OPS, the RULESUFFIX is RULES, and the *ruleset* value is IPL.

rule

(Optional) Indicates the value is a member of the current rules data set.

SYSTEM

(Optional) Enables you to perform cross-system AOF operations. Specify one of these values:

ALL

Routes the AOF command to all active MSF-defined systems, including the local system.

EXT

Routes the AOF command to all remote, active MSF-defined systems.

sysnames

Routes the AOF command to the specified systems. You may specify from one to eight system names as the value of *sysnames*.

SYSWAIT

(Optional) Specifies the number of seconds CA OPS/MVS waits for output from a remote system. You may specify a value between 1 and 600 seconds.

Note: It is intentional that this limit is higher than the limit that can be set by ADDRESS OPSCTL MSF DEFAULT.

Do not specify a value for SYSWAIT if you specify the SYSTEM(ALL), SYSTEM(EXT), or NOOUTPUT keywords.

OUTPUT|NOOUTPUT

Determines whether the command returns output to the external data queue. Specify OUTPUT to have output returned; otherwise, specify NOOUTPUT.

NOOUTPUT is implied when:

- The ADDRESS AOF command is issued in an AOF rule.
- You specify more than one system name as the value of the SYSTEM(*sysnames*) keyword.
- You specify SYSTEM(ALL) or SYSTEM(EXT).

Important! The OUTPUT and NOOUTPUT keywords are valid only when they are used in conjunction with the SYSTEM keyword. This restriction applies to all ADDRESS AOF host commands.

How ADDRESS OPSCTL Routes AOF Commands to Other Systems

You can use the ADDRESS OPSCTL host command environment to send an AOF command to a remote system.

Example: ADDRESS OPSCTL with ADDRESS AOF

This example uses ADDRESS OPSCTL in conjunction with ADDRESS AOF:

```
ADDRESS AOF  
SUBSYS "OPSF"  
ADDRESS OPSCTL  
"MSF DEFAULT SYSTEM(SYS1) SYSWAIT(10)"
```

If you use ADDRESS OPSCTL in this way, specify only one system name as the value for the SYSTEM keyword.

More information

[ADDRESS OPSCTL Host Environment](#) (see page 245)

AOF Commands and the Test Facility

Avoid issuing AOF commands from within the test facility; doing so can cause undesirable side effects on your production system.

Command Execution When the Product Is Down

When CA OPS/MVS is down:

- ADDRESS AOF commands have no effect.
- Issuing an AOF command sets the RC variable to 20.
- The external data queue contains the OPS4300I message, which reports that CA OPS/MVS could not execute the command.

OPSVIEW options 2.1, 2.2, 4.5.1, and 4.5.2, which all pertain to the AOF, are written in OPS/REXX. These options provide many examples of how to use ADDRESS AOF and ADDRESS ISPEXEC instructions.

AOF COMPILE—Create Compiled Rules

The COMPILE command creates compiled versions of rules. This command operates only if you set valid values for the CA OPS/MVS AOFPRECOMPILED and AOFPRECOMPILEDDSN parameters.

When you issue the COMPILE command CA OPS/MVS saves the compiled rule, rule set, or rule sets in the data set specified by the AOFPRECOMPILEDDSN parameter.

The advantage to compiling rules in advance is that it enables them to start automating system operations tasks immediately when the CA OPS/MVS product starts up.

This command has the following format:

```
ADDRESS AOF "COMPILE keywords"  
[ruleset]  
[ruleset rule]  
[SYSTEM{ALL|EXT}sysnames]  
[SYSWAIT(seconds)]  
[OUTPUT|NOOUTPUT]
```

Note: You cannot use the ISPF Delete PDS Member utility to delete compiled rules.

Examples: COMPILE Command

- To compile all rules in all rule sets, issue this command:

```
ADDRESS AOF "COMPILE"
```
- To compile all rules in the named rule set, issue this command:

```
ADDRESS AOF "COMPILE ruleset"
```
- To compile a single rule, issue this command:

```
ADDRESS AOF "COMPILE ruleset rule"
```

More information:

[ADDRESS AOF Command Format](#) (see page 43)

AOF DELCOMP—Delete Compiled Rules

The DELCOMP command deletes compiled versions of rules. This command operates only if you set valid values for the CA OPS/MVS AOFPRECOMPILED and AOFPRECOMPILEDDSN parameters.

When you issue the DELCOMP command CA OPS/MVS deletes the compiled rule, rule set, or rule sets in the data set specified by the AOFPRECOMPILEDDSN parameter.

This command has the following format:

```
ADDRESS AOF "DELCOMP keywords"  
[ruleset[rulesetrule]]  
[SYSTEM(ALL|EXT|sysnames)]  
[SYSWAIT(seconds)]  
[OUTPUT|NOOUTPUT]
```

Note: You cannot use the ISPF Delete PDS Member utility to delete compiled rules.

Examples: DELCOMP command

- To delete all compiled rules in all compiled rule sets, issue this command:

```
ADDRESS AOF "DELCOMP"
```
- To delete all compiled rules in the named rule set, issue this command:

```
ADDRESS AOF "DELCOMP ruleset"
```
- To delete a single compiled rule, issue this command:

```
ADDRESS AOF "DELCOMP rulesetrule"
```

More information:

[ADDRESS AOF Command Format](#) (see page 43)

AOF DISABLE—Disable Rules

The DISABLE command makes rules ineligible to run and removes them from storage.

Issuing the DISABLE command sets the RC variable to zero. Unless the NOOUTPUT keyword is explicitly or implicitly specified, the external data queue contains the results of any SAY statements in the)TERM section of the rule. These results appear as OPS1000I messages.

This command has the following format:

```
ADDRESS AOF "DISABLE keywords"  
[ruleset][ruleset.rule][*DYNAMIC.rule]  
[SYSTEM(ALL|EXT)sysnames]  
[SYSWAIT(seconds)]  
[OUTPUT|NOOUTPUT]
```

Examples: DISABLE Command

- To disable all enabled rules (for example, at CA OPS/MVS shutdown), issue this command:

```
ADDRESS AOF  
"DISABLE"
```

- To disable all enabled rules in a ruleset, issue this command:

```
ADDRESS AOF  
"DISABLE ruleset"
```

- To disable a single rule in a ruleset, issue this command:

```
ADDRESS AOF  
"DISABLE ruleset.rule"
```

- To disable a dynamic rule, issue this command:

```
ADDRESS AOF  
"DISABLE *DYNAMIC.rule"
```

The original location of the source text of the rule (either the external data queue or stem variables) is irrelevant.

Dynamic Rules Support for DISABLE

Specifying a rule set name of **DYNAMIC* on your **DISABLE** command indicates that the rules you want to disable are dynamic rules.

Keep these guidelines in mind when working with dynamic rules:

- If your **DISABLE** command is intended for dynamic rules, do not specify the **SYSTEM** keyword.
- Disabling dynamic rules requires the specification of a rule name.
- Do not use the **DYNAMIC* rule set name for non-dynamic rules.

More information:

[ADDRESS AOF Command Format](#) (see page 43)

[ADDRESS AOF Return Codes](#) (see page 71)

AOF ENABLE—Enable Rules

This command makes rules eligible to run, causing the **)INIT** sections of those rules to execute.

The **ENABLE** command sets the **RC** variable to zero. Unless the **NOOUTPUT** keyword has been explicitly or implicitly specified, the external data queue contains the results of any **SAY** statements in the **)INIT** section of the rule. These results appear as **OPS1000I** messages.

This command has the following format:

```
ADDRESS AOF "ENABLE keywords"  
[ruleset|ruleset.rule|*DYNAMIC.rule[STEM(stemname)]]  
[SYSTEM{ALL|EXT|sysnames}]  
[SYSWAIT(seconds)]  
[OUTPUT|NOOUTPUT]
```

Note: If you specify **DYNAMIC.rule*, you may also specify the **STEM(stemname)** keyword.

STEM

Specifies the stem name of the variables.

Use STEM when the rules that you want to enable are dynamic rules whose source text is contained in REXX stem variables.

The value of *stemname* can be up to 84 bytes long and does not need to end with a period. CA OPS/MVS recognizes the end of the source text of the rule when it encounters a null stem variable value or a stem variable that is undefined or uninitialized. A rule that is constructed in stem variables can have a maximum length of 255 characters per stem variable. If you omit the STEM keyword, CA OPS/MVS retrieves the source text of the rule from the external data queue; this is the default. The maximum length of lines in the external data queue is 255.

Note: Non-dynamic rules (at least at the time they are enabled), have a one-to-one correspondence with a member in a rule set. Dynamic rules, in contrast, exist only in storage.

Examples: ENABLE Command

- To activate all rules that are set to be enabled automatically, issue this command:

```
ADDRESS AOF "ENABLE"
```
- To activate all rules in a rule set that are set to be enabled automatically, issue this command:

```
ADDRESS AOF "ENABLE ruleset"
```
- To enable a single rule regardless of its auto-enable status, issue this command:

```
ADDRESS AOF "ENABLE ruleset.rule"
```
- To route an ENABLE command to two MSF-defined systems named MVS1 and MVS4, issue this command:

```
ADDRESS AOF "ENABLE ruleset.rule"; "SYSTEM(MVS1,MVS4)"
```
- To enable and disable a dynamic rule whose source text is in the external data queue, use the following command or method. The rule issues a REXX SAY statement to the console once for every ten times that the rule executes. When the rule is disabled, it issues a REXX SAY statement that indicates the total number of times that the rule executed.

```
QUEUE "JCMD TESTCMD"
QUEUE "JNT"
QUEUE "entries = 0"
QUEUE "JPROC"
QUEUE "entries = entries + 1"
QUEUE "IF entries // 10 = 0 THEN"
QUEUE "SAY 'Rule TESTRULE has been entered 'entries' times'"
QUEUE "return 'accept' "
QUEUE "SAY 'Rule TESTRULE total entries = 'entries' "
ADDRESS AOF
"ENABLE *DYNAMIC.TESTRULE"
```

```

.
.
"DISABLE *DYNAMIC.TESTRULE"

■ To enable and disable a dynamic rule whose source text is in the stem variables
  whose names have a stem of STEMVAR:

STEMVAR1 = ")CMD TESTCMD"
STEMVAR2 = ")INT"
STEMVAR3 = "entries = 0"
STEMVAR4 = ")PROC"
STEMVAR5 = "entries = entries + 1"
STEMVAR6 = "F entries // 10 = 0
THEN"
STEMVAR7 = " SAY Rule TESTRULE has been entered 'entries' times"
STEMVAR8 = " return 'accept' "
STEMVAR9 = " SAY Rule TESTRULE total entries = 'entries' "
ADDRESS AOF
"ENABLE *DYNAMIC.TESTRULE
  STEM(STEMVAR)"
.
.
"DISABLE *DYNAMIC.TESTRULE"

```

More information:

[ADDRESS AOF Command Format](#) (see page 43)

[ADDRESS AOF Return Codes](#) (see page 71)

Dynamic Rules Support for ENABLE

Specifying a rule set name of *DYNAMIC on your ENABLE command indicates that the rules you want to enable are dynamic rules. The source text of dynamic rules can be constructed in either of these places:

- The REXX external data queue.
- A series of REXX stem variables.

Use the following guidelines when working with dynamic rules:

- The maximum size of a dynamic rule is one megabyte.
- If your ENABLE command is intended for dynamic rules, do not specify the SYSTEM or SYSWAIT keywords.
- Do not use the *DYNAMIC rule set name for non-dynamic rules.
- Do not use global variable stems with the STEM keyword.

AOF INDEX—Return Rules Data Set Information

This command allows an OPS/REXX program to find the rules CA OPS/MVS is currently using on your production system.

This command has the following format:

```
ADDRESS AOF "INDEX keywords"  
[SYSTEM(ALL|EXT|sysnames)]  
[SYSWAIT(seconds)]  
[OUTPUT|NOOUTPUT]
```

The INDEX command produces these results:

- The INDEX command returns information about the rules data set that CA OPS/MVS is using. It always returns a minimum of two words.
- The first word returned by INDEX contains *prefix.*.suffix*, where *prefix* is the current setting of the RULEPREFIX parameter, *suffix* is the current setting of the RULESUFFIX parameter, and *.** is a literal place holder. This word describes the data set names that CA OPS/MVS will search to find rules.
- The second word returned by INDEX is always SINGLE or MULTIPLE. If the parameter RULEALTFIX is set to a non-blank value, the word is MULTIPLE. If RULEPREFIX2 is specified, the word is MULTS, and a third word exists that is similar to the first word, but instead of RULEPREFIX, RULEPREFIX2 is used to create the prefix2.*.suffix string.

In most installations, RULEALTFIX and RULEPREFIX2 are unset, and the second word returned by INDEX is SINGLE.

- If the command executes successfully, the RC value is set to zero. Unless the NOOUTPUT keyword has been explicitly or implicitly specified, the external data queue contains the output string, described above.

More information:

[ADDRESS AOF Command Format](#) (see page 43)

[ADDRESS AOF Return Codes](#) (see page 71)

AOF LIST—Provide Rule and Rule Sets Statistics

This command fetches statistics about rules and rule sets.

The LIST command sets the RC variable to zero. Unless the NOOUTPUT keyword is implicitly specified, the external data queue contains messages providing rule or rule set statistics. The LIST and LIST *ruleset.** forms of the command produce multiple sets of statistics, one for each rule or rule set.

This command has the following format:

```
ADDRESS AOF "LIST keywords"  
[ruleset{ruleset:*|rulesetrule}=NOSTATS]  
[SYSTEM(sysname)]  
[SYSWAIT(seconds)]
```

Note: If the SYSTEM keyword is used for this command, it may contain only one system name. EXT and ALL refer to all external systems (more than one) and are therefore not allowed.

Examples: LIST Command

- To list all rulesets and their cumulative statistics, issue this command:

```
ADDRESS AOF  
"LIST"  
  
or
```

```
ADDRESS AOF  
"LIST =NOSTATS"
```

If you use the =NOSTATS keyword, CA OPS/MVS lists only rule set names. Omitting the statistics saves system overhead.

- To list all rulesets on system MVS1 and their cumulative statistics, issue this command:

```
ADDRESS AOF  
"LIST SYSTEM(MVS1)"
```

- To list the members in a single ruleset and its cumulative statistics, issue this command:

```
ADDRESS AOF  
"LIST ruleset"
```

- To list all rules in a rule set with statistics, issue this command:

```
ADDRESS AOF  
"LIST ruleset*"
```

- To list a single rule in a rule set with statistics, issue this command:

```
ADDRESS AOF  
"LIST rulesetrule"
```

More information:

[ADDRESS AOF Command Format](#) (see page 43)

Statistics for Rules

Unless NOOUTPUT is implicitly specified, issuing the LIST command for a rule inserts the following information into the OPS/REXX external data queue:

Word Number: 1

Length: 8

Contains the message ID.

Word Number: 2

Length: 8

Contains the member or rule name.

Word Number: 3

Length: 1

Contains the rule status of either E (Enabled) or D (Disabled).

Word Number: 4

Length: 1

Contains a flag indicating whether the rule has been auto-enabled: either Y (Yes) or N (No)

Word Number: 5

Length: 1

Contents reserved for CA OPS/MVS use.

Word Number: 6

Length: 3

Contains a code indicating the rule type:

API-Application Program Interface rule

ARM-Automatic Restart Management rule

CMD-Command rule

DOM-Delete-operator-message rule

EOJ-End-of-job rule

EOM-End-of-memory rule

EOS-End-of-step rule

GLV-Global variable rule

MSG-Message rule

OMG-OMEGAMON rule

REQ-Request rule

SCR-Screen rule

SEC-Security rule

TLM-Time limited event rule

TOD-Time-of-day rule

USS-UNIX System Services rule

Word Number: 7

Length: 5

Contains the version and modification level of the rule (*vv.mm*), or ***.*** if no PDF statistics were requested.

Word Number: 8

Length: 10

Contains the date when the rule was created (*yyyy/mm/dd*), or *** if no PDF statistics were requested.

Word Number: 9

Length: 10

Contains the date when the rule was last modified (*yyyy/mm/dd*), or *** if no PDF statistics were requested.

Word Number: 10

Length: 5

Contains the time when the rule was last modified (*hh:mm*), or *** if no PDF statistics were requested.

Word Number: 11

Length: 3

Contains the current number of lines in the rule, or *** if no PDF statistics were requested.

Word Number: 12

Length: 3

Contains the initial number of lines in the rule, or *** if no PDF statistics were requested.

Word Number: 13

Length: 3

Contains the number of lines in the rule that have modification numbers appended to them, or *** if no PDF statistics were requested.

Word Number: 14

Length: 8

Contains the TSO user ID of the last user who modified the rule, or * if no PDF statistics were requested.

Word Number: 15

Length: 10

Contains the date when the rule executed last (*yyyy/mm/dd*).

Word Number: 16

Length: 8

Contains the time when the rule executed last (*hh:mm:ss*).

Word Number: 17

Length: 10

Contains the date when the rule will execute next (*yyyy/mm/dd*); for TOD rules only.

Word Number: 18

Length: 8

Contains the time when the rule will execute next (*hh:mm:ss*); for TOD rules only.

Word Number: 19

Length: 11

Contains the number of times the rule has executed.

Word Number: 20

Length: 1

Contains the flag indicating whether the message appears in OPSLOG: either Y (Yes) or N (No); for MSG rules only.

AOF LISTCOMP—List Compiled Rule Statistics

This command fetches statistics about compiled rules and rule sets.

The LISTCOMP command sets the RC variable to zero. Unless the NOOUTPUT keyword is implicitly specified, the external data queue contains messages providing rule or rule set statistics. The LISTCOMP and LISTCOMP *ruleset.** forms of the command produce multiple sets of statistics, one for each rule or rule set.

This command has the following format:

```
ADDRESS AOF "LISTCOMP keywords"  
[ruleset{ruleset:*|rulesetrule}]  
[SYSTEM(sysname)]  
[SYSWAIT(seconds)]
```

Note: If the SYSTEM keyword is used for this command, it may contain only one system name. EXT and ALL refer to all external systems (more than one) and are therefore not allowed.

Examples: LISTCOMP command

- To list all compiled rule sets and their cumulative statistics, issue this command:

```
ADDRESS AOF  
"LISTCOMP"
```

- To list a single compiled rule set and its cumulative statistics, issue this command:

```
ADDRESS AOF  
"LISTCOMP ruleset"
```

- List all rules in a compiled rule set with statistics.

```
ADDRESS AOF  
"LISTCOMP ruleset:*"
```

- List a single compiled rule in a rule set with statistics.

```
ADDRESS AOF  
"LISTCOMP rulesetrule"
```

More information:

[ADDRESS AOF Command Format](#) (see page 43)

Statistics for Rules

Unless NOOUTPUT is implicitly specified, the LISTCOMP command inserts the following information into the OPS/REXX external data queue.

- For a LISTCOMP ruleset.rulename or LISTCOMP ruleset command:

Word Number: 1

Length: 8

Contains the message ID.

Word Number: 2

Length: 8

Contains the source rule name.

Word Number: 3

Length: 10

Contains the date when the rule was compiled (in *yyyy/mm/dd* format).

Word Number: 4

Length: 5

Contains the time when the rule was compiled (in *hh:mm* format).

For a LISTCOMP ruleset command, a record with the above data will be returned for each compiled rule that exists within the specified ruleset. Meaning, if the ruleset has 10 compiled rules, then 10 records will be returned to the EDQ.

- For a LISTCOMP command to request data on all precompiled rulesets:

Word Number: 1

Length: 8

Contains the message ID

Word Number: 2

Length: 8

Contains the ruleset name.

Word Number: 3

Length: 10

Contains the earliest compile date for a compiled rule within the ruleset (in *yyyy/mm/dd* format).

Word Number: 4

Length: 5

Contains the earliest compile time for a compiled rule within the ruleset (in *hh:mm* format).

Word Number: 5

Length: 10

Contains the most recent compile date for a compiled rule within the ruleset (in *yyyy/mm/dd* format).

Word Number: 6

Length: 5

Contains the most recent compile time for a compiled rule within the ruleset (in *hh:mm* format).

Word Number: 7

Length: 7

Contains the number of compiled rules that the ruleset contains.

More information:

[ADDRESS AOF Return Codes](#) (see page 71)

AOF LISTINST—List Enabled Rule Statistics

This command fetches statistics about enabled rules and rule sets, regardless of whether the rules are resident on DASD. Unlike the LIST command, the LISTINST command returns information about dynamic rules.

The LISTINST command sets the RC variable to zero. Unless the NOOUTPUT keyword is implicitly specified, the external data queue contains messages providing rule or rule set statistics. If a parameter error occurs, the RC variable is set to 12. A return code of 12 is accompanied by descriptive messages.

This command has the following format:

```
ADDRESS AOF "LISTINST keywords"  
[ruleset|ruleset.*|rulesetrule]  
[RULETYPE(type)]  
[SYSTEM(sysname)]  
[SYSWAIT(seconds)]
```

Note: If the SYSTEM keyword is used for this command, it may contain only one system name. EXT and ALL refer to all external systems (more than one) and are therefore not allowed.

Use the RULETYPE keyword to specify the type of rules about which you are requesting statistics. The value of *type* can be any of the following: ARM, CMD, DOM, EOJ, EOM, EOS, GLV, MSG, OMG, REQ, SCR, SEC, TLM, TOD, or USS. Specifying a value for the RULETYPE keyword is optional; if you let it default, the LISTINST command returns statistics for enabled rules of all types.

Examples: LISTINST command

- To list all rule sets that contain enabled rules, issue this command:

```
ADDRESS AOF  
"LISTINST"
```

- To list all rule sets on system MVS1 that contain enabled rules, issue this command:

```
ADDRESS AOF  
"LISTINST SYSTEM(MVS1)"
```

- To list the rule sets on system MV2 that contain enabled message rules, issue this command:

```
ADDRESS AOF  
"LISTINST RULETYPE(MSG) SYSTEM(MV2)"
```

- To list all enabled command rules in a rule set, issue this command:

```
ADDRESS AOF  
"LISTINST ruleset.* RULETYPE(CMD)"
```


More information:

[ADDRESS AOF Command Format](#) (see page 43)

Statistics for Rule Sets

CA OPS/MVS inserts this information into the OPS/REXX external data queue when you issue the LISTINST command in either of these ways:

- You include *ruleset.rule* in the command.
- You include *ruleset.** in the command.

Each line of output describes one rule set that meets the selection criteria specified on the LISTINST command.

Word Number	Length	Contents
1	8	The message ID, which is OPx3945I
2	8	The name of the ruleset
3	1	The enablement status of the rule set, which is always E
4	3	The type of enabled rules in the rule set. If the enabled rules in the rule set are all the same type, that type is returned (for example, CMD or OMG); if the rules are mixed, the value MIX is returned
5	6	A count of the rules in the rule set that are enabled
6	10	The most recent date on which a rule in the rule set was executed (yyyy/mm/dd)
7	8	The most recent time at which a rule in the rule set was executed (hh:mm:ss)
8	10	If any of the selected rules in the rule set are TOD rules, this word indicates the next date on which a TOD rule is to execute (yyyy/mm/dd); if none of the enabled rules in the rule set are TOD rules, the value NONE is returned
9	8	If any of the selected rules in the rule set are TOD rules, this word indicates the next time at which a TOD rule is to execute (hh:mm:ss); if none of the enabled rules in the rule set are TOD rules, the value NONE is returned
10	10	The most recent date on which a rule in the rule set was enabled (yyyy/mm/dd)

Word Number	Length	Contents
11	8	The most recent time at which a rule in the ruleset was enabled (<i>hh:mm:ss</i>)
12	11	The number of times that the rule was executed
13	6	If any of the selected rules in the rule set are TOD rules, this word provides a count of the enabled TOD rules that contain the NOMSG qualifier; if there are no enabled TOD rules meeting this criterion, an asterisk (*) is returned
14	6	If any of the selected rules in the rule set are TOD rules, this word provides a count of the enabled TOD rules that contain the NOSYNCH qualifier; if there are no enabled TOD rules meeting this criterion, an asterisk (*) is returned
15	6	If any of the selected rules in the rule set are TOD rules, this word provides a count of the enabled TOD rules that contain the CATCHUPMAN qualifier; if there are no enabled TOD rules meeting this criterion, an asterisk (*) is returned
16	6	If any of the selected rules in the rule set are TOD rules, this word provides a count of the enabled TOD rules that contain the CATCHUPYES qualifier; if there are no enabled TOD rules meeting this criterion, an asterisk (*) is returned
17	6	If any of the selected rules in the rule set are MSG rules, this word provides a count of the enabled MSG rules that contain the NOOPSLOG option; if there are no enabled MSG rules meeting this criterion, an asterisk (*) is returned
18	12	The highest number of External Data Queue (EDQ) entries created by any rule selected in this rule set.

Statistics for Rules

CA OPS/MVS inserts this information into the OPS/REXX external data queue when you issue the LISTINST command in either of these ways:

- You include *ruleset.rule* in the command.
- You include *ruleset.** in the command.

Each line of output describes one rule that meets the selection criteria specified on the LISTINST command.

Word Number: 1

Length: 8

Contains the message ID, which is OPx3944I.

Word Number: 2

Length: 8

Contains the name of the rule set.

Word Number: 3

Length: 8

Contains the name of the rule.

Word Number: 4

Length: 1

Contains the enablement status of the rule set, which is always E.

Word Number: 5

Length: 3

Contains a code indicating the rule type:

API-Application Program Interface rule

ARM-Automatic Restart Management rule

CMD-Command rule

DOM-Delete-operator-message rule

EOJ-End-of-job rule

EOM-End-of-memory rule

EOS-End-of-step rule

GLV-Global variable rule

MSG-Message rule

OMG-OMEGAMON rule

REQ-Request rule

SCR-Screen rule

SEC-Security rule

TLM-Time limit event rule

TOD-Time-of-day rule

USS-UNIX System Services rule

Word Number: 6

Length: 10

Contains the most recent date on which the rule was executed (*yyyy/mm/dd*).

Word Number: 7

Length: 8

Contains the most recent time at which the rule was executed (*hh:mm:ss*).

Word Number: 8

Length: 10

If the selected rule is a TOD rule, this word indicates the next date on which it is to execute (*yyyy/mm/dd*); if it is not a TOD rule, this word contains the value NONE.

Word Number: 9

Length: 8

If the selected rule is a TOD rule, this word indicates the next time at which it is to execute (*hh/mm/ss*); if it is not a TOD rule, this word contains the value NONE.

Word Number: 10

Length: 10

Contains the date on which the rule was enabled (*yyyy/mm/dd*).

Word Number: 11

Length: 8

Contains the time at which the rule was enabled (*hh:mm:ss*).

Word Number: 12

Length: 9

Contains the number of times that the rule was executed.

Word Number: 13

Length: 8

Contains the maximum number of times the rule is allowed to execute.

Word Number: 14

Length: 1

If the selected rule is a TOD rule, this word contains Y if the MSG qualifier was specified in the rule or N if the NOMSG qualifier was specified in the rule; this word contains an asterisk (*) if the rule is not a TOD rule.

Word Number: 15

Length: 1

If the selected rule is a TOD rule, this word contains Y if the SYNCH qualifier was specified in the rule or N if the NOSYNCH qualifier was specified in the rule; this word contains an asterisk (*) if the rule is not a TOD rule.

Word Number: 16

Length: 1

If the selected rule is a TOD rule, this word contains M if the CATCHUPMAN qualifier was specified in the rule, Y if the CATCHUPYES qualifier was specified in the rule or N if the CATCHUPNO qualifier was specified (the default); this word contains an asterisk (*) if the rule is not a TOD rule.

Word Number: 17

Length: 1

If the selected rule is a MSG rule, this word contains N if the NOOPSLOG option was specified in the rule or Y if the rule was permitted to default to OPSLOG; this word contains an asterisk (*) if the rule is not a MSG rule.

Word Number: 18

Length: 1-50

Contains the event ID specified on the header line of the rule-this is the event that causes the rule to execute; this word contains N/A if the rule is a TOD rule.

Word Number: 19

Length: 12

Contains the highest number of External Data Queue (EDQ) entries created by this rule.

AOF LISTSRC—List Source Code

This command lists the source code of an enabled rule. For each line of the source text of the enabled rule, one line of output is returned to the external data queue.

The LISTSRC command sets the RC variable to zero and the external data queue contains no output lines.

This command has the following format:

```
ADDRESS AOF "LISTSRC keywords"
{rulesetrule}
[SYSTEM(sysname)]
[SYSWAIT(seconds)]
```

Note: If the SYSTEM keyword is used for this command, it may contain only one system name. EXT and ALL refer to all external systems (more than one) and are therefore not allowed.

Examples: LISTSRC command

- To list the source code for a rule, issue this command:

```
ADDRESS AOF
"LISTSRC rulesetrule"
```
- To list the source code for the dynamic rule named NEWRULE, issue this command:

```
ADDRESS AOF
"LISTSRC *DYNAMICNEWRULE"
```

Usage Notes

Keep these things in mind when using the LISTSRC command:

- You may specify any valid rule set name for the value of *ruleset*, including *DYNAMIC. There is no default.
- You must specify the valid name of an enabled rule for the value of *rule*. You cannot specify an asterisk (*). There is no default.
- For all enabled rules to be eligible for LISTSRC processing, the AOFSOURCETEXT parameter must be set to YES. If the AOFSOURCETEXT parameter is set to NO, only dynamic rules are eligible for LISTSRC processing.
- When the value of the AOFSOURCETEXT parameter is NO and you issue the LISTSRC command, CA OPS/MVS returns the following information to the external data queue:
 - Return code 20
 - Message OPS3811I, indicating that CA OPS/MVS could not find the source text of the rule.

More information:

[ADDRESS AOF Command Format](#) (see page 43)

AOF RESETAUTO—Prevent Automatic Enabling of Rule

The RESETAUTO command prevents a rule or rule set from being automatically enabled at CA OPS/MVS startup.

This command has the following format:

```
ADDRESS AOF "RESETAUTO keywords"  
[ruleset|ruleset.rule]  
[SYSTEM(ALL|EXT|sysnames)]  
[SYSWAIT(seconds)]  
[OUTPUT|NOOUTPUT]
```

Issuing the RESETAUTO command produces these results:

- If the command executes successfully, the RC variable is set to zero and the external data queue contains no output lines.
- The RESETAUTO command also can automatically delete a compiled version of a rule or a rule set if you do the following:
 - Set the CA OPS/MVS AOFPRECOMPILED parameter to YES.
 - Set the CA OPS/MVS AOFPRECOMPILEDSDSN parameter to the name of a valid compiled rules library. For more information, see the *Parameter Reference*.

Note: The names of members in a compiled rules library are encrypted, so you cannot read them. The AOF RESETAUTO and DELCOMP commands provide the only ways to delete a compiled version of a rule. You can use the ISPF Delete PDS Member utility to delete compiled rules; however, since you cannot read the member names, you will have no idea which rules you are deleting.

Usage Notes

Keep these things in mind when using the RESETAUTO command:

- The *ruleset* operand turns off automatic enablement for all rules in the named rule set. The *ruleset.rule* operand turns off automatic enablement for a specific rule in the named rule set.
- The CA OPS/MVS product issues the RESETAUTO command for a rule or rule set for which automatic enablement is already turned off.
- All rule members must have valid ISPF statistics.

More information:

[ADDRESS AOF Command Format](#) (see page 43)

[ADDRESS AOF Return Codes](#) (see page 71)

AOF SETAUTO—Enable a Rule at Startup

The SETAUTO command automatically enables a rule or rule set at the next CA OPS/MVS startup.

This command has the following format:

```
ADDRESS AOF "SETAUTO keywords"  
[ruleset|ruleset.rule]  
[SYSTEM(ALL|EXT|sysnames)]  
[SYSWAIT(seconds)]  
[OUTPUT|NOOUTPUT]
```

Issuing the SETAUTO command produces these results:

- If the command executes successfully, the RC variable is set to zero and the external data queue contains no output lines.
- The SETAUTO command also can automatically save a compiled version of a rule or a rule set if you do the following:
 - Set the CA OPS/MVS AOFPRECOMPILED parameter to YES.
 - Set the CA OPS/MVS AOFPRECOMPILEDSDSN parameter to the name of a valid compiled rules library. For more information, see the *Parameter Reference*.

Usage Notes

Keep these things in mind when using the SETAUTO command:

- The *ruleset* operand automatically turns on auto-enablement for all rules in the named rule set. The *ruleset.rule* operand automatically turns on enablement for a specific rule in the named rule set.
- The CA OPS/MVS product allows you to issue the SETAUTO command for a rule or rule set that is already enabled automatically.
- All rule members must have valid ISPF statistics.

Note: You can use the ISPF Delete PDS Member utility to delete compiled rules; however, since you cannot read the member names, you will have no knowledge of which rules you are deleting.

More information:

[ADDRESS AOF Command Format](#) (see page 43)

[ADDRESS AOF Return Codes](#) (see page 71)

AOF SUBSYS—Set Subsystem Identifier

The SUBSYS command sets the z/OS subsystem identifier (SSID) of the CA OPS/MVS product that will process subsequent ADDRESS AOF commands (in addition to some REXX functions, such as OPSVALUE and OPSPRM). Use this command only if you are running multiple copies of CA OPS/MVS *and* you want the commands addressed to a CA OPS/MVS copy that is not using the default SSID for the CA OPS/MVS product, which is OPSS.

Issuing the SUBSYS command sets the RC variable to zero and the external data queue contains no output lines.

This command has the following format:

```
ADDRESS AOF "SUBSYS keyword"  
[ssid]
```

SUBSYS

Identifies the subsystem where the CA OPS/MVS copy to receive ADDRESS AOF commands is running.

Note the following usage information:

- Only ADDRESS AOF commands issued from the OPS/REXX program issuing the SUBSYS command go to the specified SSID. ADDRESS AOF commands issued from subsequent OPS/REXX programs running in the same address space, or from other address spaces, go to other destinations.
- For OPS/REXX programs run as part of a started task, the default subsystem ID is the first four characters of the current step name, so long as the first three characters are OPS.

ADDRESS AOF Return Codes

ADDRESS AOF commands produce the following return codes:

<0

The host command environment abended while processing the AOF command.

0

The command executed successfully.

>0

The host command issuing the AOF command contained errors.

4

A warning message was issued.

6

AOF is inactive (The AOFACTIVE parameter is set to OFF)

8

The command timed out because it took too much time to complete. Not all responses were received.

10

Rule set OPEN error

12

An error occurred in one of the I/O service routines. Check related messages in the external data queue and/or the OPSLOG to determine the cause

16

The AOF command contained a syntax error.

18

Rule set CLOSE error

20

The subsystem is not active, and CA OPS/MVS is not running.

22

Parameter list error (this is an internal error)

24

The subsystem version is incompatible with this command.

26

GETMAIN/FREEMAIN storage error. Insufficient storage in the main product address space.

28

SENDMSG failed.

30

Catalog error detected when attempting to find the rule data sets that match the ruleset prefix and suffix.

32

The authorization exit rejected the AOF command.

34

Rule not found.

36

The user exit abended.

38

Rule has no ISPF statistics

40

Required output was not produced.

42

There is an AOF compile syntax error.

44

The SYSWAIT time expired before all output was retrieved from the remote system.

45

Ruleset prefix error.

46

Ruleset suffix error.

48

Receive message error from cross-system operation.

52

Either of the following:

- Message queue allocation or deallocation failed.
- The AOF facility is not active.

56

The specified system ID is not defined to the MSF.

57

Ruleset alternate prefix error.

60

The specified system is not active.

64

There is a mismatch between MSF versions.

68

An action command was ignored, because the remote system that sent the command is not secure.

72

An internal service failed. An informative message is issued that further documents the failure.

76

The maximum dynamic rule size of one megabyte has been exceeded.

80

The maximum dynamic rule line length of 255 bytes has been exceeded.

84

The command failed because it was not issued from the SECURITYRULESET.

196

An ADDRESS AOF list type command (LIST, LISTSRC, LISTCOMP, LISTINST) was issued but the NOOUTPUT flag was either explicitly set or implied. Since no output can be returned, this operation is considered to be an error. Specifying multiple systems or using either EXT or ALL in the SYSTEM keyword on one of the list commands above most likely causes this error.

200

The rule has no PROC or TERM section and has been automatically disabled. This does not necessarily indicate an error.

ADDRESS AP Commands

The OPS/REXX ADDRESS AP hostcommand environment allows one-way communication from CA OPS/MVS to CA Automation Point. With ADDRESS AP, you can:

- Send a REXX EXEC to be executed in CA Automation Point
- Send a Notification Manager command to CA Automation Point
- Send a PPQ WRITE command to CA Automation Point

AP NMFind—Invoke Notification Manager

The ADDRESS AP NMFIND invokes Notification Manager on an MSF-connected CA Automation Point system. You tell it:

- Whom to contact
- What to tell the person
- What action to take when the person responds

Note: NMFIND is not a synchronous process, therefore CA OPS/MVS does not wait for an action or an answer from CA Automation Point.

The ADDRESS AP NMFIND command has the following format:

```
ADDRESS AP NMFIND
SYSTEM(apsysname)
{GROUP(group)|NAME(name)|PERSON(person)}
TELL(tell-text)
[ACKNOWLEDGEOPS(OPShost,string)]
[ACKNOWLEDGEAP(string)]
[ATTACHMENT(filename)]
[FAILUREREXX(failure-action)]
[MTUP(string)]
[USERPARMS(parameter1(value1)[parameter2(value2)])]
```

SYSTEM

Defines the 1- to 8-character MSF name of the CA Automation Point system. A system ID *cannot* exceed 8 bytes.

GROUP

Defines the 1- to 32-character name of the group to contact.

Mutually exclusive with NAME and PERSON.

NAME

Defines the 1- to 32-character name of the group or person to contact.

Mutually exclusive with GROUP and PERSON.

PERSON

Defines the 1- to 32-character name of the person to contact.

Mutually exclusive with GROUP and NAME

TELL

Sends the message to the specified group, name, or person.

The length of the TELL string is subject to restrictions on the local CA Automation Point machine. For details, see the CA Automation Point documentation.

ACKNOWLEDGEOPS

(Optional) Sends an acknowledgement message with *string* included in the message text to the OPSLOG and AOF of the CA OPS/MVS system whose MSF ID is *OPShost*.

string must be a character string with a maximum length of 240 characters.

ACKNOWLEDGEAP

(Optional) Writes an acknowledgement message with *string* included in the message text to the CA Automation Point Message window specified by *apsysname*. CA Automation Point rules may therefore be written to react to acknowledgement messages.

string must be a character string with a maximum length of 240 characters.

ATTACHMENT

(Optional) Identifies the user-supplied file that is to be attached to any notifications made using a method defined to send mail using the SENDMAIL command. The file must be accessible from the Notification Server that is issuing the SENDMAIL command. Only one file can be specified per NMFIND request. The maximum length of the filename (including path) is 512 characters.

Example:

ATTACHMENT('c:\Documents and Settings\All Users\Application Data\Application\ error.log')

Note: To use the ATTACHMENT option, you must configure the Notification Server to use SMTP for mail requests.

Default: There is no default.

FAILUREREXX

(Optional) Identifies the name of the REXX program to run on the CA Automation Point system when every action in the call tree fails. This value must be a valid CA Automation Point REXX program name and should be consistent with length capabilities of the AP REXX PROGRAM keyword, allowing enough space for the directory name.

MTUP

(Optional) Specifies which method the Methods to Use Profile (MTUP) uses for a particular instance of an NMFND notification request.

Valid values are:

profile

Any combination of method type codes B through W.

A

All method types specified for all active schedules.

Default: A

For more information on the MTUP operand, see the chapter “Notification Manager Commands” in the *CA Automation Point Reference Guide*.

USERPARMS

(Optional) Specifies a list of method parameters whose associated values override any like-named parameters during the execution of NMFIND.REX. The length of the USERPARMS list is subject to restrictions on the local CA Automation Point machine.

For example, assume this operand is defined as follows:

```
USERPARMS( SubjectText(UAP notification using NM))
```

The USERPARMS-defined value overrides the parameter SubjectText for any method using that parameter.

Default: There is no default.

DEBUG

(Optional) Indicates whether debugging messages are to be generated.

Valid values are:

YES-Generate debugging messages.

NO-Do not generate debugging messages.

Default: NO

Example: ADDRESS AP NMFIND

Suppose you have determined that when a particular JES is having difficulties, you need to NMFIND the lead JES systems programmer, Jim Smith. The CA OPS/MVS rule that trapped the error message from JES would contain this clause:

```
ADDRESS AP "NMFIND SYSTEM(SYS47) PERSON('JIM SMITH') TELL('JES is down')"
```

Notification Manager uses its database technology (which is based on a relational database) to determine the communications method it should use to contact Jim Smith, based on the time and day. Notification Manager proceeds to contact Jim Smith and relay the message according to the following:

- If the contact method was the CA Automation Point voice technology, Jim receives a phone call at the phone number pointed to by his notification schedule.
- If the contact method was the numeric pager, Notification Manager pages Jim with the numeric message consisting of the phone number that he needs to call and an ID number that will authorize him to receive the voice message.

If the contact method was the alphanumeric pager, the message appears on the pager that belongs to Jim along with a phone number and ID that he can use to obtain any information that was not sent through his pager. For instance, the message to be sent might be longer than the length supported by his paging service.

AP PPQ—Write to Queue

The ADDRESS AP PPQ WRITE command writes one or more items to a specified queue on an MSF connected CA Automation Point system.

The ADDRESS AP PPQ WRITE command has the following format:

```
ADDRESS AP 'PPQ WRITE {SYSTEM(apsysname)} {QUEUE(qname)}  
{ITEM(item)}[VAR(varname)]  
[ITEMNUM(LAST|FIRST|itemnum)]'
```

The following operands are required:

SYSTEM

Defines the 1- to 8-character MSF name of the CA Automation Point system. A system ID *cannot* exceed 8 bytes.

QUEUE

Defines the 1- to 16-character name of the queue (*qname*). The queue name can contain alphanumeric characters and any of these special characters:

! @ # \$ _ (for example, !Qun1, Queue1, 12334)

The queue name cannot contain blanks.

ITEM

Specifies the item to write to the queue. The *item* value cannot exceed 30,000 characters in length and can be either a literal string (such as “this is an item”) or a simple variable name (not enclosed in the host command quotation marks so that REXX can evaluate it).

Note: You cannot specify the ITEM operand if you specify the VAR operand.

VAR

Defines the name of a simple variable on the local machine containing the item to be written to the queue, *qname*.

Example:

```
Vartest = 'this is a test'
```

```
variable='12345.....'
```

The rule for varname is to follow standard simple variable naming conventions used in REXX.

varname cannot exceed 50 characters in length. The value of *varname* cannot exceed 30,000 characters in length.

Note: You cannot specify the VAR operand if you specify the ITEM operand.

ITEMNUM

(Optional) Specifies the starting item to write to the queue.

Assume that the specified queue contains count number of items and that you want to write *n* items to the queue. Valid ITEMNUM values are:

LAST

Starts the write operation from the end of the queue. The first item written to the queue is item number *count*+1 and succeeding items are numbered *count*+2, *count*+3, and so on through *count*+*n*. This is the default method for writing to the queue.

FIRST

Starts the write operation from the beginning of the queue. The current item number 1 in the queue becomes item number 1+*n* after *n* items are written.

Itemnum

Starts the write operation from somewhere in the middle of the queue. The WRITE command writes *n* items ahead of an item (specified by *itemnum*) that already exists in the queue (so that the current *itemnum* becomes *itemnum*+*n* after *n* items are written).

Note: The starting item that you specify must exist or must be the *count*+1 item in the queue; specifying an *itemnum* value beyond the end of the queue generates an error.

Default: LAST or 0

AP REXX—Execute a REXX EXEC

ADDRESS AP REXX sends a command to an MSF-connected CA Automation Point system to execute a REXX EXEC. CA OPS/MVS can send REXX string commands up to 30,000 characters long.

ADDRESS AP REXX command has the following syntax:

```
ADDRESS AP REXX {SYSTEM(apsysname)} {PROGRAM(REXX-program-name)}  
[ARG(data)]
```

SYSTEM

Defines the 1- to 8-character MSF name of the CA Automation Point system.

PROGRAM

Defines the 1- to 256-character name of the REXX program. The program name can contain alphanumeric characters and any of these special characters: ! @ # \$ _

The program name cannot contain blanks.

ARG

(Optional) Defines an arbitrary (possibly null) string of arguments to be passed to the REXX program. The length of this string may not exceed 30,000 characters in length, and must be character data only (no binary or packed decimal data).

If the argument string contains blanks or non-alphabetic characters, enclose the argument string in quotation marks.

Example: How to specify the argument string

```
ADDRESS AP  
"REXX SYSTEM(SYS02X)",  
"PROGRAM(PROGA)",  
"ARG(This argument string contains blanks)"
```

ADDRESS AP Return Codes

ADDRESS AP commands produce the following return codes:

0

The specified CA Automation Point system was found to be active to MSF, and the NMFIND command was sent to the MSF CCI send queue. This return code does not provide any indication as to whether the command executed successfully.

08

Maximum command length exceeded.

12	Invalid character found.
14	Syntax error.
16	Parse error.
18	Item and & variable mutually exclusive.
20	Main address space inactive.
22	Major control block error.
24	Item or variable keyword missing.
28	No valid system found.
30	Internalabend occurred.
34	PPQ WRITE variable value too long.
38	Security module rejected command.
40	Invalid request type.
44	Invalid queue name blank not allowed.
48	Invalid queue name.
50	Group/Name/Person are mutually exclusive.
52	An internal abend has occurred.

53

MTUP-if first character is A, then string length must be 1.

54

MTUP characters must be alphabetic.

55

MTUP characters must be A or B to W.

56

MTUP-the A character can only be used alone.

60

Host variable not found.

61

Bad variable name.

62

Invalid variable value.

63

Variable name is invalid.

64

Host variable resolve out of storage.

68

Storage pool obtain failed.

70

No system found.

74

Current system is not an MSF AP defined system.

80

Send message to AP queue failed.

ADDRESS EPI Commands

ADDRESS EPI commands allow OPS/REXX programs to define and operate virtual terminals that interact with VTAM applications through the CA OPS/MVS External Product Interface (EPI) facility. Using EPI, OPS/REXX programs can:

- Log on to a VTAM application.
- Enter commands and data from a simulated virtual terminal keyboard.
- Read data from the virtual terminal screen.
- Log off from a VTAM application.
- Share a single session with applications (such as OMEGAMON) running in VTAM mode.
- Automate responses when a VTAM application triggers a system event; for example, when:
 - A message rule issues a message.
 - The application issues a command.
 - An OMEGAMON rule triggers an OMEGAMON event.
 - A request rule triggers a user-initiated event.
 - A TOD rule schedules an event at a predetermined time of day.

The ADDRESS EPI commands and procedures for using the EPI are so closely related that any discussion of EPI use needs to include descriptions of the commands. For this reason, the *Command and Function Reference* lists the EPI host commands, their keywords and return codes, and briefly summarizes what these commands do. For complete information about using EPI host commands, see the chapter “External Product Interface” in the *User Guide*.

Types of ADDRESS EPI Commands

The following lists the types of EPI host commands:

- Output control ADDRESS EPI commands affect the format of the output from all other EPI host commands. Commands in this group are:
 - MSGID
 - SUBATTR
 - SUBSYS
 - SUBUNPT
 - TIMEOUT
 - TRIM

- Virtual terminal ADDRESS EPI commands control the EPI use of virtual terminals. Commands in this group are:
 - CHANGE
 - DEFINE
 - DELETE
 - DEQ
 - DISABLE
 - ENABLE
 - ENQ
 - INQINPUT
 - LIST
 - LOGOFF
 - LOGON
 - MVCURSOR
 - PEEK
 - POKE
 - RDCURSOR
 - RDSCREEN
 - RDSCRROW
 - SETMODEL
 - SETTERM
 - SETUNAME
 - TRACE
 - TYPE
 - TYPESEC
 - TYPETEST
- These commands support the EPI Record and Playback feature, although they may also be used independently:
 - BIND
 - GETSCRN
 - SESSCMD
 - UNBIND

- General ADDRESS EPI commands provide information about or control processing time for other ADDRESS EPI commands. Commands in this group are:
 - DEBUG
 - HELP
 - WAIT
 - WAITTOD

ADDRESS EPI Return Codes in REXX Programs

When any ADDRESS EPI command executes in a REXX program, the variable RC in that program will contain one of the following return codes:

<0

The host environment abended while processing the EPI command.

0

The command executed successfully.

>0

The host command issuing the EPI command contained errors.

4

A warning message was issued.

8

The command timed out because it took too much time to complete. Not all responses were received.

12

The command failed, and an error message was issued.

16

The EPI command contained a syntax error.

20

Either of the following:

- The subsystem is not active, and CA OPS/MVS is not running.
- The EPI facility is not active.

24

The subsystem version is incompatible with this command.

28

SENDMG failed.

32

The authorization exit rejected the EPI command.

36

The user exit abended.

EPI BIND—Dedicate a VTAM Session

Use this command to temporarily dedicate a VTAM application session to a specific CA OPS/MVS TSO user, CLIST, or REXX EXEC. While a session is bound, the TSO user or automation procedure that issued the BIND command can send commands to that session without interruption from other users or procedures.

This command has the following format:

```
ADDRESS EPI "BIND keywords"  
{sessid}  
[waittime]  
[SUBSYS(ssid)]
```

sessid

Defines the session ID for the CA OPS/MVS-monitored VTAM session to be bound.

waittime

(Optional) Defines the time, in seconds, that CA OPS/MVS waits for the session to become available. If the session does not become available in the allotted time, CA OPS/MVS cancels the BIND attempt with a return code of 20.

The waittime can be from 1 to 600 seconds.

Default: 60

SUBSYS

(Optional) Addresses the command to a copy of CA OPS/MVS with a subsystem ID other than the default subsystem ID.

Default: OPSS

Example: EPI BIND

The following binds the VTAM session with session ID CICS1 and waits for up to two minutes:

```
BIND CICS1,120
```

BIND Return Codes

The BIND command produces these return codes:

0

The command completed successfully.

4

A syntax error occurred. CA OPS/MVS generates messages describing the error.

8

The EPI terminal, or session ID, is not defined.

12

The CA OPS/MVS subsystem is not active.

16

The session is not available.

20

The BIND command timed out.

40

There is no response from the main subsystem.

EPI CHANGE—Change Virtual Terminal Definition

Use the CHANGE command to change an existing definition of a virtual terminal. You can only change virtual terminals that are currently in an inactive state.

This command has the following format:

```
ADDRESS EPI "CHANGE keywords"  
{termname|ALL|*}  
[PASSWORD(termpasswd)]  
[APPLID(prodappl)]  
[LOGMODE(modename)]  
[LOGONPARM("parmstring")]  
[ACCEPT|REJECT]  
[NORETRY|RETRY(seconds retries)]
```

termname

Defines the name of the virtual terminal being changed, which must match the name of an APPL statement in your VTAM definition list library.

ALL

The CHANGE ALL command can save you time in modifying your virtual terminal definitions. If you define and activate new virtual terminals after you have issued the CHANGE ALL command, reissue the CHANGE ALL command to give your new virtual terminals the same characteristics as the other terminals.

*

Gives the current virtual terminal the new definition through the SETTERM command.

PASSWORD(*termpswd*)

(Optional) Provides the password for the named virtual terminal, or for all virtual terminals if you are issuing the CHANGE ALL command. Use this keyword only if the VTAM APPL statement for this virtual terminal includes the PRTCT keyword, which allows VTAM to provide password protection. The password you specify with the CHANGE command must match the password, if any, on the VTAM APPL statement for this virtual terminal.

Alternatively, you can specify the PASSWORD keyword on the EPI ENABLE command.

APPLID(*prodappl*)

(Optional) Defines the VTAM application name of the external product that this virtual terminal is logged on to. See the description of the EPI LOGON command. You can either specify the APPLID keyword in the CHANGE command or enter it when you log on to EPI. If you supply a prodappl value at logon and on the CHANGE command, the value supplied at logon takes precedence.

LOGMODE(*modename*)

(Optional) Defines the name of the entry in the VTAM logmode table to be used when this virtual terminal is logged on. This entry should describe an SLU2 type virtual terminal used by IBM 3278 terminal models 2, 3, or 4.

If neither your CHANGE command nor the LOGON command you issue for this terminal specifies a logmode, EPI uses the default logmode table entry (as defined by VTAM for your system) for this virtual terminal.

LOGONPARM(*'parmstring'*)

(Optional) Defines the character string, enclosed in quotation marks, that EPI passes to an external product when this virtual terminal logs on to that product. Supplying a parmstring on your CHANGE command or when you log the virtual terminal onto an external product is optional.

ACCEPT or REJECT

(Optional) These mutually exclusive keywords determine whether EPI accepts requests from external products to acquire this virtual terminal. VTAM does not allow disabled virtual terminals to be acquired.

NORETRY

(Optional) Overrides a previously issued EPI CHANGE or LOGON command for this virtual terminal that specified RETRY. When you specify NORETRY, EPI does not allow retrying of LOGON commands and does not try to reestablish a session that fails after logon.

RETRY(seconds retries)

(Optional) Overrides a previously issued EPI CHANGE or LOGON command for this virtual terminal that specified NORETRY. The *seconds* value specifies how many seconds EPI waits between attempts to log on to a virtual terminal or to reestablish a failed session. Use a value between 1 and 86400 (the number of seconds in 24 hours). The default value is 30 seconds. The *retries* value specifies how many times EPI can try to log on to a virtual terminal or to reestablish a failed session. The default value is 0, which allows only one retry attempt; if that attempt fails, EPI places the virtual terminal in NORETRY mode. You can specify any number of retries up to 65535.

Example: Use EPI CHANGE to change the logmode

The following code changes the logmode value for all virtual terminals to T3278M2:

```
CHANGE ALL LOGMODE(T3278M2)
```

CHANGE Return Codes

The CHANGE command produces these return codes:

0

The command executed successfully.

4

The command did not execute for one of these reasons:

- The terminal is not defined.
- No terminals were inactive when a CHANGE ALL command was issued.
- The terminal is active, so its definition could not be changed.

12

The command failed because the new terminal name already exists.

EPI DEBUG—Control VTAM Exit Debugging

Use this command to turn VTAM exit debugging on or off.

This command has the following format:

```
ADDRESS EPI "DEBUG keyword"  
{ON|OFF}
```

ON

Turns debugging on and causes EPI to write a hardcopy message each time a VTAM exit is entered.

OFF

Turns debugging off.

Default: OFF

Example: EPI DEBUG

The following turns the VTAM exit debugging on:

```
DEBUG ON
```

DEBUG Return Codes

The DEBUG command produces the return codes described in ADDRESS EPI Return Codes in REXX Programs in this chapter.

EPI DEFINE—Define a Virtual Terminal to EPI

Use this command to define a virtual terminal to EPI.

This command has the following format:

```
ADDRESS EPI "DEFINE keywords"  
{termname}  
[PASSWORD(termpasswd)]  
[APPLID(prodappl)]  
[LOGMODE(modename)]  
[LOGONPARM("parmstring")]  
[ACCEPT|REJECT]  
[NORETRY|RETRY(seconds)]
```

termname

Provides the name of the virtual terminal being changed, which must match the name of an APPL statement in your VTAM definition list library.

PASSWORD(*termpswd*)

(Optional) Provides the password for the named virtual terminal, or for all virtual terminals if you are issuing the CHANGE ALL command. Use this keyword only if the VTAM APPL statement for this virtual terminal includes the PRTCT keyword, which allows VTAM to provide password protection. The password you specify with the CHANGE command must match the password, if any, on the VTAM APPL statement for this virtual terminal.

Alternatively, you can specify the PASSWORD keyword on the EPI ENABLE command.

APPLID(*prodappl*)

(Optional) Defines the VTAM application name of the external product that this virtual terminal is logged on to. See the description of the EPI LOGON command. You can either specify the APPLID keyword in the CHANGE command or enter it when you log on to EPI. If you supply a prodappl value at logon and on the CHANGE command, the value supplied at logon takes precedence.

LOGMODE(*modename*)

(Optional) Defines the name of the entry in the VTAM logmode table to be used when this virtual terminal is logged on. This entry should describe an SLU2 type virtual terminal used by IBM 3278 terminal models 2, 3, or 4.

If neither your CHANGE command nor the LOGON command you issue for this terminal specifies a logmode, EPI uses the default logmode table entry (as defined by VTAM for your system) for this virtual terminal.

LOGONPARM(*'parmstring'*)

(Optional) The character string, enclosed in quotation marks, that EPI passes to an external product when this virtual terminal logs on to that product. Supplying a parmstring on your CHANGE command or when you log the virtual terminal onto an external product is optional.

ACCEPT or REJECT

(Optional) These mutually exclusive keywords determine whether EPI accepts requests from external products to acquire this virtual terminal. VTAM does not allow disabled virtual terminals to be acquired.

NORETRY

(Optional) Overrides a previously issued EPI CHANGE or LOGON command for this virtual terminal that specified RETRY. When you specify NORETRY, EPI does not allow retrying of LOGON commands and does not try to reestablish a session that fails after logon.

RETRY(seconds retries)

(Optional) Overrides a previously issued EPI CHANGE or LOGON command for this virtual terminal that specified NORETRY. The *seconds* value specifies how many seconds EPI waits between attempts to log on to a virtual terminal or to reestablish a failed session. Use a value between 1 and 86400 (the number of seconds in 24 hours). The default value is 30 seconds. The *retries* value specifies how many times EPI can try to log on to a virtual terminal or to reestablish a failed session. The default value is 0, which allows only one retry attempt; if that attempt fails, EPI places the virtual terminal in NORETRY mode. You can specify any number of retries up to 65535.

Example: EPI DEFINE

The following example names the virtual terminal TERM1, associates it with the application OMVTAM, and specifies the logmode as T3278M2:

```
DEFINE TERM1 APPLID(OMVTAM) LOGMODE(T3278M2)
```

DEFINE Return Codes

The DEFINE command produces these return codes:

0

The terminal is now defined to EPI.

4

You cannot use ALL as a terminal name.

12

The command failed. An error message reports the error that caused the command failure; either the new terminal name already exists or you specified no terminal name.

EPI DELETE—Delete Virtual Terminal Definitions

Use this command to delete one or more virtual terminal definitions from EPI.

This command has the following format:

```
ADDRESS EPI "DELETE keyword"  
{termname|ALL|*}
```

termname

Defines the name of the virtual terminal whose definition you want to delete. You cannot delete a virtual terminal definition unless you have disabled that terminal using the EPI DISABLE command.

ALL

Instructs EPI to delete all terminal definitions.

Deletes the current terminal defined through the SETTERM command.

Example: EPI DELETE

The following deletes the virtual terminal named TERM2:

```
DELETE TERM2
```

DELETE Return Codes

The DELETE command produces these return codes:

0

The command executed normally.

4

No terminals were deleted because the terminals have not been defined or because the terminals are enabled.

12

The deleted request failed and an error message was issued.

EPI DEQ—Release Virtual Terminal Ownership

Use the DEQ command to release ownership of (dequeue) the virtual terminal. Request rules are the only AOF rules that can use this command.

This command has the following format:

```
ADDRESS EPI "DEQ keywords"
```

```
{termname*}
```

```
/* You can choose one or both of the following optional keywords. */
```

```
[FORCE]
```

```
[ALL]
```

termname

Defines the virtual terminal to be released.

*

Release the current terminal defined through the SETTERM command.

FORCE

(Optional) Forces the current ownership of this virtual terminal to be released, even if the current program does not own the terminal.

Important! Use the FORCE keyword with caution, because it can produce undesirable results.

ALL

(Optional) Forces the release of *all* virtual terminals.

Example: EPI DEQ

The following releases ownership of the TERM2 virtual terminal:

```
DEQ TERM2
```

DEQ Return Codes

The DEQ command produces these return codes:

0

The command executed successfully.

4

The command did not execute because the named terminal is disabled, or because the command specified a nonexistent terminal name.

12

The command failed. When this occurs, EPI issues this error message:

No ENQ was found. Error posting next ENQ in chain.

EPI DISABLE Command—Disable Virtual Terminals

Use this command to disable one or more virtual terminals.

Note: We strongly recommend that you do not use the EPI DISABLE command in your automation procedures; the use of this command may result in hang conditions inside the VTAM code, and the EPI will be unable to process commands. Rather, you should use the EPI ENABLE command at initialization to enable your virtual terminals, and you should keep the terminals enabled until the product is terminated.

This command has the following format:

```
ADDRESS EPI "DISABLE keywords"  
{termname|ALL|*}
```

termname

Defines the name of the virtual terminal you want to disable. EPI stops the session between this virtual terminal and the external product it is logged onto before disabling the terminal. Once a terminal is disabled, it stops communicating with the external product and EPI. If any commands were being processed when the terminal was disabled, those commands do not execute. An EPI message reports that the terminal was disabled.

You cannot delete a virtual terminal definition unless you have disabled that terminal using the DISABLE command.

ALL

Disables all virtual terminals.

Important! Although the command DISABLE ALL shuts the EPI down quickly, use it with caution, because an external product logged onto one or more of your virtual terminals can have problems if you disable its terminal without logging off properly.

*

Disables the current terminal defined through the SETTERM command.

Example: EPI DISABLE

The following disables the TERM1 virtual terminal:

```
DISABLE TERM1
```

DISABLE Return Codes

The DISABLE command produces these return codes:

0

The command executed as usual.

4

No terminals were disabled because the terminals have not been defined or because the terminals are already disabled.

EPI GETSCRN—Display Screen Image

Use this command to fetch the current screen image from a CA OPS/MVS monitored VTAM session and display it on the terminal. Optionally, GETSCRN can store the screen image in OPS/REXX variables describing the contents and state of the screen.

This command has the following format:

```
ADDRESS EPI "GETSCRN keywords"  
{SESSION(ssid)}  
[CMDRESP(dest)]  
[PREFIX(prefix)]  
[SUBSYS(ssid)]  
[TRUNCATE(YES|NO)]
```

SESSION

Defines the one- to eight-character session ID of the monitored session.

CMDRESP

(Optional) The CMDRESP operand specifies the kind of variables to store the fetched screen image. The *dest* value can be either CLIST or REXX. CLIST is the default.

PREFIX

(Optional) The PREFIX operand specifies the one- to six-character prefix to be used when creating the CLIST or REXX variables. For more details on these variables, see Using SESSCMD in a CLIST or REXX EXEC in this chapter.

SUBSYS

(Optional) The SUBSYS operand addresses the command to a copy of CA OPS/MVS with a subsystem ID other than the default subsystem ID, which is OPSS.

TRUNCATE

(Optional) The TRUNCATE operand specifies whether CA OPS/MVS truncates the screen image display when GETSCRN is entered from a TSO terminal. Possible TRUNCATE values are YES or NO.

Default: YES

Examples: GETSCRN command

- The following command fetches the current screen of the VTAM session with session ID CICS2 and displays it on the TSO terminal:

```
GETSCRN SESSION(CICS2)
```

- This example shows the use of GETSCRN in a REXX program. This REXX program retrieves the current screen from a VTAM session and displays all the CLIST variables that CA OPS/MVS generates from the display.

```
SESSION = ARG(1)
ADDRESS EPI
'GETSCRN SESSION('SESSION') PREFIX(LINE)'
IF RC > 0 THEN EXIT
SAY 'THE FOLLOWING VARIABLES HAVE BEEN SET'
SAY 'CSRCOL = CSRCOL'
SAY 'CSRPOS = CSRPOS'
SAY 'CSRROW = CSRROW'
SAY 'SCREEN = SCREEN'
SAY 'SCRLEN = SCRLEN'
SAY 'SCRSIZE = SCRSIZE'
SAY 'SCRSTAT = SCRSTAT'
SAY 'SCRWDTH = SCRWDTH'
DO I=1 TO LINE0
  SAY I ' ' LINEI
END
RETURN
```

GETSCRN Return Codes

The GETSCRN command produces these return codes:

0

Completed successfully.

Cancels Currently Executing Script: No

4

ATM0111I USER NOT AUTHORIZED TO ACCESS command

Cancels Currently Executing Script: Yes

8

ATM1110I SESSION sessid DOES NOT EXIST

Cancels Currently Executing Script: Yes

12

ATM0106I CA OPS/MVS IS NOT ACTIVE TO PROCESS COMMAND

Cancels Currently Executing Script: No

16

One of the following messages:

- ATM1140I SESSION CONTROL TASK IS NOT ACTIVE
- ATM1141I SESSION IS NOT AVAILABLE
- ATM1142E OPEN INTERFACE HAS ABENDED

Cancels Currently Executing Script: Yes (for all three messages)

20

ATM1143I SCREEN IMAGE IS NOT AVAILABLE

Cancels Currently Executing Script: No

EPI ENABLE—Activate a Virtual Terminal

Use this command to activate a virtual terminal by issuing a VTAM OPEN ACB request. The ENABLE command also determines whether external products can acquire this virtual terminal.

This command has the following format:

```
ADDRESS EPI "ENABLE keywords"  
      {termname|ALL|*}
```

termname

Defines the name of the virtual terminal to be enabled. You can enable only terminals previously defined through the DEFINE command, and the virtual terminal name (the VTAM application ID) must be defined to VTAM.

ALL

Instructs EPI to try to enable all defined virtual terminals that currently are disabled.

Enables the current terminal defined through the SETTERM command.

Example: EPI ENABLE

The following enables the virtual terminal flagged as the current terminal (through the SETTERM command):

```
ENABLE *
```

ENABLE Return Codes

The ENABLE command produces these return codes:

0

The command executed as usual.

4

No terminals were enabled (ENABLE ALL) because the terminals have not been defined or because the terminals are disabled.

12

The enable request failed and an error message was issued.

EPI ENQ—Enqueue a Virtual Terminal

Use this command to enqueue a virtual terminal or test for terminal ownership. If no EPI DEQ command is issued to release an enqueued terminal, that terminal is dequeued automatically when the REXX program ends. To enqueue more than one terminal, issue multiple ENQ commands in ascending order by terminal name. Request rules are the only AOF rules that can use this command.

This command has the following format:

```
ADDRESS EPI "ENQ keywords"  
    {termname!}  
    [NOWAIT|TEST|WAIT]
```

termname

Defines the name of the terminal that you want to enqueue.

The current terminal defined through the SETTERM command.

NOWAIT

(Optional) Tells EPI to assign ownership of the virtual terminal only if it is available immediately (otherwise, ownership is not assigned).

TEST

(Optional) Tells EPI to return control to the REXX program immediately and to set the return code to indicate the status of terminal ownership.

WAIT

(Optional) Tells EPI to wait until the virtual terminal is owned before returning control to the REXX program issuing the ENQ command. Specifying *termname* as the only ENQ operand has the same result as specifying the WAIT keyword.

Default: WAIT

Example: EPI ENQ

The following enqueues virtual terminal TERM1:

```
ENQ TERM1
```

ENQ Return Codes

The ENQ command produces these return codes:

0

If the command specified WAIT or NOWAIT, ownership has been assigned. If the command specified TEST, the terminal was available without waiting.

4

The command did not execute. EPI issues a warning message explaining that the command did not execute because the terminal was disabled, was not defined, or is already enqueued.

12

The command failed for one of the following reasons, described in an EPI error message:

- Another user has already enqueued the terminal.
- A conflict exists with an ENQ command issued previously.
- The ENQ request area GETMAIN failed.

EPI HELP—Returns Valid EPI Commands

The EPI HELP command returns a list of all the valid EPI host commands to the external data queue. Use this command when you receive a new version of the EPI to determine which EPI commands it recognizes.

This command has the following format:

```
ADDRESS EPI "HELP" (no keywords)
```

HELP Return Code

The HELP command produces this return code:

0

The command executed successfully.

EPI INQINPUT—Input Indicator

Use this command before issuing the TYPE command, to make sure that the virtual keyboard is not locked (that is, its input inhibited indicator is off and it can accept input from the REXX program that issued the INQINPUT command).

This command has the following format:

```
ADDRESS EPI "INQINPUT keywords"  
{termname*}  
[NOWAIT|WAIT]
```

termname

Defines the name of the terminal about which you are inquiring.

*

Inquire about the state of the current terminal defined through the SETTERM command.

NOWAIT

(Optional) Tells EPI to return the current state of the input indicator immediately. To force this indicator off, allowing the program to send data to the virtual terminal, issue this command:

```
TYPE termname RESET
```

Issuing the above command does not prevent the external product from turning the indicator back on.

WAIT

(Optional) Tells EPI to place the program issuing the INQINPUT command into a wait state until the external product connected to this virtual terminal has unlocked the keyboard.

Example: INQINPUT WAIT

The following code causes the REXX program interacting with the TERM2 virtual terminal to pause until the virtual keyboard can accept input:

```
INQINPUT TERM2 WAIT
```


INQINPUT Return Codes

The INQINPUT command produces these return codes:

- 0**
Input is not inhibited.
- 1**
Input is inhibited. In this situation, only the !RESET scan code will operate.
- 4**
The virtual terminal is not defined.
- 12**
The command failed because the virtual terminal is not enabled.

EPI LIST—Display Virtual Terminal Status

Use this command to display the status of one or more virtual terminals. The LIST command runs synchronously in AOF rules and returns data to the external data queue.

This command has the following format:

```
ADDRESS EPI "LIST keywords"  
{termname|ALL|*}
```

termname
Defines the name of the terminal for which you want status information. EPI displays only information about the named terminal.

ALL
Instructs EPI to display the status of all defined virtual terminals.

Shows the status of the current terminal defined through the SETTERM command.

Example: EPI List Command

Screen returned from the command LIST ALL

VTAM		== RETRY ==		VTAM						
TERMINAME	USERNAME	STATUS	APPLNAME	LOGMODE	SECS	MAX	NOW	RTNCD	FDBK2	

OMTERM3	OMEGAMON	ACTIVE	OM/TAM	T3278MB	30	30	0	X'00'	X'00'	
OMTERM1	OM/TAM	ACTIVE	OM/TAM	T3278MB	30	30	0	X'00'	X'00'	
OMTERM2	OMCICS	ENABLED	OM/TAM	T3278MB	30	30	0	X'10'	X'01'	
CA7TERM	CA7	RETRYING	CA7		30	4	2	X'00'	X'00'	
OPSS0001	TSO1	ACTIVE	TSO	NO RETRY				X'00'	X'00'	
OPSS0002	TSO2	ACTIVE	TSO	NO RETRY				X'00'	X'00'	

LIST Return Codes

The LIST command produces these return codes:

0

The command executed successfully.

4

No terminals were defined.

EPI LOGOFF—Log Off a Virtual Terminal

The LOGOFF command logs a virtual terminal off from an external product. The terminal remains active unless you also issue the EPI DISABLE command.

When issuing the LOGOFF command, you also need to log off from the external product associated with this virtual terminal. For example, if a virtual terminal were logged on to TSO, you would log off from TSO by exiting to the TSO READY prompt, and then issuing the TSO LOGOFF command.

This command has the following format:

```
ADDRESS EPI "LOGOFF keywords"  
      {termname|ALL|*}
```

termname

Defines the name of the virtual terminal you are logging off.

ALL

Instructs EPI to log off all virtual terminals.

Logs off the current terminal defined through the SETTERM command.

Example: EPI LOGOFF

The following log off virtual terminal with message prefixing turned off:

```
MSGID OFF  
LOGOFF TERM4
```

LOGOFF Return Codes

The LOGOFF command produces these return codes:

0

The command executed successfully.

4

The specified virtual terminal was not logged off because it has not been defined or is not logged on.

12

The LOGOFF command failed.

EPI LOGON—Log On a Virtual Terminal

The LOGON command logs a virtual terminal onto an external product by causing EPI to issue a VTAM REQSESS request.

This command has the following format:

```
ADDRESS EPI "LOGON keywords"  
  {termname|ALL|*}  
  [APPLID(prodappl)]  
  [LOGMODE(modename)]  
  [LOGONPARM("parmstring")]  
  [NORETRY|RETRY(seconds retries)]
```

The LOGON command uses most of the same keywords as the EPI CHANGE and EPI DEFINE commands. For descriptions of these keywords, see The EPI CHANGE Command in this chapter.

Example: EPI LOGON

The following logs on with message prefixing active:

```
MSGID ON  
LOGON TERM4
```

LOGON Return Codes

The LOGON command produces these return codes:

0

The command executed successfully.

4

The specified virtual terminal was not logged on because it has not been defined, is disabled, or is already logged on.

12

The LOGON command failed.

EPI MSGID—Prefix Message IDs

The MSGID command determines whether output lines that *all* EPI host commands return to the EPI external data queue have a prefix. This prefix, which begins with the characters OPS, consists of an eight-character message ID and a trailing blank.

This command has the following format:

```
ADDRESS EPI "MSGID keyword"  
{ON|OFF}
```

ON

Causes message IDs to be prefixed to output lines in the external data queue. ON is the default setting.

OFF

Causes output lines to return to the external data queue without prefixes.

Default: ON

Example: EPI MSGID

The following turns message prefixing off:

```
MSGID OFF
```

MSGID Return Codes

The MSGID command produces these return codes:

0

The command executed successfully, turning message prefixing ON.

1

The command executed successfully, turning message prefixing OFF.

EPI MVCURSOR—Move Cursor Position

This command moves the cursor to the named row and column position on the virtual terminal screen.

This command has the following format:

```
ADDRESS EPI "MVCURSOR keywords"  
/* You must specify a value for each keyword. */  
  {termname!}  
  {row}  
  {col}
```

termname

Defines the name of the virtual terminal for which you are specifying cursor position.

Moves the cursor for the current terminal defined through the SETTERM command.

row

Identifies the number of the row where you want to place the cursor.

col

Identifies the number of the column where you want to place the cursor.

Example: EPI MVSCURSOR

The following example moves the cursor to column 1 of row 1:

```
MVCURSOR TERM01 1 1
```

MVCURSOR Return Codes

The MVCURSOR command produces these return codes:

0

The command executed successfully.

4

The command did not execute. An error message reports whether the command failed because the named terminal is not defined or because the terminal is disabled.

12

The command specified an invalid row, column, or both.

EPI PEEK—Display Screen Buffer in Hexadecimal

Use this command to display the contents of the virtual terminal screen buffer in hexadecimal format. EPI displays 16 hexadecimal bytes for each output line except for row 0 of the screen. PEEK under EPI does not read the contents of row 0.

Usually, you issue this command to interrogate 3278 host attribute bytes. The PEEK command runs synchronously in AOF rules and returns data to the external data queue. The command returns buffer code in EBCDIC format; values ranging from X'20' through X'3F' are attribute bytes. You can read the screen more easily using the RDSCREEN and RDSCRROW commands of the EPI (described later in this chapter).

This command has the following format:

```
ADDRESS EPI "PEEK keywords"  
/* You must specify a value for each keyword. */  
  {termname}*  
  {row}  
  {col}  
  {length}
```

termname

Defines the virtual terminal for which you want to display buffer contents.

*

Specifies the current terminal defined through the SETTERM command.

row

Identifies the number of the screen row where capturing of buffer contents begins.

col

Identifies the number of the screen column where capturing of buffer contents begins.

length

Defines the number of lines from the screen buffer to be displayed.

Example: EPI PEEK

Suppose that you issue this PEEK command:

```
PEEK TERM4 22 1 30
```

You receive this output:

```
24 D9 C5 C1 C4 EB 20 00 00 00 00 00 00 00 00 00 *READY.....*  
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

This example shows the beginning of the TSO READY message on terminal TERM4. The first number, 24, is a high-intensity unprotected attribute byte. The five characters following it are READY in 3278 buffer code. The number following that, 20, is a low-intensity unprotected attribute byte.

PEEK Return Codes

The PEEK command produces these return codes:

0

The command executed successfully.

4

The virtual terminal is not defined or not enabled.

12

The command specifies an invalid row/column/length value. Either the row or the column is larger than the screen size or the length value extends past the end of the screen.

EPI POKE—Modify Screen Buffer Contents

Use this command to modify the contents of the virtual terminal screen buffer. POKE under EPI does not read the contents of row 0.

This command has the following format:

```
ADDRESS EPI "POKE keywords"  
/* You must specify a value for each keyword. */  
{termname}  
{row}  
{col}  
{poketext}
```

termname

Defines the virtual terminal for which you want to modify buffer contents.

Specifies the current terminal defined through the SETTERM command.

row

Specifies the number of the screen row where altering of buffer contents begins.

col

Specifies the number of the screen column where altering of buffer contents begins.

poketext

Provides a character string or a hexadecimal string enclosed in quotation marks. The value is the actual 3278 buffer code.

Example: EPI POKE

The following shows buffer code representing the string READY at column 2 of row 22:

```
POKE TERM4 22 2 "D9 C5 C1 C4 EB 20"x
```

POKE Return Codes

The POKE command produces these return codes:

0

The command executed successfully.

4

The command did not execute because the virtual terminal is not defined or is disabled.

12

The command failed, probably because it specifies an invalid row/column value. Either the row or the column is larger than the screen size.

EPI RDCURSOR—Request Cursor Position

Use the RDCURSOR command to return the cursor position, specified by row and column number. This command runs synchronously in AOF rules and returns data in the external data queue.

This command has the following format:

```
ADDRESS EPI "RDCURSOR keywords"  
      {termname*}
```

termname

Defines the virtual terminal for which you want to retrieve the cursor position.

*

Retrieves the cursor position for the current terminal defined by the SETTERM command.

Example: EPI RDCURSOR

The following requests the cursor position on virtual terminal TERM3:

```
RDCURSOR TERM3
```

If the cursor is at row 23, column 1, the output from this command is 23 1.

RDCURSOR Return Codes

The RDCURSOR command produces these return codes:

0

The command executed successfully.

4

The command did not execute because the virtual terminal is not defined or is disabled.

EPI RDSCREEN—Retrieve Virtual Terminal Screen

The RDSCREEN command returns the entire virtual terminal screen as a set of output lines. The first output line is the operator information area, which is the status line at the bottom of the screen. The remaining lines are the screen contents starting with the top row. If trimming is on, trailing blanks are trimmed.

This command runs synchronously in AOF rules and returns data in the external data queue.

This command has the following format:

```
ADDRESS EPI "RDSCREEN keywords"  
      {termname*}
```

termname

Defines the virtual terminal for which you want to retrieve the screen.

*

Retrieves the screen for the current terminal defined by the SETTERM command.

Example: EPI RDSCREEN

The following returns the screen image on virtual terminal TERM4:

```
RDSCREEN TERM4
```

RDSCREEN Return Codes

The RDSCREEN command produces these return codes:

0

The command executed successfully.

4

The command did not execute because the virtual terminal is not defined or is disabled.

EPI RDSCRROW—Return Text from Specified Row

This command returns a specified row as one output line. EPI does not allow the RDSCRROW command to read row 0, the operator information area. If trimming is on, trailing blanks are trimmed.

This command runs synchronously in AOF rules and returns data in the external data queue.

This command has the following format:

```
ADDRESS EPI "RDSCRROW keywords"  
    {termname*}  
    {row}
```

termname

Defines the virtual terminal from which you want to retrieve the row.

*

Retrieves the specified row for the current terminal defined by the SETTERM command.

row

Specifies the number of the row to be returned.

Example: EPI RDSCRROW

The following returns the text on line 24 of the screen image on virtual terminal TERM4:

```
RDSCRROW TERM4 24
```

RDSCRROW Return Codes

The RDSCRROW command produces these return codes:

0

The command executed successfully.

4

The command did not execute because the virtual terminal is not defined or is disabled.

12

The command specified an invalid row number.

EPI SESSCMD—Issue Command to VTAM Session

Use the SESSCMD command to issue a command to a VTAM application session. For SESSCMD to function, CA OPS/MVS must be monitoring the VTAM session.

This command has the following format:

```
ADDRESS EPI "SESSCMD keywords"
{cmdtext}
{SESSION(sessid)}
[CMDRESP(dest)]
[CMDWAIT(maxwait)]
[D(screenid)]
[MAXCMDOUT(maxlines)]
[PREFIX(prefix)]
[STOPMSG(text,text)]
[SUBSYS(ssid)]
[TRUNCATE{YES|NO}]
```

cmdtext

Specifies the text of the command to be issued to the VTAM session. It can be up to 1,000 bytes long. If the *cmdtext* contains quotes, you can specify the command in one of the following ways:

- Use two consecutive single quotes for each quote that will be passed to the application. For example, SESSCMD "'DSIRB11.ASM'" causes the text 'DSIRB11.ASM' to be set.
- Use a pair of characters other than question marks to enclose the text. For example, SESSCMD /'DSIRB11.ASM'/ causes the text 'DSIRB11.ASM' to be set.

As part of the command text you enter, include an abbreviation for an AID key, that is, one of the function keys on the 3270 keyboard. When CA OPS/MVS executes SESSCMD, it issues the command and sends the function keystroke you specified. If you do not enter a key abbreviation, CA OPS/MVS issues the specified command, and then sends an Enter keystroke.

To simulate an Enter keystroke, use this text:

```
SESSCMD "
```

Otherwise, the Enter keystroke is implied. A key abbreviation consists of the at-sign (@) followed by a lowercase letter, a digit, or uppercase C. The abbreviations correspond as much as possible to the actual name of the key. For example, the abbreviation for the CLEAR key is @C, the abbreviation for PF3 is @3, and so on. For a list of possible abbreviations, see Keyboard Mnemonics for Special Function Keys in this chapter.

When specifying an abbreviation for an AID key, enter the abbreviation exactly as shown in Keyboard Mnemonics for Special Function Keys in this chapter. Otherwise, SESSCMD cannot interpret the abbreviation you entered as a valid keystroke. The abbreviation must be the final item in your text, or else CA OPS/MVS issues this message:

```
OPS4386W  EXTRANEIOUS CHARACTERS IGNORED AFTER FF, PA OR CLEAR KEY
```

After this message appears, CA OPS/MVS continues to process the SESSCMD command processor. However, CA OPS/MVS sends only the command text that preceded the key abbreviation, plus the key abbreviation itself, to the VTAM session.

SESSION

Specifies the session ID of the VTAM application session.

CMDRESP

(Optional) Specifies the kind of variables to store the fetched screen image.

The dest value can be either CLIST or REXX.

Default: CLIST

CMDWAIT

(Optional) Specifies the maximum amount of time that SESSCMD waits before it times out. SESSCMD normally waits for the terminal keyboard unlock condition before it returns to the current screen image (unless it is used with the STOPMSG keyword).

When used with the STOPMSG keyword, SESSCMD tries to satisfy the STOPMSG request until the CMDWAIT value expires.

Default: 1 second, and the maximum value is 900 seconds.

ID

(Optional) Assigns a one- to eight-character ID to the screen image generated by the command. In screen rules, this ID identifies the command that produced the screen.

CA OPS/MVS assigns this ID to all responses from the application while the SESSCMD command processor is executing, regardless of whether those responses are responses to the command or to some other application action.

Default: No screen ID.

MAXCMDOUT

(Optional) Specifies the maximum number of lines that SESSCMD returns.

PREFIX

(Optional) Specifies the one- to six-character prefix to be used when creating the CLIST or REXX variables. For more details on these variables, see Using SESSCMD in a CLIST or REXX EXEC in this chapter.

STOPMSG

(Optional) Specifies a text string that CA OPS/MVS searches for as it receives the response from a command issued to a VTAM application session (through SESSCMD). If *text* matches any text in the response screen returned by the VTAM session, CA OPS/MVS terminates the SESSCMD. If no text in the response screen matches *text*, SESSCMD times out, producing an error message and return code.

You can specify up to ten text elements with the STOPMSG keyword. Each text element can use as many as 32 characters. If you specify more than one text element, you must separate those elements with commas as shown in the following example:

```
STOPMSG(text,text,text, ...)
```

Also, *text* can contain non-alphabetic characters. For example, many application screens terminate with three asterisks (***). So if you want SESSCMD to terminate when CA OPS/MVS detects the end of the response screen from the VTAM session, specify STOPMSG(***).

Notes:

- Exercise caution when using the STOPMSG keyword to search for a text string when a high value is specified for the CMDWAIT keyword—doing so may encumber your system resources.
- SESSCMD polls the terminal four times a second for updates to the screen until the CMDWAIT value expires or a STOPMSG string is found.

SUBSYS

(Optional) Addresses the command to a copy of CA OPS/MVS with a subsystem ID other than the default subsystem ID, which is OPSS.

TRUNCATE

(Optional) Specifies whether CA OPS/MVS truncates the screen image display when you enter SESSCMD from a TSO terminal. Possible TRUNCATE values are YES, which tells CA OPS/MVS to truncate this display to make it more readable; and NO, which tells CA OPS/MVS not to truncate this display. The default is YES.

Examples: SESSCMD Command

- This command gets TSO session TSOAA11 out of ISPF and back to the READY prompt:

```
SESSCMD 'X' SESSION(TSOAA11)
```

- The following shows the use of the SESSCMD command in a REXX program:

```
sess = 'A44EPC1'
/* GET THE CURRENT SCREEN IMAGE INTO STEM VARIABLES PREFIXED WITH LINE */
ADDRESS EPI
"GETSCRN SESSION('sess') CMDRESP(REXX) PREFIX(LINE)"
IF RC > 0 THEN
DO
  SAY "GETSCRN RETURN CODE =" RC
  EXIT
END
DO FOREVER
DO Z=1 TO LINE0
  LINEZ = SUBSTR(LINEZ,1,60)
END
ADDRESS WTO
"TEXTVAR(LINE) MSGID(*)"
"TEXT('CURSOR AT ROW' csrcol 'COLUMN' csrcol 'ENTER COMMAND',
  'OR EXIT) REPLY"
IF RC > 0 THEN EXIT
IF QUEUED() > 1 THEN EXIT
PARSE FULL . RPLY
"TEXT('OPERATOR REPLY IS: ' rply ')"
IF POS('EXIT',RPLY) > 0 THEN EXIT
ADDRESS EPI
"SESSCMD /rply/ SESSION('sess') CMDRESP(REXX) PREFIX(LINE)",
  "CMDWAIT(5)"
IF RC > 0 THEN
DO
  DO WHILE QUEUED() > 0
    PARSE FULL X; SAY X
  END
  EXIT
END
END
RETURN
```


Keyboard Mnemonics for Special Function Keys

The following is a list of function key abbreviations for use in the *cmdtext* operand under SESSCMD. Note that the uppercase and lowercase alphabetic characters are abbreviations for different keys.

@C Clear	@7 PF7	@e PF14	@I PF21
@1 PF1	@8 PF8	@f PF15	@m PF22
@2 PF2	@9 PF9	@g PF16	@n PF23
@3 PF3	@a PF10	@h PF17	@o PF24
@4 PF4	@b PF11	@i PF18	@x PA1
@5 PF5	@c PF12	@j PF19	@y PA2
@6 PF6	@d PF13	@k PF20	@z PA3

You also can insert special *escape* instructions in the text string to specify where on the screen the text should be placed. You can specify the following instructions:

Abbreviations	Instructions
@(xx,yy)	Go to line xx, column yy. (1,1) is the topmost left character on the screen.
@B	Tab backwards one field.
@F	Erase the End-of-File character. This replaces all remaining characters in the current field with blanks.
@L	Move the cursor left one position.
@N	Move the cursor down one line and to the first unprotected field that follows the current cursor position. If the cursor is on the last screen line, wrap to the home position on the screen.
@T	Tab forward one field.
@U	Move the cursor up one position.
@V	Move the cursor down one position.
@Z	Move the cursor right one position.
@O	Go to home (position (1,1)).
@-	Continuation feature. When this operator is entered at the end of the SESSCMD text, the automatic Enter key is not sent.

To include the @ sign as text, specify @@.

Note: CA OPS/MVS does not translate the text string to uppercase.

Use SESSCMD in a CLIST or REXX EXEC

When you use the SESSCMD command in a CLIST or a REXX EXEC, you can capture the response to the issued command in CLIST or REXX variables. A CLIST or REXX EXEC can then analyze the response and take the appropriate action. SESSCMD creates the output CLIST or REXX variables shown in the following table:

Variable Contents	Variable Name If CMDRESP(CLIST)	Variable Name If CMDRESP(REXX)
The column in which the cursor appears. The leftmost column of the screen is column 1.	CSRCOL	CSRCOL
The location of the cursor as a single number.	CSRPOS	CSRPOS
The row or line on which the cursor appears. The first line on the screen is line 1.	CSRROW	CSRROW
The number of lines up to the last non-blank line on the screen.	LINENO	LINENO
The number of lines in the returned screen image. LINE is the default prefix.	<i>prefix</i>	<i>prefix.0</i>
The text of line <i>m</i> in the returned screen image.	<i>prefixm</i>	<i>prefix.m</i>
The number of words in line <i>m</i> of the screen image.	<i>prefixmW</i>	<i>prefix.m.0</i>
The <i>n</i> th word in line <i>m</i> of the screen image. The value of <i>n</i> can be any integer from 1 to 4.	<i>prefixmWn</i>	<i>prefix.m.n</i>
The entire screen contents. This variable contains a string consisting of all of the lines from the screen laid end-to-end, starting from the left.	SCREEN	SCREEN
The total number of rows or lines (blank and non-blank) on the screen.	SCRLEN	SCRLEN
The total size of the screen. CA OPS/MVS generates this value by multiplying &SCREEN by &SCRWDTH.	SCRSIZE	SCRSIZE

Variable Contents	Variable Name If CMDRESP(CLIST)	Variable Name If CMDRESP(REXX)
The current state of the keyboard (LOCKED or UNLOCKED).	SCRSTAT	SCRSTAT
The total number of columns on the screen. An 80-byte screen has 80 columns.	SCRWIDTH	SCRWIDTH

EPI SESSCMD Return Codes

If SESSCMD executes in a script, some error conditions (listed in the following table) may cause the script to be canceled. The SESSCMD command produces one of these return codes:

4

A command syntax error occurred.

8

The EPI terminal, or session ID, is not defined.

12

The subsystem is inactive.

20

The screen is not available.

24

The EPI is inactive.

28

Protected field store.

36

The command timed out.

56

The EPI command send failed.

60

Authorization check failed.

64

An authorization check abend occurred.

ops--EPI SETMODEL—Set Model Number

This command sets the model number for the virtual terminal. If this terminal is enabled, the SETMODEL command internally disables the terminal, sets the model number, and then re-enables the terminal.

Important! We strongly recommend you do not use this command because it will be removed in a future release. This command does not actually set the model number for the virtual terminal. The type of the terminal is actually controlled by the value of the LOGMODE keyword on the EPI CHANGE, EPI DEFINE, or EPI LOGON command.

This command has the following format:

```
ADDRESS EPI "SETMODEL keywords"  
    {termname*}  
    {model}
```

termname

Defines the virtual terminal for which you are setting a model number.

Sets a model number for the current terminal defined by the SETTERM command.

model

Specifies the model number, which must be 2, 3, or 4.

Example: EPI SETMODEL

The following sets the terminal model for TERM3 to model 3:

```
SETMODEL TERMB 3
```

SETMODEL Return Codes

The SETMODEL command produces these return codes:

0

The command executed successfully.

4

The virtual terminal is not defined.

EPI SETTERM—Set the Virtual Terminal

This command sets the current virtual terminal. Subsequent commands that specify * as the *termname* value will refer to this terminal. When the REXX program begins, the current virtual terminal is undefined.

This command has the following format:

```
ADDRESS EPI "SETTERM keyword"  
      {termname}
```

termname

Defines the virtual terminal to be designated as the current terminal.

Example: EPI SETTERM

The following sets virtual terminal 5 as the current virtual terminal:

```
SETTERM TERM5
```

SETTERM Return Code

The SETTERM command produces this return code:

0

The command executed successfully.

EPI SETUNAME—Set User-Defined Name

This command sets the user-defined name for a virtual terminal. To cancel out the current user-defined name, omit the *user_termname* operand.

This command has the following format:

```
ADDRESS EPI "SETUNAME keyword"  
{termname}  
[user_termname]
```

termname

Defines the name of the virtual terminal that you are giving a user-defined name. This name must be the *termname* value defined for this terminal through the DEFINE or CHANGE command.

Sets a model number for the current terminal defined by the SETTERM command.

user_termname

(Optional) Specifies the name that you use for this virtual terminal.

Example: EPI SETUNAME

The following sets user-defined name for virtual terminal TERM3 to CICSTERM:

```
SETUNAME TERM3 CICSTERM
```

SETUNAME Return Codes

The SETUNAME command produces these return codes:

0

The command executed successfully.

4

The command did not execute because the virtual terminal name is not defined.

12

The *user_termname* value is already defined.

EPI SUBATTR—Return Attribute Characters

The SUBATTR command affects how the RDSCREEN and RDSCRROW commands return host attribute characters. Use this command to return all attribute bytes as values between X'20' and X'3F' (no translation).

This command has the following format:

```
ADDRESS EPI "SUBATTR keyword"  
{ONOFF}  
[sub_char]
```

ON

Tells EPI to translate attribute characters. This value reverses the effects of a previously issued SUBATTR OFF command.

OFF

Tells EPI not to translate attribute characters.

Default: OFF

sub_char

(Optional) Specifies the character to be substituted for all host attribute byte characters of output from the RDSCREEN and RDSCRROW commands. This value can be a single character or a numeric value between 0 and 255 that corresponds to the EBCDIC value of the desired character.

Example: EPI SUBATTR

The following sets substitute character for host attribute byte characters to a percent sign:

```
SUBATTR %ON
```

SUBATTR Return Codes

Return codes from the SUBATTR command represent the EBCDIC equivalents of the previous SUBATTR value. You can use a subroutine to alter the SUBATTR value temporarily, and then restore the environment to its former state.

EPI SUBSYS—Direct Host Commands

The SUBSYS command directs EPI host commands from the current OPS/REXX program to an EPI other than the one using the CA OPS/MVS default z/OS subsystem ID of OPSS. Once the SUBSYS command is issued, all subsequent EPI host commands from the current REXX program go to this subsystem until the program encounters another EPI SUBSYS command.

This command has the following format:

```
ADDRESS EPI "SUBSYS keyword"  
      {ssid}  
ssid
```

Defines the four-character z/OS subsystem ID of the copy of CA OPS/MVS.

Example: EPI SUBSYS

The following sets Subsystem ID to OPSV:

```
SUBSYS OPSV
```

Return Codes

The EPI SUBSYS command produces only the return codes described in ADDRESS EPI Return Codes in REXX Programs.

More information:

[ADDRESS EPI Return Codes in REXX Programs](#) (see page 84)

EPI SUBUNPT—Set Substitute Character for Unprintable Characters

The SUBUNPT command affects how the RDSCREEN and RDSCRROW commands return unprintable characters.

This command has the following format:

```
ADDRESS EPI "SUBUNPT keywords"  
    {sub_char}  
    {ONOFF}
```

sub_char

Defines the character to be substituted for all unprintable characters of output from the RDSCREEN and RDSCRROW commands. This value can be a single character or a numeric value between 0 and 255 that corresponds to the EBCDIC value of the desired character.

ON

Tells EPI to translate unprintable characters. This value reverses the effects of a previously issued SUBUNPT OFF command.

OFF

(Default) Tells EPI not to translate unprintable characters.

Example: EPI SUBUNPT

The following sets the substitute character for unprintable characters to a slash and turns translation for these characters on:

```
SUBUNPT /ON
```

SUBUNPT Return Codes

Return codes from the SUBUNPT command represent the EBCDIC equivalents of the previous SUBUNPT value. You can use a subroutine to alter the SUBUNPT value temporarily, and then restore the environment to its former state.

EPI TIMEOUT—Set Timeout Value

The TIMEOUT command determines how long EPI commands wait before timing out. When a command times out, the external data queue receives this message:

```
EPI COMMAND TIMED OUT BEFORE ALL RESPONSES RECEIVED
```

This command has the following format:

```
ADDRESS EPI "TIMEOUT keyword"  
{seconds}
```

seconds

Defines the number of seconds, between 1 and 86400, to wait before EPI commands time out.

Default: 60 seconds

Example: EPI TIMEOUT

The following sets the timeout value to 45 seconds:

```
TIMEOUT 45
```

TIMEOUT Return Codes

The TIMEOUT command produces only the return codes described in ADDRESS EPI Return Codes in REXX Programs.

More information:

[ADDRESS EPI Return Codes in REXX Programs](#) (see page 84)

EPI TRACE—Set Trace Value

The TRACE command turns tracing of terminal activity on and off. When tracing is on, EPI writes hardcopy messages to display RPLs and buffers being sent and received.

This command has the following format:

```
ADDRESS EPI "TRACE keywords"  
{ON|OFF}  
{termname*}
```

ON

Turns terminal tracing on.

OFF

(Default) Turns terminal tracing off.

termname

Defines the name of the virtual terminal for which you are setting the trace.

*

Traces the current terminal as defined by the SETTERM command.

Example: EPI TRACE

The following sets terminal tracing on:

```
TRACE term1 ON
```

TRACE Return Codes

The TRACE command produces these return codes:

0

The command executed successfully.

4

The command did not execute. A warning message explains why, either because no terminals were found to trace or because the specified terminal name could not be found.

EPI TRIM—Remove Leading and Trailing Blanks

The TRIM command causes all leading and trailing blanks to be kept on output.

Important! We strongly recommend you do not use this command since it will be removed in a future release. This command does not actually turn trimming on or off. Trimming is always off for EPI.

This command has the following format:

```
ADDRESS EPI "TRIM keyword"  
{ON|OFF}
```

ON

Turns terminal trimming on.

OFF

(Default) Turns terminal trimming off.

Example: EPI TRIM

The following sets terminal trimming on:

```
TRIM ON
```

TRIM Return Codes

The TRIM command produces these return codes:

0

The command executed successfully, changing the setting from OFF to ON.

1

The command executed successfully, changing the setting from ON to OFF.

EPI TYPE—Simulate Typing

Use the TYPE command to simulate typing at a 3278 keyboard. If the value of the EPICMDLOGGING parameter is YES, EPI writes to OPSLOG any text sent to a virtual terminal through the TYPE command.

This command has the following format:

```
ADDRESS EPI "TYPE keywords"  
    {termname}  
    {text}
```

termname

Defines the terminal to which you are sending text.

text

Defines the text to send to the external product associated with this virtual terminal. This text combines literal text enclosed in single quotation marks (such as the text of a command) and the names of keys on the 3278 keyboard (such as Enter).

Note: For more information on using the TYPE command, see the *User Guide* and the *Command and Function Reference*.

Example: EPI TYPE

The following sends virtual terminal TERM1 a character sequence that moves the cursor to the first input field on the screen, erases its contents, enters the characters =X, and then issues the ENTER keystroke:

```
TYPE TERM1 HOME!ERASE_EOF=X!ENTER
```

TYPE Return Codes

The TYPE command produces these return codes:

0

The command executed successfully.

4

The command did not execute. A warning message explains why, either because the terminal is not defined or because it is not enabled.

12

The command tried to issue an invalid keystroke. This code often results when the keyboard is inhibited from accepting input or when the command tries to enter text into a protected field on the screen.

3278 Host Key Names

The key equivalents described in the following table are for use with the TYPE and TYPESEC commands. There are 87 keys on the real 3278 keyboard; the following four of them, however, are not accessible from a virtual keyboard:

- ALT
- SHIFT_KEY_L
- SHIFT_KEY_R
- SHIFT_LOCK

Keys that are the same on a PC keyboard and a 3278 keyboard (alphanumeric keys and some symbols) are accessed as usual.

The others are accessed through the HOSTKEYS command, which defines about 160 OPS/REXX variables to simulate the 3278 keyings that a PC keyboard does not duplicate. Each variable consists of an exclamation point (!) followed by a variable name describing the host key. Thus, !A has not been assigned by HOSTKEYS (key A as usual), but !ALT_A has been assigned.

3278 Host Key	Simulated by Typing...
lowercase alphabetic keys (a-z)	a-z
uppercase alphabetic keys (A-Z)	A-Z - To simulate the Alt versions of these keys, type !ALT_x where x is the desired key. For example, to simulate Alt-A type !ALT_A.
numeric keys (0-9)	0-9
APL key	!APL
CLEAR	!CLEAR
Alt-Cursor	!ALT_CURSOR
DEV CNCL	!DEV_CNCL
Home key	!HOME
IDENT key	!IDENT
PA1	!PA1
PA2	!PA2
SYS_REQ	!SYS_REQ
TEST key	!TEST
@	@
#	#

3278 Host Key	Simulated by Typing...
\$	\$
%	%
&	&
*	*
(	(
)	)
-	-
=	=
_	_
+	+
<	< or !LT
>	> or !GT
{	{
}	}
/	/
\	\
?	?
!	!
.	.
,	,
:	:
;	;
'	' or !QUOTE1
"	" or !QUOTE2
backquote key	!BKQUOTE
Alt-¢ cent sign	!ALT_CENT_SIGN
~	!TILDE
	!VERTICAL_BAR
¬	!NOT_SIGN
ATTN	!ATTN

3278 Host Key	Simulated by Typing...
backspace key	!BKSPACE
backtab key	!BACKTAB
clicker key	!CLICKER
CURSR BLINK	!CURSR_BLINK or !CSR_BLN
CURSR SEL	!CURSOR_SEL or !CSR_SEL
DUP key	!DUP
FIELD MARK key	!FIELD_MARK or !FLD_MRK
printscreen key	!PRT_SCR
Reset key	!RESET - You can simulate the shifted versions of these keys by entering !SHF_xxxx, where xxxx is the OPS/REXX variable name for the simulated key. For example, to simulate Shift-ATTN, enter !SHF_ATTN.
PF keys (PF1-PF24)	!PF1-!PF24
down arrow	!D_ARROW
left arrow	!L_ARROW
right arrow	!R_ARROW
up arrow	!U_ARROW
cent sign	!CENT_SIGN
DELETE key	!DELETE
ENTER key	!ENTER
ERASE EOF	!ERASE_EOF or !ERS_EOF
ERASE INPUT	!ERASE_INPUT or !EI
Insert key	!INSERT
RETURN key	!RETURN
Space bar	!SPACE
Tab key	!TAB
Unused key	!UNUSED_KEY or !UK - To simulate the shifted and Alt versions of these keys, enter ! SHF_xxxx or ! ALT_xxxx, where xxxx is the OPS/REXX variable name for the simulated key. For example, to simulate Shift-Insert, enter !SHF_INSERT; to simulate Alt-Insert, type !ALT_INSERT.

EPI TYPESEC—Simulate Typing without Displaying in OPSLOG

Use the TYPESEC command to simulate typing at a 3278 keyboard *without* writing the typed text to OPSLOG. For example, you can use the TYPESEC command to keep passwords from appearing in OPSLOG.

```
ADDRESS EPI "TYPESEC keywords"  
  {termname}*  
  {text}
```

termname

Defines the terminal to which you are sending text.

*

Sends the text to the current virtual terminal.

text

Defines the text to be sent to the external product associated with this virtual terminal. This text combines literal text enclosed in single quotation marks (such as the text of a command) and the names of keys on the 3278 keyboard (such as Enter).

Note: For more information on using the TYPESEC command, see the *User Guide* and the *Command and Function Reference*.

Example: Character sequence sent to a virtual terminal

This example sends virtual terminal TERM1 a character sequence that moves the cursor to the first input field on the screen, erases its contents, enters a password, and then issues an ENTER keystroke:

```
TYPESEC TERM1 HOME!ERASE_EOFsecret!ENTER
```

TYPESEC Return Codes

The TYPESEC command produces these return codes:

0

The command executed successfully.

4

The command did not execute. A warning message explains why, either because the terminal is not defined or because it is not enabled.

12

The command tried to issue an invalid keystroke. This code often results when the keyboard is inhibited from accepting input or when the command tries to enter text into a protected field on the screen.

EPI TYPETEST—Test a TYPE Command

Use the TYPETEST command to test-run a TYPE command. Instead of simulating typing at a 3278 keyboard, the TYPETEST command returns the keystrokes as output lines, one per key.

This command has the following format:

```
ADDRESS EPI "TYPETEST keywords"  
  {termname}  
  {text}
```

termname

Defines the terminal to which you are sending text.

Sends the text to the current virtual terminal.

text

Defines the text to send to the external product associated with this virtual terminal. This text combines literal text enclosed in single quotation marks (such as the text of a command) and the names of keys on the 3278 keyboard (such as Enter).

Example: TYPETEST Command

This TYPETEST command sends the current virtual terminal a character sequence that moves the cursor to the home position on the screen, tabs backward, erases the screen text there, enters the text K E,D and then presses the Enter key.

```
TYPETEST *HOME!BACKTAB!ERASE_EOFK ED!ENTER
```

The TYPETEST command then returns this output:

```
0 04 HOME
0 0C BACKTAB
0 08 ERASE_EOF
0 7D '
0 D2 K
0 40
0 C5 E
0 6B ,
0 C4 D
0 7D '
0 7D ENTER
```

The first, one-digit field may be 0 for a normal key, 1 for a shifted key, or 2 for a key pressed while the ALT key is held down. The two-digit field is the EPI scan code (useful for debugging). The last field is the EPI host key name.

TYPETEST Return Codes

The TYPETEST command produces these return codes:

0

The command executed successfully.

4

The command did not execute. A warning message explains why, either because the terminal is not defined or because it is not enabled.

12

The command tried to issue an invalid keystroke. This code often results when the keyboard is inhibited from accepting input or when the command tries to enter text into a protected field on the screen.

EPI UNBIND—Undo a Bind

Use this command to reverse the effect of a BIND command and release the indicated session for access by other sources.

This command has the following format:

```
ADDRESS EPI "UNBIND keywords"  
    {ssid}  
    [,SUBSYS(ssid)]
```

ssid

Defines the session ID for the VTAM application session that should no longer be bound.

,SUBSYS(*ssid*)

(Optional) Addresses the command to a copy of CA OPS/MVS with a subsystem ID other than the default subsystem ID.

Default subsystem ID: OPSS

Example: EPI UNBIND

This example releases the CICS2 session so that other TSO users or automation procedures can bind to it:

```
UNBIND CICS2
```

UNBIND Return Codes

The UNBIND command produces these return codes:

0

The command completed successfully.

4

A syntax error occurred. CA OPS/MVS will generate messages describing the error.

8

The EPI terminal, or session ID, is not defined.

12

The CA OPS/MVS subsystem is not active.

16

The session is not available.

40

There is no response from the main subsystem.

EPI WAIT—Delay Rule Processing

Use the WAIT command in a REXX program or an AOF request rule to delay processing of that rule or program for a specified number of seconds. Request rules are the only rules that can use this command.

This command has the following format:

```
ADDRESS EPI "WAIT keyword"  
      {seconds}
```

seconds

Defines the wait interval in seconds. You can specify any number of seconds between 0 and 86400 (the number of seconds in 24 hours). The number can be either an integer or a real number with two places to the right of the decimal point.

Example: EPI WAIT

The following pauses processing for 5 seconds:

```
WAIT 5
```

WAIT Return Code

The WAIT command produces this return code:

0

The command executed successfully.

EPI WAITTOD—Set Processing Delay Time

Use the WAITTOD command to delay processing of a REXX program or an AOF request rule until a specific time of day has been reached. Request rules are the only rules that can use this command.

This command has the following format:

```
ADDRESS EPI "WAITTOD keyword"  
      {hh mm ss}
```

hh mm ss

Defines the wait time of day in hours (*hh*), minutes (*mm*), and seconds (*ss*). You must specify the time in military format (for example, 8:15:30 a.m. equals 081530 in military time). The time you specify also must be at least one second, and not more than 24 hours, into the future.

Example: EPI WAITTOD

The following pauses processing until 1:35 p.m.:

```
WAITTOD 13 35 0
```

WAITTOD Return Codes

The WAITTOD command produces these return codes:

0

The command executed successfully.

12

The time specified is not within 1 to 86400 seconds into the future.

ADDRESS HWS Commands

The ADDRESS HWS host command environment requires the CA Common Services Hardware Interface Service (HiSrv) to interact with IBM's BCPii which allows you to perform the following tasks:

- Execute hardware commands through BCPii.
- Retrieve system attributes through BCPii.
- Update system attributes through BCPii.

ADDRESS HWS requires that the INITHWS parameter must be set to YES. ADDRESS HWS is *only* available in an OPS/REXX program and an AOF Request (REQ) rules environment.

Note: For more information about how to install and configure the Hardware Interface Service (HiSrv), see the *CA Common Services for z/OS Installation Guide*.

ADDRESS HWS Command Format

The ADDRESS HWS command provides operations to do the following operations:

- Send a command to HiSrv.
- Get the value of a system attribute from HiSrv.
- Update the value of a system attribute through HiSrv.

This command has the following format:

```
ADDRESS HWS "operation parameters"  
{SEND CMD}  
{GET ATTR}  
{SET ATTR}
```

HWS SENDCMD—Sends command to HiSrv

The ADDRESS HWS SENDCMD command lets you execute a specific hardware command for a specific target.

This command has the following format:

```
ADDRESS HWS "SENDCMD keywords"  
/* required keywords */  
{CMDTEXT(full command)}  
{ENTITY(list of entities)}  
/* optional keywords */  
[DEBUG(YES|NO)]  
[TIMEOUT(value)]
```

CMDTEXT

Contains the full text command that you want to execute through HiSrv. Commands are formatted for the Hardware Interface Service. A complete list of commands and their functions is available in *CA Common Services for Z/OS Reference GuideHardware*.

COMMON KEYWORDS

ENTITY, DEBUG and TIMEOUT are keywords common across every HWS command, and are defined in section [ADDRESS HWS Common Keyword](#) (see page 145).

Examples: HWS Connect

This example sends a command to deactivate LPAR SAMPLE1, because SAMPLE1 is not a unique LPAR name the CPC must be specified. In this case IBM.MACHINE.SAMPLE:

```
ADDRESS HWS "SENDCMD CMDTEXT('DEACTIVATE FORCE=YES') ENTITY(LPAR('SAMPLE1')  
CPC('IBM.MACHINE.SAMPLE'))"
```

This example executes the operating system command: *command* on LPAR SAMPLE2, which is uniquely named in the topology. It also prints debugging messages:

```
ADDRESS HWS "SENDCMD CMDTEXT('OSCMD PRIORITY=HIGH COMMAND=command')  
ENTITY(LPAR('SAMPLE2')) (DEBUG(YES))"
```

Return information from SENDCMD

The REXX variables HWSRC and HWSRS contain the address environment return and reason codes respectively. When a hardware command is executed, a hardware event notification is created. Validation and/or error reporting must be done by catching these events, see *Hardware Event API Rules* in *CA OPS/MVS AOF Rules User Guide*.

HWS GETATTR—Get the value of a system attribute

The ADDRESS HWS GETATTR command lets you retrieve the value of a single attribute from a specific target.

This command has the following format:

ADDRESS "HWS GETATTR keywords"

*/*required keywords */*

{ATTR(name of attr)}

{ENTITY(list of entities)}

/ optional keywords */*

[DEBUG(YES|NO)]

[REFRESH(YES|NO)]

[TIMEOUT(value)]

ATTR

ATTR specifies the name of the target attribute to be updated. These names are derived from the attribute names used by BCPii.

REFRESH

REFRESH defaults to NO, if REFRESH(YES) is specified HiSrv calls to BCPii to get the updated value.

COMMON KEYWORDS

ENTITY, DEBUG and TIMEOUT are keywords common across every HWS command, and are defined in section [ADDRESS HWS Common Keyword](#). (see page 145)

Examples: HWS GETATTR

This example gets the value of the attribute CPCVERSION and returns it in the REXX variable HWS.CPCVERSION. The command forces BCPii to go out and get the most updated value, times out after five seconds, and creates debugging messages.

```
ADDRESS HWS "GETATTR ATTR(CPCVERSION) ENTITY(CPC('SAMPLE.IBM.CPCNAME')) DEBUG(YES)
REFRESH(YES) TIMEOUT(5)"
```

Return information from GETATTR

The REXX variables HWSRC and HWSRS contain the address environment return and reason codes respectively. A REXX variable containing the attribute value is returned. The variable has the name structure HWS.attribute_name where the attribute name is the character string that is specified in the ATTR keyword. For example, a request for the attribute CPCNAME would be returned in the REXX variable HWS.CPCNAME. The format of the value in the variable is different depending on the data type of the variable. Attribute data types include:

Binary

This data type is a fixed or variable length binary values, they are returned as hex value strings.

Boolean

True and False values that are returned as either the character string 'TRUE' or the character string 'FALSE'.

Character Strings

This data type is a fixed or variable length strings of characters and is returned as such.

Handler

A BCPII attribute type that is not supported so a hex representation of the binary attribute value is returned.

IP Address

A BCPII attribute type that is not supported so a hex representation of the binary attribute value is returned.

Signed Binary Number

Returned as a signed integer value in the range -2,147,483,648 through 2,147,483,647.

Unsigned Binary Number

Returned as a positive integer value in the range 0 through 4,294,967,295.

Unknown Binary Value

The type for a BCPII attribute that is undocumented or not identifiable. Unknown binary is not supported so a hex representation of the binary attribute value is returned.

Note: All attributes are returned in the format that is required for the ATTRVAL keyword in the SETATTR call.

HWS SETATTR— Update the value of a system attribute

The ADDRESS HWS SETATTR command lets you update the value of a single attribute for a specific target.

This command has the following format:

```
ADDRESS HWS "SETATTR keywords"  
/* required keywords */  
{ATTR(name of attr)}  
{ATTRVAL(new attr value)}  
{ENTITY(list of entities)}  
/* optional keywords */  
[DEBUG(YES|NO)]  
[TIMEOUT(value)]
```

ATTR

ATTR specifies the name of the target attribute to be updated. These names are derived from the attribute names used by BCPii.

ATTRVAL

ATTRVAL contains the value the specified attribute should be updated to. Attribute values have several different types including:

Binary

This data type is a fixed or variable length binary values, they are entered as either hex values or hex strings. For example: ATTRVAL(X'A23F4D0F') or ATTRVAL(A43B5CFF)

Boolean

True and False values that are set using the character strings 'TRUE' and 'FALSE'. For example: ATTRVAL(FALSE)

Character Strings

This data type is a fixed or variable length strings of characters and are specified in ATTRVAL as such. For example: ATTRVAL(string) or ATTRVAL('another string')

Handler

A BCPii attribute type that is not supported by SETATTR.

IP Address

A BCPii attribute type that is not supported by SETATTR.

Signed Binary Number

A signed integer value in the range -2,147,483,648 through 2,147,483,647. This value requires a leading sign for negative numbers. For example: ATTRVAL(-34523) or ATTRVAL(4)

Unsigned Binary Number

A positive integer value in the range 0 through 4,294,967,295. For example:
ATTRVAL(657874)

Unknown Binary Value

The data type for a BCPii attribute is undocumented or not identified. Unknown binary is not supported by SETATTR.

COMMON KEYWORDS

ENTITY, DEBUG and TIMEOUT are keywords common across every HWS command, and are defined in section [ADDRESS HWS Common Keyword](#) (see page 145).

Example: HWS SETATTR

This example sets the value of the attribute CPCVERSION to 15, it also creates debugging messages:

```
ADDRESS HWS "SETATTR ATTR(CPCVERSION) ATTRVALUE(15)  
ENTITY(CPC(SAMPLE.IBM.CPCNAME)) DEBUG(YES)"
```

This example sets the value of the attribute ERRSTAT to the boolean value true:

```
ADDRESS HWS "SETATTR ATTR(ERRSTAT) ATTRVALUE(TRUE)  
ENTITY(CPC(SAMPLE.IBM.CPCNAME))"
```

Return information from SETATTR

The REXX variables HWSRC and HWSRS contain the address environment return and reason codes respectively.

ADDRESS HWS Common Keywords

ENTITY

Entities are specific components of a systems topology and allow you to target different sections of the topology with commands. The system topology is a hierarchical tree with multiple possible nodes for each entity type. For example, there could be many LPAR nodes under their respective CPC nodes. Each system has a unique topology which can be viewed by interacting directly with HiSrv.

One entity can be specified for each entity type and one or more entity types can be provided in ENTITY(). The order of the entities does not matter and the entity that is the lowest in the topology is assumed to be the target entity. A single entity can be specified if the entity's name is unique across the entire topology for entities of that type. In the case where a non-unique name is given for an entity, the name of its parent must also be provided for a unique identify.

The entity keyword takes the following form:

```
ENTITY({TYPE(entity name)} {TYPE(entity name)} ... )
```

Possible TYPEs are listed:

INSTALLATION

The highest targetable level in the topology. The parent for this entity type is the root and is never specified.

ENSEMBLE

Used to specify all the machines in a given area. The parent of this entity type is INSTALLATION.

CPC

Specifies a specific Central Processing Complex. The parent of this entity type is ENSEMBLE.

LPAR

Specifies a specific logical partition. The parent of this entity type is CPC.

CAPREC

Specifies a specific Capacity Record. The parent of this entity type is CPC.

PROFILES

This type currently only acts as a parent for the next three entity types. The parent of this entity type is CPC.

RAP

Specifies a specific Reset Activation Profile. The parent of this entity type is PROFILES.

IAP

Specifies a specific Image Activation Profile. The parent of this entity type is PROFILES.

LAP

Specifies a specific Load Activation Profile. The parent of this entity type is PROFILES.

TIMEOUT

TIMEOUT is an optional keyword that specifies the maximum number of seconds to wait for a single HiSrv request. TIMEOUT is a value from 0 through 21474836. If TIMEOUT is not provided, a default value of 120 seconds is used. If 0 is specified, HiSrv interactions will never timeout.

Note: This value is not the maximum time for the entire HWS command but rather is used separately for each individual call to HiSrv.

DEBUG

Debug is an optional keyword that can be set to either YES or NO. If DEBUG is not specified the value defaults to NO. If a value of YES is provided, more debugging messages are created with the message IDs OPS0349T, OPS1946E, or OPS341S.

ADDRESS HWS Return and Reason Codes in REXX Programs

When an error occurs, regardless of the value of `DEBUG` an error message with the ID OPS0341S is created. This error message contains the return and reason code. These codes are also placed into REXX variables. When any ADDRESS HWS command executes in a REXX program, the variable `HWSRC` and `HWSRS` in that program contains the return and reason codes. The return code is:

0

The command has executed successfully.

>0

The host command issuing the HWS command contained errors.

8

The command has failed, and an error message was issued.

12

A nonzero return code was received from HiSrv. The first two bytes of the reason code were set and an error message was issued.

16

The host environment has abended while processing the HWS command.

The reason code is a hex value where the first two bytes contain the HiSrv reason code or zeros if there is no HiSrv error. The third byte is zeros, and the fourth byte contains one of the following reason codes:

X'10

Internal error accessing control blocks. Contact CA support and provide the full return and reason codes.

X'11'

The INITHWS parameter was not set.

X'12'

Attempting to run from within a rule.

X'20'

Internal parsing error. Contact CA support and provide the full return and reason codes.

X'21'

Internal parsing error. Contact CA support and provide the full return and reason codes.

X'22'

Internal parsing error. Contact CA support and provide the full return and reason codes.

X'23'

Unrecognized attribute name.

X'24'

The attribute type was not supported.

X'30'

HiSrv connection error. Contact CA support and provide the full return and reason codes.

X'31'

No entity was specified.

X'32'

HiSrv internal communication error. Contact CA support and provide the full return and reason codes.

X'33'

Parent entity was not specified.

X'34' X'35'

HiSrv error connecting to a target entity. Contact CA support and provide the full return and reason codes.

X'36'

HiSrv disconnect error.

X'37'

HiSrv GETATTR error.

X'38'

HiSrv SETATTR call error.

X'39'

HiSrv SENDCMD call error.

X'40'

Internal error, invalid attribute length. Contact CA support and provide the full return and reason codes.

X'41'

An internal REXX variable error. Contact CA support and provide the full return and reason codes.

X'42'

An internal REXX variable error. Contact CA support and provide the full return and reason codes.

ADDRESS HWS Security

The Hardware Interface Service (HiSrv) handles all security with external SAF calls to determine the authority of a user that is associated with the calling address space. If the call is made in a cross-memory environment, the user of the address space where the call originated is used.

For more information on a HiSrv security configuration, see: *Hardware Interface Service Component Guide*.

ADDRESS LXCON Commands

The ADDRESS LXCON host command environment requires the CA Common Services Mainframe Connector for Linux on System Z be operational as documented in the CA Common Services bookshelf. The OPS/MVS Parameter Guide describes all associated Linux Connector Interface parameters and how OPS/MVS interfaces with CA Mainframe Connector for Linux on System Z.

The ADDRESS LXCON is an OPS/REXX host command environment that allows you to perform the following actions:

- Send a command to Linux or VM operating systems.
- (Optionally) Receive a response from Linux or VM operating systems. Command output from any ADDRESS LXCON command returns only when the STEM keyword is specified.
- List the properties of selected VM and Linux systems that are connected to the Linux Connector component.

Linux and VM commands are sent to the USS server where they are processed and sent to the Linux Connector component that routes the commands to the desired Linux or VM system. The response lines are returned to the issuing OPS/REXX program in REXX stem variables.

Since the ADDRESS LXCON VM and Linux command processing uses the USS server for command execution, it shares the ADDRESS USS Return Codes.

For More Information:

[ADDRESS USS Return Codes](#) (see page 398)

For more information:

[Keywords Common to All ADDRESS USS Commands](#) (see page 397)

ADDRESS LXCON LXCMD—Send Commands To Linux System

The LXCMD command issues any Linux command that can be executed from a shell environment.

This command has the following format:

```
ADDRESS LXCON "LXCMD keywords"  
{COMMAND('cmd text')}  
{LXNAME(linux name)}  
{VMNODE(vm name)}  
[LOG(Y|N)]  
[STEM(stem)]  
[WAIT(seconds)]  
[DELAY(seconds)]
```

COMMAND

This required keyword defines the text of the LINUX command. This keyword can be up to 2048 characters in mixed case.

LXNAME

Defines the name of a Linux guest to which you want to send the command.

VMNODE

Defines a name of VM system under which Linux guest, specified by the LXNAME parameter, is running.

LOG, STEM, WAIT, and DELAY

(Optional) Specify the LOG, STEM, WAIT, and DELAY keywords. These keywords have the same meaning as the ADDRESS USS Common Keywords.

ADDRESS LXCON VMCMD—Send Commands To VM System

The VMCMD command issues a VM command that can be executed from a VM command line.

This command has the following format:

```
ADDRESS LXCON "VMCMD keywords"  
{COMMAND('cmd text')}  
{VMNODE(vm name)}  
[LOG(Y|N)]  
[STEM(stem)]  
[WAIT(seconds)]  
[DELAY(seconds)]
```

COMMAND

This required keyword defines the text of the VM command. This keyword can be up to 2048 characters in mixed case.

VMNODE

Defines the name of a VM system to which you want to send the command.

LOG, STEM, WAIT, and DELAY

(Optional) Specify the LOG, STEM, WAIT, and DELAY keywords. These keywords have the same meaning as the ADDRESS USS Common Keywords.

Example: Issue command to a Linux system

This command lists active processes on a Linux system that is a VM guest machine and returns the response lines in stem variables with the stem name LXCMD. LXCMD.0 is set to the number of lines returned. The command waits up to 30 seconds for the response.

```
ADDRESS LXCON "LXCMD LXNAME(LINUX113) VMNODE(ZVMD02) COMMAND(ps -e) STEM(LXCMD)  
WAIT(30)"
```

For more information:

[Keywords Common to All ADDRESS USS Commands](#) (see page 397)

ADDRESS LXCON LIST—Return VM And Linux System Properties

The LIST command returns formatted, word delimited records of the properties of all or selected VM and Linux systems that are connected to the Linux Connector component. The records are returned in stem variables. This command is always executed synchronously without any waits or the services of the USS server.

This command has the following format:

```
ADDRESS LXCON "LIST keywords"
```

```
[VMNODE(vm name |N/A)]
```

```
[LXNAME(linux name |N/A)]
```

```
[STEM(stem)]
```

VMNODE

(Optional) Filters selection of the VM system records by a full or partial VM node name or the value N/A to exclude all VM systems. Partial VM node names are specified as a prefix with a trailing '*'. If VMNODE is not specified, all VM system records are returned.

LXNAME

(Optional) Filters selection of Linux system records by a full or partial Linux system name or the value N/A to exclude all Linux systems. Partial Linux system names are specified as a prefix with a trailing '*'. If LXNAME is not specified, all Linux system records are returned.

STEM

(Optional) Output from this command is always returned in stem variables. Specify the keyword only, if the default value is not acceptable.

Default stem

```
LXLST
```

Created Stem Variable For A VM System Record

The output stem variable record for a VM system has the following word delimited fields. Some field values contain blanks that have been translated to hexadecimal X'FE'. Translate these values to restore the original values.

VM system record ID

Always contains 'VMTP'

CP node name

Contains one through eight characters

CMS node name

Contains one through eight characters

Time zone seconds offset from GMT

Contains an integer

CP system ID

Contains the first line of 'Q CPLEVEL' VM command response

CP version number

Contains an integer

CP release level

Contains an integer

CP modification level

Contains an integer

CP service level

Contains integer

CMS system ID

Contains the first line of 'Q CMSLEVEL' VM command response

CMS level

Contains an integer

CMS service level

Contains an integer

Created Stem Variable For A Linux System Record

The output stem variable record for a Linux system has the following word delimited fields. Some field values contain blanks that have been translated to hexadecimal X'FE'. Translate these values to restore the original values.

Linux system record ID

Always contains 'LXTP'

Linux system name

Contains a character string

VM guest node name

Contains one through eight characters or 'N/A' when not a VM guest.

Linux system IP address

IP address in presentation format

Linux version

Contains a character string

Linux release

Contains a character string

Machine name

Contains a character string

OS name

Contains a character string.

The following REXX function restores embedded blanks:

```
NORMAL_FIELD = TRANSLATE(INCOMING_FIELD, 'FEX')
```

Example: List with filters

This command finds all VM systems whose names begin with 'ZV'. Exclude all Linux records. Results are returned in stem variables LXLST.nn with LXLST.0=number of records returned.

```
ADDRESS LXCON "LIST VMNODE(ZV*) LXNAME(NA)"
```

WTOR Records That OPSTATUS Returns

If the OPSTATUS function specifies a *func* argument of R and a *subfunc* argument of I, L, S, or X the function generates WTOR records with the following format:

Word Number: 1

Length: 2 or 4

Contains the outstanding reply number

Word Number: 2

Length: 126

Contains WTOR message text (starts with the three-digit reply ID, followed by a space)

If the OPSTATUS function specifies a function argument of R and a subfunction argument of S or X, the function generates a second record for each WTOR containing sysplex wide WTOR records with the following format:

Word Number: 1

Length: 8

Originating address space name.

Word Number: 2

Length: 8

JES job ID associated with the issuing address space.

This list illustrates the format of data that is returned for each type of job:

- For a started task-STC12345
- For a TSO user-TSU12345
- For a batch job-JOB12345

Note: The JES job ID of an idle initiator is always in the STCnnnnn format.

Word Number: 3

Length: 4

Address Space Identifier (ASID) (4 hexadecimal digits with leading zeroes) on the originating system.

Word Number: 4

Length: 8

System name on which the WTOR was issued.

Word Number: 5

Length: 8

Security user ID of the WTOR issuer.

Word Number: 6

Length: 10

Elapsed time in seconds that the WTOR has been outstanding.

XML Encapsulation of Types 1 and 2 Responses

All data returned from an IMS Type 2 command is encapsulated within XML framing, much of which is not pertinent to viewing the desired results. By default, IMS Type2 command support will present only what is considered significant, thus reducing the number of created variables.

To make the complete collection of XML frames into variables, specify XML(YES) on the ADDRESS OPER command.

Examples 1 and 2 below illustrate the difference. In both examples, the stem name prefix defaults to IMSTYPE2.

For more information on XML usage in IMS Type 2 output, see the IBM manual SC18-9967, *IMS V10 System Programming API Ref* section "Descriptions of XML Tags Returned as CSL OM Response."

ADDRESS MIM Command

The ADDRESS MIM host command environment interfaces with the MIM API to retrieve environment, qname, and tape information collected and maintained by CA MIM. These three unique functions of CA MIM information have their own individual formats. The MIM API interface call returns the collected data to the caller as a series of REXX variables.

The MIM API returns the following three types of data:

ENV

Contains environment information about the MIM driver

QNAME

Contains statistics and status of system ENQ/DEQ and RESERVE/RELEASE activity

TAPE

Contains status information relating to the system tape activity

Common REXX Variables

The MIM API interface call returns the data collected by the ADDRESS MIM functions to the caller as a series of REXX variables. Variables unique to each function are described in their appropriate sections.

The three ADDRESS MIM functions return the following common REXX variables:

MIM.RETURN

Contains the return code. You can examine this variable to determine the results of command execution. In some cases, the reason code may contain additional information. The value is identical to the REXX return code RC.

MIM.REASON

Contains additional information as to the reason for the return code. The return code will indicate when the reason code may be examined.

Address MIM Formats

Each command request type has a set of common parameters and a set of unique parameters that are tailored to the function they perform.

The ADDRESS MIM command has the following format:

```
ADDRESS MIM "function 'common keywords' 'specific keywords'"
```

Use the following format to request system tape information:

```
ADDRESS MIM "TAPE 'common keywords' 'specific keywords'"
```

Use the following format to request environmental information:

```
ADDRESS MIM "ENV 'common keywords' 'specific keywords'"
```

Use the following format to request ENQ/DEQ information:

```
ADDRESS MIM "QNAME 'common keywords' 'specific keywords'"
```

Common Keywords

The following keyword variables are common to all ADDRESS MIM functions. Variables unique to each function are described in their appropriate sections.

SYSTEM(*name*)

Specifies the name of the CA MIM subsystem to which the request is directed. Enter the name as a 4-character string. The request can be sent to a specific MIM facility or the special name MIMD if a specific name is not known.

Default: MIMD

STEM(*string*)

Specifies the first qualifier of the stem variable name, which is generated for all REXX variables produced by the ADDRESS MIM command. The character string defining the stem can be up to 48 characters long. The total length for the variable name cannot exceed 64 characters.

Default: The function identifiers: TAPE, ENV, or QNAME

DEBUG(YES|NO|*nnn*)

Displays diagnostic information. Use this diagnostic tool with care. DEBUG can generate a large volume of information to OPSLOG if the DISPLAY option is also selected. To display only the startup parameter values to OPSLOG, specify DEBUG without using the DISPLAY option.

YES

Activates tracing. The DISPLAY option controls the information being traced.

nnn

Activates tracing and overrides the display length of 320 for MIM API control blocks. The DISPLAY option controls the information being traced.

NO

Turns tracing off.

Default: NO

DISPLAY(CB|VAR|ALL)

Specifies the type of data to display. The DEBUG option must be active. All data displays in hex and character and all display output is directed to OPSLOG.

CB

Displays CAMIM API structures for a length of 320 or the value of DEBUG(*nnn*).

VAR

Displays all REXX variables created as a result of command execution. The entire variable is displayed.

ALL

Displays both CB and VAR data.

If DISPLAY is omitted, no trace information will be displayed.

Common Return and Reason Codes

The following return codes are common to most functions. In addition to the actual return code in R15, REXX variables MIM.RETURN and MIM.REASON can also be examined.

0

Indicates the command executed successfully and terminated properly.

1

Indicates the command string is too long.

2

Indicates an error occurred during a parsing scan.

3

Received a syntax error from an executing TAPE function. The MIM.REASON normally contains a zero, but may contain one of the following specific codes to indicate the cause of certain errors when received from an executing TAPE function:

- 1 - An invalid range has been specified.
- 2 - A count was specified with non-numeric characters.
- 3 - An invalid hardware address was used under LOCAL search. However, partial results may still be returned.

9

Indicates an error occurred during the creation of a REXX variable.

10

Indicates there is no MIM facility available to service the request.

11

QNAME

Indicates the specified QNAME does not exist. This return code applies only in a case where the NAMELIST option is used, and there is not a single match.

TAPE

Indicates the specified devices do not all matched selection criteria. The reason code contains one of the following specific codes:

1 - Indicates no TAPE devices were found.

2 - Indicates a device list is specified, but only some devices were found.

12

The storage requirement for a response buffer exceeds the allowable limits. If you are running the command as a rule, the storage limitation is 32K. Additional filtration might be necessary to reduce the amount of returning information. If the command is not running as a rule, the limit is the amount of available storage.

13

Indicates the request to free storage has failed.

14

Indicates an abend occurred. MIM.REASON contains the abend code.

16

Indicates the OPS subsystem is not active.

17

Indicates an internal control block error was detected. MIM.REASON contains one of the following specific codes:

1 - Indicates the control block belongs to OPS/MVS.

2 - Indicates the control block belongs to the MIM API.

10nn

Specifies the return codes over 1000 are indicative of an error recognized by the MIM API. The actual MIM return code *nn* is incremented by 1000. MIM.REASON contains 1 if the error is SSOB related, or 2 if from within the MIM API.

ENV Function

The ENV function interfaces with the CA MIM DRIVER API. This function is a common class service type. All active CA MIM address spaces can process the ENV function and provide environmental information to the API caller that relates to the MIM subsystems running on the local system.

The ADDRESS MIM ENV command syntax may contain optional common keywords, but has no keywords specific to the function:

```
ADDRESS MIM "ENV 'common keywords'"
```

ENV Generated REXX Variables

All REXX variables created by the ENV function have a name stem beginning with either ENV., or the override value specified in the STEM keyword. This section uses the default value ENV for the example stem.

The ENV function returns data to the calling program or calling rule that contains the following three sets of REXX variables:

- The first group displays a list of CA MIM address spaces and has the following format:

ENV.*n*

- The second group returns information related to systems that ENV.*n* connects to. These variables have the following format:

ENV.*n*.SYS*m*

- The last group contains information about facilities that ENV.*n* has defined. These variables have the following format:

ENV.*n*.FAC*m*

Examples: ENV REXX Variables■ ENV.*n*

```
ENV.0=1
ENV.1=01 *MIPROC &r11.6 SP0 11.50 MIM 0038 0000 YYNNNNN 3 IC CC
C1CC1B09BADB87EB NORMAL Active C1CA19611F35DFE4 UP
```

■ ENV.*n*.FAC.*n*

```
ENV.1.FAC.0=3
ENV.1.FAC.1=MIM Multi-image_Manager/MIM CA_MIM_Resource_Sharing
ENV.1.FAC.2=GDIF Global_Data_Integrity_Facility/GDIF CA_MI_Data_Sharing
ENV.1.FAC.3=ECMF ENQ_Conflict_Management_Facility/ECMF CA_MI_Data_Sharing
```

■ ENV.*n*.SYS.*n*

```
ENV.1.SYS.0=3
ENV.1.SYS.1=01 NONICMF XAD1 MIPROC r11.6 SP0 01 001 zOS Active **XAD1
PLEXC1 *XAD1 MF01 z/OS 01.08.00 XAD1
ENV.1.SYS.2=01 NONICMF XA55 *** 55 002 zOS Freed **XA55 *****XA55
ENV.1.SYS.3=01 NONICMF CA11 MIPROC r11.6 SP0 11 005 zOS Active **CA11
PLEXC1 *CA11 MF01 z/OS 01.08.00 CA11
```

Variable ENV.*n*

The ENV function creates the ENV.*n* variable for every active CA MIM address space. The ENV.0 variable contains the count of these CA MIM subsystems. For example, values about CA MIM subsystems are in ENV.1 through ENV.*x* where *x* is equal to ENV.0. The index numbers do not have leading zeros.

Every ENV.*n* stem has the ENV.*n*.SYS.0 variable, which contains the number (*m*) of systems connected to MIM.*n*. These systems are in ENV.*n*.SYS.1 through *m*. In the same manner every ENV.*n* stem has also ENV.*n*.FAC.0 variable with number *m* of facilities. These are in ENV.*n*.SYS.1 through *m*.

In all cases, leading zeroes and multiple blanks are suppressed in the data fields within each variable. All components within the created variables comply with REXX standards, and as such are variable length, blank separated, and easily parsed into individual REXX names. If no data is available for a particular field, it will contain a single asterisk. Also, any blanks within a field will be replaced by an underscore.

The following lists the sequence of fields for ENV.*n* variable:

1

Contains the two-character CA MIM block version identifier.

2

Contains the CA MIM complex name, which can be up to eight characters.

3

Contains the jobname, which can be up to eight characters.

4

Contains the SSI command prefix, which can be up to eight characters.

5

Contains the CA MIM product version, which can be up to eight characters.

6

Contains the CA MIM product maintenance level, which can be up to eight characters.

7

Contains the CA MIM compatibility level, which can be up to eight characters.

8

Contains the CA MIM subsystem name, which can be up to four characters.

9

Contains the Address space ID (ASID), specified in four hexadecimal characters.

10

Contains the number of active facilities specified as a four-character numeric value.

11

Contains the Facility Mask specified as an eight-bit mask translated to a single eight-character word. Characters are Y or N corresponding to on or off. The facilities correspond to the character positions as follows:

- 1 - MIM Driver
- 2 - GDIF
- 3 - ECMF
- 4 - EDIF
- 5 - GTAF
- 6 - TPCF
- 7 - GCMF
- 8 - ICMF

12

Contains the control file medium specified as a one-character numeric value, the possible values and their meanings are:

- 1 - None
- 2 - DASDONLY
- 3 - CTCDASD
- 4 - CTCONLY
- 5 - XCF

13

Contains the initial control file technique, which has the following possible values:

- ID - Initial DASD
- IC - Initial CTC
- * - Not CTCDASD

14

Contains the current control file technique, which has the following possible values:

- CD - ICurrent DASD
- CC - ICurrent CTC
- * - INot CTCDASD

15

Contains the heart-beat time. The time-of-day clock value consists of 16 hexadecimal characters.

16

Contains the heart-beat status, which consists of the following possible values:

- NORMAL
- WARNING
- PROBLEM

17

Contains the heart-beat reason, which can have up to 32 characters. The heart-beat reason is documented in the CA MIM Programming Guide section CA MIM Health Check State Management.

18

Contains the CA OPS/MVS operational time. The time-of-day clock value consists of 16 hexadecimal characters.

19

Contains the CA OPS/MVS operational state, which can have the following possible values:

- STARTING
- UP
- STOPPING
- DOWN

Variable ENV.n.SYS.m

The MIMSYS command displays the status and configuration of the CA MIM address spaces running in the complex. The following lists the sequence of fields for ENV.n.SYS.m variable:

1

Contains the two-character MIM block version identifier.

2

Indicates whether the system is connected using ICMF or a control file. The possible values are:

- ICMF - ICMF connected system
- NONICMF - Control File System

3

Contains the CA MIM system name, which is up to eight characters.

4

Contains the CA MIM jobname, which is up to eight characters.

5

Contains the CA MIM product release, which is up to eight characters.

6

Contains the CA MIM product maintenance level, which is up to eight characters.

7

Contains the CA MIM alias name, which is up to two characters.

8

Contains the CA MIM system index number, which is a numeric value of up to three characters.

9

Contains the operating system type, specified as either zOS or zVM.

10

Contains the CA MIM state, which has the following possible values:

- Unknown
- Starting
- Active
- Stopped
- Sleeping
- Awakening
- Migrating
- Freed
- Quiesced
- Pending
- Synchronizing
- Inactive

11

Contains the ICMF system name, which can be up to eight characters.

12

Contains the LSERV subsystem name, which can be up to four characters.

13

Contains the system name, which can be up to eight characters.

14

Contains the sysplex name, which can be up to eight characters.

15

Contains the VM user ID, which can be up to eight characters.

16

Contains the LPAR name, which can be up to eight characters.

17

Contains the hardware configuration name, which can be up to eight characters.

18

Contains the operating system name, which can be up to eight characters.

19

Contains the operating system version, which can be up to eight characters.

20

Contains the system SMF ID, which can be up to four characters.

Variable ENV.n.FAC.m

This set of variables contains details about the active facilities in each CA MIM address space. The following lists the sequence of fields for ENV.*n*.FAC.*m* variable:

1

Contains the facility name. The possible values are:

- MIM
- GDIF
- ECMF
- EDIF
- GTAF
- TPCF
- GCMF
- ICMF

2

Contains the facility full name, which can be up to 48 characters.

3

Contains the component family name, which can be up to 64 characters.

Example: ENV

The ENV function does not have any specific keywords:

```
ADDRESS MIM "ENV"
```

QNAME Function

The QNAME function interfaces with the CA MIM DRIVER API service type that provides environmental information to the API caller on the status of the ENQ/DEQ and the reserve and release activity. Information is available on both managed and non-managed QNAMES, with an option to select either or both.

QNAME Function Keywords

The following keyword variables are unique to ADDRESS MIM QNAME functions:

QINFO(M|NM|ALL|NONE)

Specifies the type of QNAMES to satisfy this request for information. Valid values are:

M

Returns header and detail information on managed QNAMES only.

NM

Returns header and detail information on non-managed QNAMES only.

ALL

Returns header and detail information on QNAMES that are both managed and non-managed.

NONE

Returns header information only.

Default: ALL

NAMELIST(name,...name)

Restricts the amount of returned information to a specific name, a range of names, or a list of up to 16 names. If you use a list, enclose the names in parentheses and separate individual entries with either a comma or space. Names can be up to 8 characters. Use wildcards by appending the leading portion of a name with an asterisk (*). Unless the trailing * is present, the name is considered to be specific, and must match exactly. Lists can contain a mix of both specific and wildcard names.

Only one header information variable is created per ADDRESS MIM command. Variables containing detail QNAME information are created in the order they appear in the NAMELIST. Variables for managed entries are created first, followed by those for non-managed entries.

Default: Returns information on all QNAME entries available to CA MIM at the time of the call.

Important! There is a direct connection between the number of QNAME entries available to the CA MIM API and the storage required to satisfy this command. When running as a rule, storage demands are limited to 32K, which will probably cause a QNAME request for all available information to fail. Therefore, name filtration is highly recommended when running as a rule.

QNAME Generated REXX Variables

All REXX variables created by the QNAME function have a name stem beginning with either QNAME, or the override value specified in the STEM keyword. This section uses the default value of QNAME for the example stem.

This function always creates the following variables:

- One REXX variable with the following name:
QNAME\$
- A string of variables, one for each of the QNAME entries returned by the CA MIM API, which has the following name:
QNAME.QNM.*n*

The last element of the name QNAME.QNM.*n* is a running sequence number, which starts with 1 and is incremented by 1 for each new variable, with no leading zeroes. To bypass creation of the QNAME.QNM.*n* variable, use the QNINFO keyword. In all cases, leading zeroes and multiple blanks are suppressed in the data fields within each variable.

All components within the created variables comply with REXX standards of being variable length, blank separated, and can easily be parsed into individual REXX names.

Examples of the QNAME REXX Variables

- QNAME.SL

```
QNAME.SL=01 MIPROC r11.6 SP0 PD01 1 0 108012 21271060 0 0 0 0 10 0 7538123  
0 4 923822 0 37727431 85021
```

- QNAME.QNM.n

```
QNAME.QNM.0=1  
QNAME.QNM.1=SYSDSN 776805 769331 0 0 36 776805 769331 0 0 36 M
```

QNAME.SL Variable

This header variable may contain the following information:

- The general status and statistics on the entire CA MIM subsystem facility servicing QNAME requests
- The counts of the created individual QNAME.QNM.n variables

Only one QNAME.SL variable is created per ADDRESS MIM QNAME command, regardless of the NAMELIST option.

The following lists the sequence of individual elements in this variable:

1

Contains the two-character CA MIM block version identifier.

2

Contains the jobname, which can be up to eight characters.

3

Contains the CA MIM product version, which can be up to eight characters.

4

Contains the CA MIM product maintenance level, which can be up to eight characters.

5

Contains the CA MIM system name, which can be up eight characters.

6

Contains the count of REXX variables created for managed entries, expressed as nine numeric characters.

7

Contains the count of REXX variables created for non-managed entries, expressed as nine numeric characters.

8

Contains the activation date, expressed as six numeric characters formatted as CYYDDD (century, year, and day).

9

Contains the activation time, expressed as a seven- or eight-character string formatted as hhmmsssth (hours, minutes, seconds, and tenths of seconds). A leading 0 in the hh area is suppressed.

10

Contains the count reset date, which has the same format as #8 above.

11

Contains the count reset time, which has the same format as #9 above.

12

Contains the service reset date, which has the same format as #8 above.

13

Contains the service reset time, which has the same format as #9 above.

14

Contains the accumulated service reset time in seconds, expressed as up to 9 numeric characters. It will contain 0 if the counter is totally empty, or can be expressed as "<1" if only microseconds are accumulated.

15

Contains the accumulated service reset time since last reset, and has the same format as #14 above.

16

Contains the accumulated ENQ/RESERVE requests, which can be up to 9 numeric characters.

17

Contains the accumulated ENQ/RESERVE requests since the last reset, which can be up to 9 numeric characters.

18

Contains the accumulated ENQ/RESERVE control file cycles expressed as up to 9 numeric characters.

19

Contains the accumulated ENQ/RESERVE control file cycles since last reset, which can be up to 9 numeric characters.

20

Contains the total number of integrity failures that were prevented, which can be up to 9 numeric characters.

21

Contains the total number of local integrity failures that were prevented, which can be up to 9 numeric characters.

QNAME.QNM.n

This set of variables contains detailed conflicts, ENQ, and RESERVE information. A variable set is created for every QNAME entry that matches the selection criteria specified by the QINFO and NAMELIST command options. The trailing element of the stem name is a sequence number starting with 1 and incremented by 1 for each variable created. Internal processing checks for the managed QNAMES first, non-managed QNAMES second. So, when non-managed QNAMES are desired, their sequence numbers will be the highest. If multiple names are specified within the NAMELIST keyword, sequence number incrementing continues across names.

The sequence of individual elements in these variables is as follows:

1

Contains the name of the QNAME entry, which can be up to eight characters.

2

Contains the number of ENQ requests since last reset.

3

Contains the number of ENQ processed since last reset.

4

Contains the number of RESERVE requests since last reset.

5

Contains the number of RESERVE processed since last reset.

6

Contains the number of conflicts since last reset.

7

Contains the number of ENQ requests since last restart.

8

Contains the number of ENQ processed since last restart.

9

Contains the number of RESERVE requests since last restart.

10

Contains the number of RESERVE processed since last restart.

11

Contains the number of conflicts since last restart.

12

QNAME type, valid values are:

- * = non-managed
- N = GDIF=NO
- M = SYSTEM
- S = SYSTEMS
- R = RESERVES
- A = ALL

Note: Items 2-11 above are all expressed as numeric strings up to 9 characters long, with leading zeroes removed, or 0 if empty.

Example 1: Request information for QNAME SYSABCDE

To request information for QNAME SYSABCDE only, regardless if it is managed or not, the CA MIM subsystem name is not known so the default MIMD is used, and the desired stem is SYST, enter the following:

```
ADDRESS MIM "QNAME 'NAMELIST(SYSABCDE)' 'STEM(SYST)'"
```

If the name SYSABCDE does not exist, return code 11 results. If it does exist, REXX variables SYST.SL and SYST.QNM.1 are created.

Example 2: Request QNAME information for all non-managed entries

To request QNAME information for all non-managed entries, and the default system and stem names are acceptable, enter the following:

```
ADDRESS MIM QNAME QNFO(NM)
```

If no non-managed QNAME entry is found, the return code is 10. Otherwise, it is 0, the REXX variable QNAME.SL returns with one each for every QNAME entry found. The names start with QNAME.QNM.1, then progress to QNAME.QNM.2, and so on.

Example 3: Request QNAME information for name ENQTEST

To request QNAME information for name ENQTEST when tracing is activated to display the REXX variables only, and the QNAME ENQTEST is serviced only by the CA MIM subsystem MIMT, enter the following:

```
ADDRESS MIM "QNAME NAMELIST(ENQTEST) 'DEBUG(YES)' 'DISPLAY(VAR)'  
'SYSTEM(MIMT)'"
```

Return code 10 may result if the CA MIM system MIMT is not active.

Example 4: Request SYS* managed entries

To request information on all managed entries with names starting with SYS*, and the specific entries SVTS and KEESPIN, enter the following:

```
ADDRESS MIM QNAME NAMELIST(SYS*,SVTS,KEESPIN)
```

If at least one entry matches the selection criteria, the return code is 0, along with the appropriate REXX variables. If no QNAME entry is found, the return code is 11.

Example 5: Request header information

To request only the header information in the REXX variable STEM.SL, with the default stem name QNAME, enter the following:

```
ADDRESS MIM QINFO(NONE)
```

TAPE Function

The TAPE function interfaces with the CA MIM TPDVD API service type that provides environmental information to the API caller relating to the status of system tapes. Potentially, the amount of TAPE data returned can be very large. However, this data can be filtered by using the keywords listed below which are unique to ADDRESS MIM TAPE.

TAPE Function—Keyword Variables

This section discusses the keyword variables unique to the ADDRESS MIM TAPE functions.

- Specify GLOBAL or LOCAL to limit the range of devices returned

GLOBAL|LOCAL(*name*[FOR *number*][TO *name*][*name*[,*name*...]])

LOCAL

Searches for devices based on their hardware address.

GLOBAL

Searches for devices based on the symbolic CAMIM names and then returns information for all devices on all systems in the CA MIA complex that match the specified criteria.

There are 3 methods to specify devices within the GLOBAL|LOCAL keyword:

- **List** - A list of up to 50 device names, delimited by a comma, can be stipulated. ADDRESS MIM TAPE will limit its enquiry to the devices in this list. For example:

ADDRESS MIM "TAPE 'GLOBAL(R180,R182,R18A,R18F)"

- **Range** - Using TO as the second parameter between two device names will cause ADDRESS MIM TAPE to examine every device in-between, inclusive of the device names specified. An example of its use could be:

ADDRESS MIM "TAPE 'GLOBAL(R180 TO R18F)"

Please note that the first device specified, in this case R180, must be the lower value.

- **Count** - Using FOR as the second parameter will cause ADDRESS MIM TAPE to examine every device from the 1st parameter for the number specified in the 3rd parameter. An example of its use could be:

ADDRESS MIM "TAPE 'GLOBAL(R180 FOR 16)"

- Specify TAPE to choose the type of device on which to collect and return the Tape information

TYPE(3420|3480|3490|3495|3590)

You can choose only one of these five device types at any one time. If the TYPE keyword is omitted, all device types within the criteria are examined.

Note: Device Type is only known for devices managed on the local system. Specifying this keyword automatically excludes any device not managed on the local system. Therefore, if the GLOBAL keyword is used in conjunction with the TYPE keyword, all results will be for devices that match the criteria on the local system only.

- Specify STATUS to restrict the returned data to only those devices that match the parameters on this keyword

STATUS(ONLINE|OFFLINE|ALLOC|MPEND|OPEND)

Only one of these five possible parameters can be used at a time.

ALLOC

Refers to devices that have been allocated to a job

MPEND

Refers to devices that are mount-pending

OPEND

Refers to devices that are offline-pending.

If the STATUS keyword is omitted, data will be returned for devices under any status, provided they match the rest of the specified criteria.

TAPE Generated REXX Variables

All REXX variables created by the TAPE function have a name stem beginning with either TAPE., or an override value supplied in the STEM keyword. This section uses the default value TAPE for the example stem.

Examples: TAPE REXX Variables

- TAPE.*n*

TAPE0=1

TAPE1=01 NNNNNNNNNNNNNNNNNNNNNNNNN E7D **3480 *

YNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN SE7D 0*****

- TAPE.*n*.SYS.*n*

TAPE1.SYS.0=3

TAPE1.SYS.1=01 NNNNNNNNNNNNNNNNNNNNNNNNN PD01

NNNNYNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

TAPE1.SYS.2=01 NYNNNNNNNNNNNNNNNNNNNNNNNN PD02

NNNNYNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

TAPE1.SYS.3=01 NNNNNNNNNNNNNNNNNNNNNNNNN PD03

NNNNYNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

Variable TAPE.n

Variables of the form TAPE.*n* are created, one for each device that matches the criteria specified by the user. The very first variable, TAPE.0, contains the total number of these devices that are returned. Each of these variables contains a single string of 16 blank-delimited fields. If no data is available for a particular field, a single asterisk displays. Blanks within a field are replaced by an underscore.

The following lists the sequence of fields for TAPE.*n*:

1

Contains the two-character CA MIM block version identifier.

2

Contains flag bytes 0, 1 and 2, 24 Boolean characters (eight for each flag), and either Y or N table column identifier. The conditions represented by each flag position are as follows:

- 1 - Device not locally managed.
- 2 - Caller asked for OS system name but the OS system name is not yet available. System names for this block will be filled in with the value of the 3rd position on the DEFSYS statement.
- 3-24 - Reserved.

3

Contains the internal CA MIM device number, this is equivalent to the device hardware address, specified in four hexadecimal characters.

4

Contains the four-character device UCB address.

5

Contains the four-character preference value. The possible values are from 0001 to 0255.

6

Contains one of the following device types: 3420, 3480, 3490, 3495 or 3590.

7

Contains the autopath host name, which can be MIM or any name of up to four characters.

8

Contains flag bytes 3, 4, 5, and 6, 32 Boolean characters (eight for each flag), and either Y or N table column identifier. The following are the local device attributes represented by each flag:

- 1 - Device has ACL installed
- 2 - Device has active ACL
- 3 - Device has IDRC
- 4-32 - Reserved

9

Contains the four character global device name.

10

Contains the elapsed mount-pending time in seconds, which can be up to 10 numeric characters.

11

Contains the elapsed mount pending time in the format *hh:mm:ss*.

12

Contains the jobname for which the device is reserved. The name can be up to eight characters.

13

Contains the system name on which the device is reserved for the job. The name can contain up to 8 characters.

14

Contains the job name that has this device allocated. The name can contain up to eight characters.

15

Contains the user data assigned to the device by the CA MIA VARY command.

16

Contains the VOLSER when the device is mounted or mount-pending. The VOLSER is six alpha-numeric characters.

Variable TAPE.n.SYS.m

For each TAPE.*n* REXX variable, a varying number of associated variables return data on the status of the device on each system in the CA MIA complex.

These variables take the form of TAPE.*n*.SYS.*m*, where *m* is any decimal number in a sequence of decimal numbers. Each variable contains a single string of four blank-delimited fields. If no data is available for a particular field, it will contain a single asterisk. Blanks within a field are replaced by underscores.

TAPE.*n*.SYS.0 contains the total number of systems in the CA MIA complex. The sequence for TAPE.*n*.SYS.*m* follows:

1

Contains the two-character CA MIM block version identifier.

2

Contains flag bytes 0, 1 and 2, 24 Boolean characters (eight for each flag), and either Y or N table column identifier. The conditions represented by each flag position are as follows:

- 1 - Device not managed on this system.
- 2 - This system is the local system to the API request. That is, this system is the system that originated the API request.
- 3 - This system has been freed from the CA MIA complex.
- 4 - Caller asked for OS system name but the OS system name is not yet available. System names for this block will be filled in with the value of the third position on the DEFSYS statement.
- 5-24 - Reserved.

3

Contains the system name represented by this variable, which can be up to eight characters.

4

Contains flag bytes 3, 4, 5 and 6, 32 Boolean characters (eight for each flag), and either Y or N table column identifier. The device status values (for this system) represented by each flag position are as follows:

- 1 - Device is DEDICATED to this system (CA MIA VARY DEDICATED)
- 2 - Device is DEDICATED to an external system (CA MIA VARY DEDICATED)
- 3 - Device is NOTAVAILABLE (CA MIA VARY NOTAVAILABLE)
- 4 - Device is OVERGENNED (CA MIA VARY OVERGENNED)
- 5 - Device is ALLOCATED
- 6 - Device is ONLINE

- 7 - Device is MOUNT PENDING on this system
- 8 - Device is JOB RESERVED (CA MIA VARY JOB)
- 9 - Device is PENDING OFFLINE
- 10 - Reserved
- 11 - Device is reserved for CP (VM only - CA MIA VARY CPON)
- 12 - Device is attached to a guest operating system (VM only - CA MIA ATTACH with SYSTEM option)
- 13 - Device is JOB RESERVED with the FORCE option (CA MIA VARY JOB with FORCE option)
- 14-32 - Reserved

Example 1: Request allocated tape information

To request information for all allocated tapes with device names between 3100 and 3900 within the CA MIM complex and the desired stem is MIA, enter:

```
ADDRESS MIM "TAPE 'GLOBAL(3100 TO 3900)' 'STATUS(ALLOC)' 'STEM(MIA)'"
```

GLOBAL keyword

Ensures the allocation of a matching device on any system in the CA MIM complex.

TO parameter

Specifies the examination of all devices between and inclusive of 3100 and 3900.

ALLOC parameter in STATUS

Ensures the return of only those devices that are currently allocated.

Result: REXX variables are generated in the form of *MIA.n* and *MIA.n.SYS.m*, when allocated devices within this range are found. If no device exists that matches the criteria, a return code of 11 is set.

Example 2: Request device information for hardware addresses

To request information for the devices with online hardware addresses of 3200, E70 and 3800 and use the default stem of TAPE, issue:

```
ADDRESS MIM "TAPE 'LOCAL(3200,E70,3800)' 'STATUS(ONLINE)'"
```

The LOCAL keyword indicates that we want to search for devices using their hardware address. The ONLINE parameter of STATUS further limits the returned devices to only those that are presently online.

ADDRESS MQ Commands

The ADDRESS MQ host command environment interfaces with IBM WebSphere MQ to let you do the following tasks:

- Send and retrieve MQ messages
- Set attributes for the queues or queue managers
- Inquire which attributes are set for the queues or queue managers.

ADDRESS MQ requires that the INITMSF parameter must be set to YES. ADDRESS MQ is not available in AOF Rules.

Note: Any address space that issues ADDRESS MQ commands must have The MQSERIES load library in the steplib concatenation, unless it is in linklist.

ADDRESS MQ Command Format

The ADDRESS MQ command provides operations to do the following:

- Connect to a queue manager.
- Open a queue.
- Put messages in the queue and get messages from the queue as many times as needed.
- Set queue manager attributes.
- Inquire which attributes are set for the queue or queue manager.
- Close the queue.
- Disconnect from the queue manager.

This command has the following format:

```
ADDRESS MQ "operation parameters"  
{CONNECT} {OPEN} {GET} {PUT} {SET} {INQUIRE} {CLOSE} {DISCONNECT}
```

MQ CONNECT—Provide Queue Manager Connection

The ADDRESS MQ Connect command lets you specify a specific Queue Manager connection.

This command has the following format:

```
ADDRESS MQ CONNECT {QMGR(name)} {CONHANDLE(varHCONN)}
```

QMGR

Specifies the queue manager name to which you want to connect. The name can be up to 48 bytes and is case sensitive.

CONHANDLE(*varHCONN*)

Specifies the variable name to store the connection handle.

Example: MQ Connect

This example connects to the queue manager CSQ7:

```
ADDRESS MQ "CONNECT QMGR(CSQ7) CONHANDLE(hconn)"
```

MQ OPEN—Open a Queue

The ADDRESS MQ OPEN command lets you open a specific message queue.

This command has the following format:

```
ADDRESS MQ OPEN {QUEUE(name)|QMGR(name)}  
{OPTIONS(DEFAULT|SHARED|EXCLUSIVE|BROWSE|OUTPUT|SET|INQUIRE)}  
{CONHANDLE(varHCONN)}  
{OBJHANDLE(varHOBJ)}
```

QUEUE

Defines the message queue name. The name is case sensitive and can be up to 48 bytes.

Mutually exclusive with QMGR.

QMGR

Defines the queue manager name. The name is case sensitive and can be up to 48 bytes.

Mutually exclusive with QUEUE.

OPTIONS

Specifies your access or other options.

- For QUEUE, you can specify BROWSE, OUTPUT, SET, or INQUIRE. In addition, you may omit or specify one of the following mutually exclusive parameters: DEFAULT, SHARED, EXCLUSIVE.
- For QMGR, you can specify only INQUIRE mode.

DEFAULT

Indicates the default queue setting. The message is deleted from the queue.

SHARED

Indicates shared access. The message is deleted from the queue.

EXCLUSIVE

Indicates exclusive access. The message is deleted from the queue.

BROWSE

Indicates using browse mode to access the message queue. The message is kept on the queue.

OUTPUT

Indicates output mode. This option is necessary for putting messages on the queue.

SET

Indicates set mode. This option is necessary for setting the attributes of the queue.

INQUIRE

Indicates inquire mode. This option is necessary for inquiring which attributes of the queue or queue manager are set.

CONHANDLE(varHCONN)

Defines the variable name that stores the Connection Handle.

OBJHANDLE(varHOBJ)

Defines the variable name that stores the Object Handle.

Example: MQ OPEN

This example shows how to open a queue (QM2) for output.

```
ADDRESS MQ "OPEN QUEUE(QM2) OPTIONS(OUTPUT)",  
          "CONHANDLE(hconn) OBJHANDLE(hobj)"
```

MQ GET—Retrieve Messages

The ADDRESS MQ GET command retrieves messages from the message queue using the FIFO retrieval method.

This command has the following format:

```
ADDRESS MQ GET OPTIONS([TRUNCATE] [KEEP] [CORRELID(corr)] [WAIT(time)] [CONVERT(ccsid)]  
{CONHANDLE(varHCONN)}  
{OBJHANDLE(varHOBJ)}  
{DATA(varDATA)}
```

OPTIONS

Specifies the processing options.

TRUNCATE

(Optional) Allows data to be truncated. If omitted, the GET operation fails when the data contains over 32,000 bytes.

KEEP

(Optional) Causes messages to be kept on the queue. Keep is required when the queue is being browsed.

CORRELID

(Optional) Defines the correlation ID of the message to be returned.

Note: The CORRELID must be specified in hex format.

WAIT

(Optional) Specifies the amount of time in seconds to wait for the message. If omitted, GET does not wait for any messages.

Possible value: **1 to 900 seconds**

CONVERT

(Optional) Converts data to the host code page that is specified by the *ccsid*.

CONHANDLE(*varHCONN*)

Defines the variable name that stores the connection handle.

OBJHANDLE(*varHOBJ*)

Defines the variable name that stores the object handle.

DATA(varDATA)

Defines the variable name that stores the message.

Example: MQ GET

This example gets a message with a specific CorrelID, stored in variable mycorr:

```
ADDRESS MQ "GET OPTIONS(CORRELID('mycorr'))",  
"CONHANDLE(hconn) OBJHANDLE(hobj) DATA(data2get)"
```

MQ PUT—Put Messages on the Queue

The ADDRESS MQ PUT command puts the message on a message queue.

This command has the following format:

```
ADDRESS MQ PUT OPTIONS([STEM(varSTEM)] [CORRELID(NEW)] [EXPIRY(time units | UNLIMITED)]  
[FORMAT(NONE | STRING)] [PRIORITY(msgpriority)])  
{CONHANDLE(varHCONN)}  
{OBJHANDLE(varHOBJ)}  
{DATA(varDATA)}
```

OPTIONS

Specifies the processing options.

STEM(varSTEM)

(Optional) Creates the variable varSTEM.CORRELID that contains a value of CorrelID. The output displays in hex format.

CORRELID(NEW)

(Optional) Puts the message and receives new unique CORRELID, which is stored in varSTEM.CORRELID.

Specifying STEM automatically creates the variable varSTEM.CORRELID. If STEM is omitted, the default name MQ.CORRELID is used.

Note: The new CORRELID is prefixed with 'OPS/MVS'.

EXPIRY(time units | UNLIMITED)

(Optional) Specifies the expiration time of the message in minutes, hours, days, or weeks.

Possible values:

- Time: 1 to 120
- Units: Minutes, hours, days, or weeks
- UNLIMITED: An unlimited expiration time

Default: Unlimited**FORMAT(NONE | STRING)**

(Optional) Specifies a format name of message data.

Possible values:

- NONE: Format of message data is undefined. Data cannot be converted using the CONVERT option.
- STRING: Message of this format can be converted when the CONVERT option is specified.

Default: NONE**PRIORITY(msgpriority)**

(Optional) Puts the message and set the message priority.

Possible values:

- msgpriority: 0 - 9.

Default: A queue default message priority.**CONHANDLE(varHCONN)**

Defines the variable name that stores the connection handle.

OBJHANDLE(varHOBJ)

Defines the variable name that stores the object handle.

DATA(varDATA)

Defines the variable name that stores the message.

Note: Input data automatically truncates to 32,000 bytes if data > 32000 bytes.**Examples: MQ PUT**

This example puts the message on the queue, specifies an expiry time, and receives the Correl ID to MYSTEM.CORRELID:

```
ADDRESS MQ "PUT OPTIONS(EXPIRY(80 DAYS),STEM(myStem),CORRELID(NEW))",  
          "CONHANDLE(hconn) OBJHANDLE(hobj) DATA(data2put)"
```

This example does not specify the STEM option, which receives the Correl ID to MQ.CORRELID:

```
ADDRESS MQ "PUT OPTIONS(EXPIRY(80 DAYS),CORRELID(NEW))",
```

```
"CONHANDLE(hconn) OBJHANDLE(hobj) DATA(data2put)"
```

MQ SET—Set Attributes

The ADDRESS MQ SET command sets the queue attributes.

Before calling this command, you must store the attributes in compound variables. This lets you specify a stem variable or use the default stem, and the tail is the attribute name.

This command has the following format:

```
ADDRESS MQ SET
  {ATTRIBUTES(attributes)}
  [STEM(varSTEM)]
  {CONHANDLE(varHCONN)}
  {OBJHANDLE(varHOBJ)}
```

ATTRIBUTES(*attributes*)

Specifies the attributes to set. You can specify all valid attributes.

There are two types of attributes:

Character attributes

The character length is defined with IBM definitions. If the length is greater, then the attribute will be truncated.

TRIGGER_DATA

Sets the trigger data (length - MQ_TRIGGER_DATA_LENGTH).

Integer attributes

INHIBIT_GET

Controls whether to allow GET operations.

Valid values: 0 - 1

0

enable

1

disable

INHIBIT_PUT

Controls whether to allow PUT operations.

Valid values: 0 - 1

0

enable

1

disable

TRIGGER_CONTROL

Sets the trigger control.

Valid values: 0 - 1

0

enable

1

disable

TRIGGER_DEPTH

Sets the trigger depth.

Valid values: 1 - 999999999

TRIGGER_MSG_PRIORITY

Sets the threshold message priority for triggers.

Valid values: 0 - MaxPriority

MaxPriority

Maximum message priority supported by the queue manager.

Lowest: 0

Highest: MaxPriority

TRIGGER_TYPE

Sets the trigger type.

Valid values: 0 – 3

0

none

1

first

2

every

3

depth

Note: To see full description of attributes, see *IBM WebSphere MQ Application Programming Reference*, where character attributes are prefixed with MQCA_ and integer attributes with MQIA_.

STEM(varSTEM)

(Optional) Defines the stem of the compound variable that stores the attributes. If omitted, MQ will be used.

Attributes are read from this variable and set by the MQ SET command.

Default: MQ

CONHANDLE(*varHCONN*)

Defines the variable name that stores the connection handle.

OBJHANDLE(*varHOBJ*)

Defines the variable name that stores the object handle.

Examples: Use MQ SET to Set Attributes

This example inhibits messages from being put on the queue and omits the STEM keyword.

```
MQ.INHIBIT_PUT = 1
address MQ "SET CONHANDLE(HCONN) OBJHANDLE(HOBJ);
          "ATTRIBUTES(INHIBIT_PUT)"
```

This example allows messages to be put on the queue and sets the trigger data field.

```
SET.INHIBIT_PUT = 0
SET.TRIGGER_DATA = "MQTEST"
address MQ "SET CONHANDLE(HCONN) OBJHANDLE(HOBJ);
          "ATTRIBUTES(INHIBIT_PUT, TRIGGER_DATA) STEM(SET)"
```

MQ INQUIRE—Inquire Attributes

The ADDRESS MQ INQUIRE command inquires which attributes are set for the queue or queue manager.

Calling this command creates the compound variables, where you can specify a stem variable or use the default stem, and the tail is the attribute name.

This command has the following format:

```
ADDRESS MQ INQUIRE
  {ATTRIBUTES(attributes)}
  [STEM(varSTEM)]
  {CONHANDLE(varHCONN)}
  {OBJHANDLE(varHOBJ)}
```

ATTRIBUTES(*attributes*)

Specifies which attributes to get. You can specify up to 16 attributes.

There are six types of attributes:

- Character attributes for the queue
- Character attributes for the queue manager
- Character attributes common for queue and queue manager
- Integer attributes for the queue
- Integer attributes for the queue manager
- Integer attributes common for queue and queue manager

Character attributes for the queue:

- BACKOUT_REQ_Q_NAME
Returns the excessive backout requeue name.
- BASE_Q_NAME
Returns the queue name to which the alias resolves.
- CF_STRUC_NAME
Returns the coupling-facility structure name.
- CLUSTER_NAME
Returns the cluster name.
- CLUSTER_NAMELIST
Returns the cluster name list.
- CREATION_DATE
Returns the queue creation date.

- CREATION_TIME
Returns the queue creation time.
- INITIATION_Q_NAME
Returns the initiation queue name.
- PROCESS_NAME
Returns the process definition name.
- Q_DESC
Returns the queue description.
- Q_NAME
Returns the queue name.
- REMOTE_Q_MGR_NAME
Returns the remote queue manager name.
- REMOTE_Q_NAME
Returns the remote queue name as known on the remote queue manager.
- STORAGE_CLASS
Returns the storage class name.
- TRIGGER_DATA
Returns the trigger data.
- XMIT_Q_NAME
Returns the transmission queue name.

Character attributes for the queue manager:

- CHANNEL_AUTO_DEF_EXIT
Returns the automatic channel definition exit name.
- CLUSTER_WORKLOAD_DATA
Returns the data passed to cluster workload exit.
- CLUSTER_WORKLOAD_EXIT
Returns the cluster workload exit name.
- COMMAND_INPUT_Q_NAME
Returns the system command input queue name.
- DEAD_LETTER_Q_NAME
Returns the dead-letter queue name.
- DEF_XMIT_Q_NAME
Returns the default transmission queue name.

- **DNS_GROUP**
Returns the TCP listener group name.
- **IGQ_USER_ID**
Returns the intra-group queuing user identifier.
- **LU_GROUP_NAME**
Returns the LU 6.2 listener generic LU name.
- **LU_NAME**
Returns the LU name to use for outbound LU 6.2 transmissions.
- **LU62_ARM_SUFFIX**
Returns the suffix of the SYS1.PARMLIB member APPCPMxx.
- **Q_MGR_DESC**
Returns the queue manager description.
- **Q_MGR_IDENTIFIER**
Returns the queue-manager identifier.
- **Q_MGR_NAME**
Returns the local queue manager name.
- **QSG_NAME**
Returns the queue-sharing group name.
- **REPOSITORY_NAME**
Returns the cluster name for which queue manager provides repository services.
- **REPOSITORY_NAMELIST**
Returns the name of the namelist object containing the names of clusters for which queue manager provides repository services.
- **TCP_NAME**
Returns the name of the TCP/IP system that you are using.

Character attributes common for queue and queue manager:

- **ALTERATION_DATE**
Returns the date of most-recent alteration.
- **ALTERATION_TIME**
Returns the time of most-recent alteration.

Integer attributes for the queue:

- **BACKOUT_THRESHOLD**

- Returns the backout threshold.
- CLWL_Q_PRIORITY
 - Returns the priority of the queue.
- CLWL_Q_RANK
 - Returns the rank of the queue.
- CLWL_USEQ
 - Returns the remote queues to be used.
- CURRENT_Q_DEPTH
 - Returns the number of messages on queue.
- DEF_BIND
 - Returns the default binding.
- DEF_INPUT_OPEN_OPTION
 - Returns the default open-for-input option.
- DEF_PERSISTENCE
 - Returns the default message persistence.
- DEF_PRIORITY
 - Returns the default message priority.
- DEFINITION_TYPE
 - Returns the queue definition type.
- HARDEN_GET_BACKOUT
 - Returns whether to harden the backout count.
- INDEX_TYPE
 - Returns the type of index maintained for queue.
- INHIBIT_GET
 - Returns whether get operations are allowed.
- INHIBIT_PUT
 - Returns whether put operations are allowed.
- MAX_MSG_LENGTH
 - Returns the maximum message length.
- MAX_Q_DEPTH
 - Returns the maximum number of messages allowed on queue.
- MSG_DELIVERY_SEQUENCE
 - Returns whether message priority is relevant.

- **NPM_CLASS**
Returns the level of reliability for nonpersistent messages.
- **OPEN_INPUT_COUNT**
Returns the number of MQOPEN calls that have the queue open for input.
- **OPEN_OUTPUT_COUNT**
Returns the number of MQOPEN calls that have the queue open for output.
- **Q_TYPE**
Returns the queue type.
- **QSG_DISP**
Returns the queue-sharing group disposition.
- **RETENTION_INTERVAL**
Returns the queue retention interval.
- **SHAREABILITY**
Returns whether the queue can be shared for input.
- **TRIGGER_CONTROL**
Returns the trigger control.
- **TRIGGER_DEPTH**
Returns the trigger depth.
- **TRIGGER_MSG_PRIORITY**
Returns the threshold message priority for triggers.

Valid values: 0 - MaxPriority

MaxPriority

Maximum message priority supported by the queue manager.

Lowest: 0

Highest: MaxPriority

- **TRIGGER_TYPE**
Returns the trigger type.
- **USAGE**
Returns the usage.

Integer attributes for the queue manager:

- **ACTIVE_CHANNELS**
Returns the maximum number of active channels.

- **ADOPTNEWMCA_CHECK**
Returns whether to adopt the new MCA when a new inbound channel is detected with the same name as the active MCA.
- **ADOPTNEWMCA_TYPE**
Returns whether to automatically restart an orphaned MCA instance of a given channel type when a new inbound channel request matches the AdoptNewMCACheck parameters.
- **BRIDGE_EVENT**
Returns the control attribute for IMS bridge events.
- **CHANNEL_EVENT**
Returns the control attribute for channel events.
- **CHINIT_ADAPTERS**
Returns the number of adapter subtasks to use for processing WebSphere MQ calls.
- **CHINIT_DISPATCHERS**
Returns the number of dispatchers to use for the channel initiator.
- **CHINIT_TRACE_AUTO_START**
Returns whether to start channel initiator trace automatically.
- **CHINIT_TRACE_TABLE_SIZE**
Returns the size of the channel initiator's trace data space (in MB).
- **CLUSTER_WORKLOAD_LENGTH**
Returns the cluster workload length.
- **CLWL_MRU_CHANNELS**
Returns the channel MRU.
- **CODED_CHAR_SET_ID**
Returns the coded character set identifier.
- **COMMAND_EVENT**
Returns the control attribute for command events.
- **COMMAND_LEVEL**
Returns the command level supported by queue manager.
- **DNS_WLM**
Returns whether the TCP listener registers with Workload Manager for Dynamic Domain Name Services.
- **EXPIRY_INTERVAL**

Returns the interval between scans for expired messages.

■ IGQ_PUT_AUTHORITY

Returns the intra-group queuing put authority.

■ INTRA_GROUP_QUEUING

Returns the intra-group queuing support.

■ LISTENER_TIMER

Returns the time interval in seconds between WebSphere MQ attempts to restart the listener (in case of an APPC or TCP/IP failure).

■ LU62_CHANNELS

Returns the maximum number of channels that can be current, or clients that can be connected.

■ MAX_CHANNELS

Returns the maximum number of channels that can be current.

■ MAX_HANDLES

Returns the maximum number of handles.

■ MAX_MSG_LENGTH

Returns the maximum message length.

■ MAX_PRIORITY

Returns the maximum priority.

■ MAX_UNCOMMITTED_MSGS

Returns the maximum number of uncommitted messages within a unit of work.

■ OUTBOUND_PORT_MAX with OUTBOUND_PORT_MIN

Defines range of port numbers to use when binding outgoing channels.

■ OUTBOUND_PORT_MIN with OUTBOUND_PORT_MAX

Defines range of port numbers to use when binding outgoing channels.

■ PLATFORM

Returns the platform on which the queue manager resides.

■ RECEIVE_TIMEOUT

Returns approximately how long a TCP/IP channel waits to receive data.

■ RECEIVE_TIMEOUT_MIN

Returns the minimum time that a TCP/IP channel waits to receive data.

■ RECEIVE_TIMEOUT_TYPE

Returns approximately how long a TCP/IP channel waits to receive data.

- **SSL_EVENT**
Returns the control attribute for channel events.
- **SSL_RESET_COUNT**
Returns the SSL key reset count.
- **SYNCPOINT**
Returns the syncpoint availability.
- **TCP_CHANNELS**
Returns the maximum number of channels that can be current, or clients that can be connected.
- **TCP_KEEP_ALIVE**
Returns whether to use the TCP KEEPALIVE facility to check that the connection is still available.
- **TCP_STACK_TYPE**
Returns whether the channel initiator can use only the TCP/IP address space specified in TCPNAME, or can optionally bind to any selected TCP/IP address.
- **TRACE_ROUTE_RECORDING**
Controls recording of trace-route information.
- **TRIGGER_INTERVAL**
Returns the trigger interval.

Integer attributes for the queue and queue manager

- **CLWL_USEQ**
Use remote queues.

STEM(varSTEM)

(Optional) Defines the stem of the compound variable that stores the attributes. If omitted, MQ will be used.

This compound variable is created after the MQ INQUIRE command.

CONHANDLE(varHCONN)

Defines the variable name that stores the connection handle.

OBJHANDLE(varHOBJ)

Defines the variable name that stores the object handle.

Examples: MQ INQUIRE

This example of MQ INQUIRE has two attributes, alteration date and time, and omits the STEM.

```
address MQ "INQUIRE CONHANDLE(HCONN) OBJHANDLE(HOBJ)",  
          "ATTRIBUTES(ALTERATION_DATE ALTERATION_TIME)"  
say MQ.ALTERATION_DATE  
say MQ.ALTERATION_TIME
```

This example of MQ INQUIRE has three attributes, TCP name, number of handles, and TCP channels. The STEM is omitted. This example is valid only if the object is queue manager (object opened with MQ OPEN command).

```
address MQ "INQUIRE CONHANDLE(HCONN) OBJHANDLE(HOBJ)",  
          "ATTRIBUTES(TCP_NAME MAX_HANDLES TCP_CHANNELS)"  
say MQ.TCP_NAME  
say MQ.MAX_HANDLES  
say MQ.TCP_CHANNELS
```

MQ CLOSE—Close Queue

The ADDRESS MQ CLOSE command closes the message queue.

This command has the following format:

```
ADDRESS MQ CLOSE  
  {CONHANDLE(varHCONN)}  
  {OBJHANDLE(varHOBJ)}
```

CONHANDLE(varHCONN)

Defines the variable name that stores the connection handle.

OBJHANDLE(varHOBJ)

Defines the variable name that stores the object handle.

Example: MQ CLOSE

This example closes the message queue:

```
ADDRESS MQ "CLOSE CONHANDLE(hconn) OBJHANDLE(hobj)"
```

MQ DISCONNECT—Disconnect Queue Manager

The ADDRESS MQ DISCONNECT command disconnects from a specific queue manager.

This command has the following format:

```
ADDRESS MQ DISCONNECT  
      {CONHANDLE(varHCONN)}
```

CONHANDLE(varHCONN)

Defines the variable name that stores the connection handle.

Example: MQ DISCONNECT

This example disconnects from the message queue:

```
ADDRESS MQ "DISCONNECT CONHANDLE(hconn)"
```

ADDRESS MQ Debug Parameter

The DEBUG parameter can be specified to create debugging information and send it to the OPSLOG. DEBUG is an optional parameter available for every ADDRESS MQ command.

The DEBUG parameter can be specified for any MQ command using the following format:

```
ADDRESS MQ "command DEBUG(keyword)"  
      {YES|NO}
```

YES

Turns debugging on, which causes MQ to send debugging information to the OPSLOG.

NO

Turns debugging off.

Default: NO

How to Use ADDRESS MQ Commands

The following process provides an example of how to use MQ commands.

We will do the following:

1. Connect to the queue manager.
2. Open the queue for output and setting.
3. Inhibit the putting of the messages.
4. Try to put the message (command will fail).
5. Allow the putting of the messages.
6. Try again to put the message with expiration time 5 minutes (command will work).
7. Close the queue.
8. Disconnect from the queue manager.

Using the following ADDRESS MQ commands:

(1) address MQ "CONNECT QMGR(CS01) CONHANDLE(CH)"

(2) address MQ "OPEN QUEUE(CS01.MYQUEUE) OPTIONS(OUTPUT, SET),
"CONHANDLE(CH) OBJHANDLE(OH)"

(3) MQINHIBIT_PUT = 1
address MQ "SET CONHANDLE(CH) OBJHANDLE(OH),
"ATTRIBUTES(INHIBIT_PUT)"

(4) MYDATA = "This message will not be on the queue."
address MQ "PUT CONHANDLE(CH) OBJHANDLE(OH) DATA(MYDATA)"

(5) MQINHIBIT_PUT = 0
address MQ "SET CONHANDLE(CH) OBJHANDLE(OH),
"ATTRIBUTES(INHIBIT_PUT)"

(6) MYDATA = "This message will be on the queue."
address MQ "PUT CONHANDLE(CH) OBJHANDLE(OH) OPTIONS(EXPIRY(5 MINUTES)),
"DATA(MYDATA)"

(7) address MQ "CLOSE CONHANDLE(CH) OBJHANDLE(OH)"

(8) address MQ "DISCONNECT CONHANDLE(CH)"

ADDRESS MQ Return Codes

Variables MQ.RETURN and MQ.REASON are created after each CONNECT, OPEN, GET, PUT, CLOSE, DISCONNECT command executes. The MQ.RETURN variable contains the same value as RC. The MQ.REASON variable contains detail information about the RETURN code.

ADDRESS MQ produces the following return and reason codes:

0

Okay

1. Output data from the GET function truncated
2. Input data for the PUT function truncated
3. Input data for the SET function truncated

2

PSF error

1. Parse error
2. Syntax error

4

Error when accessing or creating a REXX variable.

1. Invalid integer value

8

Error during the CONNECT, DISCONNECT, OPEN, or CLOSE operation.

1. ADDRESS MQ cannot be used in a rule.

Possible error in the called MQ macro (see the note).

12

Error during the GET, PUT, SET, or INQUIRE operation. Possible error in the called MQ macro (see the note).

1. Convert error

Possible error with *ccsid* or data format.

16

Unexpected ABEND.

20

ADDRESS MQ could not send or retrieve MQ messages. Check to ensure that the INITMSF parameter is set to YES.

Note: To see details about MQ macro reason codes, see the *IBM WebSphere MQ for z/OS Messages and Codes*. MQ.REASON contains the decimal form of the MQ reason code.

ADDRESS NETMAN Host Environment

The ADDRESS NETMAN host environment enables you to open, update, or close records in CA Netman. This REXX-based interface allows CA OPS/MVS rules or OPS/REXX programs to perform CA Netman functions in real time, while an event is actually occurring. The ADDRESS NETMAN host environment uses an Application Programming Interface (API) to issue Machine-Generated Problem Tracking (MGPT) commands. The API allows CA OPS/MVS to manipulate problem records in a CA Netman database, whether CA Netman is running on the same system or a remote system. This feature also forces important information such as job names, message IDs, and message text to be automatically filled in while you are using the interface.

Keep these points in mind when you issue the ADDRESS NETMAN host command:

- The maximum length of an ADDRESS NETMAN host command is 256 characters. Whenever possible, use the MGPT table overrides to automatically fill in field values, providing more room for the values that cannot be overridden.
- ADDRESS NETMAN commands that are issued from a rule, except a TOD or REQ rule, cause CA Netman response messages to be issued as WTOs to the hardcopy log (a message rule may trap these messages). OPS/REXX programs, TOD rules, and REQ rules obtain their CA Netman response messages synchronously in the external data queue.
- The ADDRESS NETMAN interface requires the NETMAN CCI, or NCCI, receiver to be active.

Note: For more information about the NCCI, see the *CA Netman Techniques Guide* and *Systems Programmer Guide*. For details about MGPT and the API, you should also see the CA Netman API documentation.

ADDRESS NETMASTR Host Environment

The ADDRESS NETMASTR host environment is used to create, update, replace, annotate, set, close, and clear alerts on the CA NetMaster Alert Monitor screen. This host environment can be used in both OPS/REXX programs and synchronous AOF rules.

Note: The ADDRESS NETMASTR interface is available in CA NetMaster.

ADDRESS NETMASTR Command—Create and Work with Alerts

The ADDRESS NETMASTR host environment is used to create, update, replace, annotate, set, close, and clear alerts on the CA NetMaster Alert Monitor screen. This host environment can be used in both OPS/REXX programs and synchronous AOF rules.

Note: The ADDRESS NETMASTR interface is available in CA NetMaster.

This command has the following format:

```
ADDRESS NETMASTR "function"  
{CALLTYPE(ANNOTATE|CLEAR|CLOSE|CREATE|REPLACE|SET|UPDATE)}  
[ANUSERID(uid)]  
[APPLID(ApplicationID)]  
[CLASSID(ClassID)]  
[DESC(Description)]  
[EXPLAIN('string1','string2',...,'string20')]  
[EXPLVAR(GrexxVarPrefix)]  
[NMSUBSYS(NetMasterSSID)]  
[RECMACTN('string1','string2',...,'string20')]  
[RECMAVAR(GrexxVarPrefix)]  
[RESCLASS(ResourceClass)]  
[RESID(ResourceID)]  
[RESOURCE(ResourceName)]  
[RESTYPE(ResourceType)]  
[SEVERITY(1|2|3|4)]  
[SEVUPDT(YES|NO)]  
[SYSACTN('string1','string2',...,'string20')]  
[SYSACVAR(GrexxVarPrefix)]  
[SYSTEMNM(SystemName)]  
[TEXT('string1','string2',...,'string20')]  
[TEXTUPDT(YES|NO)]  
[TEXTVAR(GrexxVarPrefix)]  
[TRACKID(TrackingID)]
```

function

The required *function* code is the only positional parameter. Currently, the only supported *function* code is ALERT.

ANUSERID

(Optional) Defines the user ID to be associated with the annotation text that is specified on the TEXT or TEXTVAR keywords. This keyword only has meaning for CALLTYPE(ANNOTATE) and is ignored if specified with any other CALLTYPE value. The *uid* can be any 1- to 32-character text string representing a user ID. Enclose the string in quotes if it contains any blanks. This string may, but does not necessarily have to, represent a standard TSO USERID.

APPLID

(Optional) Defines the application ID as it appears in the CA NetMaster Alert Monitor display. The APPLID must be a valid CA OPS/MVS subsystem name. It must be exactly four characters long, begin with the character string OPS, and end with an uppercase alphabetic character.

Example: Specifying an application ID

APPLID(OPSA)

When this keyword is omitted, the current CA OPS/MVS subsystem ID (typically OPSS) is used as the default value. The APPLID value is used for alert correlation in combination with the CLASSID, RESID, and in some cases the TRACKID values, when you perform operations on existing alerts.

CALLTYPE

Specifies the type of ALERT call. This keyword is required.

ANNOTATE

Adds notes to an existing alert that matches the APPLID, CLASSID, RESID, and TRACKID specified when the alert was created. ANNOTATE uses keywords ANUSERID and TEXT or TEXTVAR.

CLEAR

Clears one or more alerts that match the values specified in the APPLID, CLASSID, and RESID keywords to identify the alerts you want to clear.

CLOSE

Closes the most recent alert created with the APPLID, CLASSID, RESID, and TRACKID specified when the alert was created.

CREATE

Creates a new alert.

REPLACE

Replaces one or more active alerts that match the values specified in the APPLID, CLASSID, and RESID keywords with a new alert. All old alerts that match the APPLID, CLASSID, and RESID keywords are closed and annotated with text indicating that they were replaced by a new alert. A new alert will be created if there is no match on APPLID, CLASSID, and RESID.

SET

Sets certain attributes of an alert that was previously created by specifying its APPLID, CLASSID, RESID, and TRACKID. Only the following alert attributes may be changed with a SET command:

- DESC
- EXPLANAT
- RECMACTN
- RESOURCE
- SEVERITY
- SYSTEMNM
- SYSACTN
- TEXT

UPDATE

Updates a recurring alert that matches the APPLID, CLASSID, and RESID. A recurring alert includes fields that indicate how many times the alert has occurred, the last date/time it occurred, and the elapsed time between the first and most recent occurrences. These fields are also present, but null, in normal alerts. The procedure to update a recurring alert is exactly the same as that to create a recurring alert. This call creates a new alert, if none exist, or updates the alert, if it does exist. The alert severity can optionally be updated by specifying the SEVUPDT keyword. The alert description, text, explanation, system action and recommended action can optionally be updated by specifying the TEXTUPDT keyword.

CLASSID

(Optional) Defines the class of alert. This is used to differentiate between types of alerts. The class ID can be any 1- to 30-character text string. Enclose the string in quotes if it contains any blanks. The CLASSID value is used for alert correlation in combination with the APPLID, RESID, and in some cases the TRACKID values, when you perform operations on existing alerts. For example:

```
CLASSID('OPS/MIS')
```

DESC

(Optional) Defines the alert description. If this keyword is omitted, then a default description string of 'No description' is used. Specify a text string of up to 80 characters enclosed in single quotes. The text is displayed under the Alert Description heading in the CA NetMaster Alert Monitor display.

EXPLAIN

(Optional) Specifies up to 20 text strings, each containing up to 75 characters. Each line of text is displayed as a separate line under the Alert Explanation heading in the CA NetMaster Alert Monitor display. Below are sample system action strings:

```
EXPLAIN('This alert is being displayed because today is Blue Monday','Second explanation string','Third explanation string')
```

You can insert the contents of a REXX variable into the system action string. For example, the sample below inserts the contents of variable EXPLANAT into the explanation text as an explanation string:

```
EXPLAIN("||EXPLANAT||")
```

Note: The EXPLVAR and EXPLAIN keywords are mutually exclusive.

EXPLVAR

(Optional) Defines the stem for a set of REXX variables with each variable containing one line of explanation text. You define these variables and assign lines of text to them in separate statements in the OPS/REXX program that invokes the ADDRESS NETMASTR environment. You are limited to a maximum of 20 lines of explanation text and each line can contain up to 75 characters. If you specify more than 20 lines or the lines are longer than 75 characters, then the data beyond the defined limits is ignored. For example, if you specify EXPLVAR(LINE_DATA.), then the names of the variables storing lines of your explanation text will be LINE_DATA.1, LINE_DATA.2, and so on.

The stem name you specify must meet standard REXX conventions for stem names. If you are not familiar with these conventions, then see the book *The REXX Language: A Practical Approach to Programming* by M. F. Cowlshaw.

Instead of a stem, the EXPLVAR keyword can also specify a valid REXX variable name prefix. For example, if you specify EXPLVAR(PREFIX_DATA), then the names of variables storing your system action text are PREFIX_DATA1, PREFIX_DATA2, and so on.

The number of lines of explanation text produced depends on the number of consecutive variables that meet the criteria. Therefore, in the following example:

```
explvar.1="This alert is being displayed because today is Blue Monday."
explvar.2="Second explanation string"
explvar.3="Third explanation string"
```

A command specifying ADDRESS NETMASTR EXPLVAR("explvar.") produces only three lines of explanation text.

Note: The EXPLVAR and EXPLAIN keywords are mutually exclusive.

NMSUBSYS

(Optional) Defines the name of the CA NetMaster EPS subsystem to which the request is directed. The CA NetMaster EPS subsystem name can be any 4-character name of an active CA NetMaster subsystem. This keyword should be specified only if the typical mechanism of automatically finding an active CA NetMaster subsystem does not work or you want to select an alternate CA NetMaster subsystem.

RECMACTN

(Optional) Specifies up to 20 text strings, each containing up to 75 characters. Each line of text is displayed as a separate line under the Recommended Action heading in the CA NetMaster Alert Monitor display. Below are some sample system action strings:

```
RECMACTN('Restart the system to correct the error',  
"Second recommended action string",Third rec string)
```

You can insert the contents of a REXX variable into the system action string. For example, the sample below inserts the contents of variable RECACT into the recommended action text as a recommended action string:

```
RECMACTN("||RECACT||")
```

Note: The RECMAVAR and RECMACTN keywords are mutually exclusive.

RECMAVAR

(Optional) Defines the stem for a set of REXX variables with each variable containing one line of recommended action text. You define these variables and assign lines of text to them in separate statements in the OPS/REXX program that invokes the ADDRESS NETMASTR environment. You are limited to a maximum of 20 lines of recommended action text and each line can contain up to 75 characters. If you specify more than 20 lines or the lines are longer than 75 characters, then the data beyond the defined limits is ignored. For example, if you specify RECMAVAR(LINE_DATA.), the names of the variables storing lines of your alert text will be LINE_DATA.1, LINE_DATA.2, and so on.

The stem name you specify must meet standard REXX conventions for stem names. If you are not familiar with these conventions, then see the book *The REXX Language: A Practical Approach to Programming* by M. F. Cowlshaw.

Instead of a stem, the RECMAVAR keyword can also specify a valid REXX variable name prefix. For example, if you specify RECMAVAR(PREFIX_DATA), then the names of variables storing your recommended action text are PREFIX_DATA1, PREFIX_DATA2, and so on.

The number of lines of recommended action text produced depends on the number of consecutive variables that meet the criteria. Therefore, in the following example:

```
recmavar.1='Restart the system to correct the error'  
recmavar.2='Second recommended action string'  
recmavar.3='Third rec string'
```

A command specifying ADDRESS NETMASTR RECMAVAR("recmavar.") produces only three lines of recommended action text.

Note: The RECMAVAR and RECMACTN keywords are mutually exclusive.

RESCLASS

(Optional) Defines the logical resource class associated with the alert resource. The resource class can be any 1- to 50-character text string. Enclose the string in quotes if it contains any blanks. The resource class can contain any string that is meaningful to the operator viewing the Alert Monitor. For example:

```
RESCLASS('System State Manager Resource Class')
```

RESID

(Optional) Defines an identifier that uniquely identifies the resource to the application. The resource identifier can be any 1- to 128-character text string. Enclose the string in quotes if it contains any blanks. The RESID value is used for alert correlation in combination with the APPLID, CLASSID, and in some cases the TRACKID values, when you perform operations on existing alerts. For example:

```
RESID(SystemStateManager)
```

RESOURCE

(Optional) Defines the logical resource name associated with the alert. The resource name can be any 1- to 50-character text string. Enclose the string in quotes if it contains any blanks. The resource name can be a System State Manager (SSM) resource name or any other name that is meaningful to the operator viewing the Alert Monitor. For example:

```
RESOURCE(SSMRES1)
```

RESTYPE

(Optional) Defines the logical resource type associated with the alert resource. The resource type can be any 1- to 50-character text string. Enclose the string in quotes if it contains any blanks. The resource type can contain any string that is meaningful to the operator viewing the Alert Monitor. For example:

```
RESTYPE(STC)
```

SEVERITY

(Optional) Specifies the severity of the alert. When this keyword is not specified, the alert severity defaults to a value of 4 (low). The severity value must be a single numeric digit from the following list of severities:

1-Critical

2-High

3-Medium

4-Low

For example:

SEVERITY(2)

SEVUPDT

(Optional) Indicates whether the SEVERITY keyword value is used to update the severity of the alert when specified with CALLTYPE(UPDATE). This keyword is ignored with all other CALLTYPE values.

SYSACTN

(Optional) Specifies up to 20 text strings, each containing up to 75 characters. Each line of text is displayed as a separate line under the System Action heading in the CA NetMaster Alert Monitor display. Below are sample system action strings:

SYSACTN('The system will shut down if no corrective action is taken','Second system action string','Third system string')

You can insert the contents of a REXX variable into the system action string. For example, the sample below inserts the contents of variable SYSACT into the system action text as a system action string:

SYSACTN(''||SYSACT||')

Note: The SYSACVAR and SYSACTN keywords are mutually exclusive.

SYSACVAR

(Optional) Defines the stem for a set of REXX variables with each variable containing one line of system action text. You define these variables and assign lines of text to them in separate statements in the OPS/REXX program that invokes the ADDRESS NETMASTR environment. You are limited to a maximum of 20 lines of system action text and each line can contain up to 75 characters. If you specify more than 20 lines or the lines are longer than 75 characters, then the data beyond the defined limits is ignored. For example, if you specify SYSACVAR(LINE_DATA.), then the names of the variables storing lines of your system action text will be LINE_DATA.1, LINE_DATA.2, and so on.

The stem name you specify must meet standard REXX conventions for stem names. If you are not familiar with these conventions, then see the book *The REXX Language: A Practical Approach to Programming* by M. F. Cowlshaw.

Instead of a stem, the SYSACVAR keyword can also specify a valid REXX variable name prefix. For example, if you specify SYSACVAR(PREFIX_DATA), then the names of variables storing your system action text are PREFIX_DATA1, PREFIX_DATA2, and so on.

The number of lines of system action text produced depends on the number of consecutive variables that meet the criteria. Therefore, in the following example:

```
sysacvar.1='The system will shut down if no corrective action is taken'
sysacvar.2='Second system action string'
sysacvar.3='Third sys string'
```

A command specifying ADDRESS NETMASTR SYSACVAR("sysacvar.") produces only three lines of system action text.

Note: The SYSACVAR and SYSACTN keywords are mutually exclusive.

SYSTEMNM

(Optional) Defines the system name associated with the alert. The resource type can be any 1- to 8-character text string. For example:

```
SYSTEMNM(SOMESYS)
```

TEXT

(Optional) Specifies up to 20 text strings, each containing up to 75 characters. Each line of text is displayed as a separate line under the Alert Text heading in the CA NetMaster Alert Monitor display. Below are sample text strings:

```
TEXT('text string1','second text string','third text string')
```

You can insert the contents of a REXX variable into the message text. For example, the sample below inserts the contents of variable FIELD1 into the message text as a text string:

```
TEXT("||FIELD1||")
```

Note: The TEXTVAR and TEXT keywords are mutually exclusive.

TEXTUPDT

(Optional) Indicates whether the DESC, EXPLAIN, EXPLVAR, RECMACTN, RECMVAR, SYSACTN, SYSAVAR, TEXT, and TEXTVAR keyword values are used to update the alert when specified with CALLTYPE(UPDATE). This keyword is ignored with all other CALLTYPE values.

TEXTVAR

(Optional) Defines the stem for a set of REXX variables with each variable containing one line of alert text. You define these variables and assign lines of message text to them in separate statements in the OPS/REXX program that invokes the ADDRESS NETMASTR environment. You are limited to a maximum of 20 lines of message text and each line can contain up to 75 characters. If you specify more than 20 lines or the lines are longer than 75 characters, then the data beyond the defined limits is ignored. For example, if you specify TEXTVAR(LINE_DATA.), then the names of the variables storing lines of your alert text will be LINE_DATA.1, LINE_DATA.2, and so on.

The stem name you specify must meet standard REXX conventions for stem names. If you are not familiar with these conventions, then see the book *The REXX Language: A Practical Approach to Programming* by M. F. Cowlishaw.

Instead of a stem, the TEXTVAR keyword can also specify a valid REXX variable name prefix. For example, if you specify TEXTVAR(PREFIX_DATA), the names of variables storing your text are PREFIX_DATA1, PREFIX_DATA2, and so on.

The number of lines of alert text produced depends on the number of consecutive variables that meet the criteria. Therefore, in the following example:

```
LineData.1="Line1"  
LineData.2="Line2"  
LineData.3="Line3"
```

A command specifying ADDRESS NETMASTR TEXTVAR("LineData.") produces only three lines of text data.

Note: The TEXTVAR and TEXT keywords are mutually exclusive.

TRACKID

(Optional) Defines a unique 1- to 32-character *TrackingID* or correlation token that must be specified on the CREATE request, if you want to be able to perform certain operations (ANNOTATE, CLOSE, and SET) on the same alert in the future. To do so, you should save the *TrackingID* that you specify on the CREATE request for later use. You may choose to define and use a set of unique *TrackingIDs* for use by your OPS/REXX applications or you may choose to dynamically generate unique tokens on each request. A good mechanism for doing so would be to use OPSINFO("SYSNAME") || C2X(OPSINFO("CLOCK")) as the TRACKID value.

Example:

```
TRACKID(AnyIDThatYouLike12345)
```

Note: CA NetMaster internally appends the APPLID to the TRACKID so that you cannot create a token that conflicts with another application.

Examples: ADDRESS NETMASTR

■ Create an alert

This example uses the ADDRESS NETMASTR host environment to create an alert on the CA NetMaster Alert Monitor screen:

```
/*Create an alert */
txtvar.1 = "The named resource is in an alert logical state"
txtvar.2 = "Reason: The system needs to be restarted"
recmavar.1 = "Restart the system to correct the error"
recmavar.2 = "Second recommended action string"
recmavar.3 = "Third rec string"
sysacvar.1 = "The system will shut down if no"
sysacvar.2 = "corrective action is taken"
address NETMASTR "Alert callTYPE(CR) TRACKID(MyTrackingID)" ,
  "DESC('This is a test alert from CA OPS/M/S)" ,
  "RESOURCE(SSMRES1) APPLID(OPSA) Severity(3)" ,
  "RESCLASS('System State Manager Resource Class) RESTYPE(STG)" ,
  "TEXTVAR(txtvar.)" ,
  "RECMVAR(recmavar.)" ,
  "SYSACVAR(sysacvar.)" ,
  "SYSTEMNM(SYSA)"
if RC > 0 then
do
  say "address NETMASTR failed, RC="RC
  do QUEUED()
  parse pull line
  say line
end
end
```

- Update the severity of the alert

This example uses the ADDRESS NETMASTR host environment to update the severity of the alert created in the previous example:

```
/* Update the alert */
address NETMASTR "Alert callTYPE(UPD)" ,
  "RESOURCE(SSMRES1) APPLID(OPSA) Severity(2) SEVUPDT(YES)" ,
  "RESCLASS(System State Manager Resource Class)"
if RC <> 0 then
do
  say "address NETMASTR failed, RC="RC
do QUEUED()
  parse pull line
  say line
end
end
```

- Annotate the alert

This example uses the ADDRESS NETMASTR host environment to annotate the alert created above:

```
/* Annotate the alert */
address NETMASTR "Alert callTYPE(AN)" ,
  "APPLID(OPSA)" ,
  "TRACKID(MyTrackingID)" ,
  "ANUSERID(USERID1) ",
  "TEXT(Annotating this alert,More annotation text)"
if RC <> 0 then
do
  say "address NETMASTR failed, RC="RC
do QUEUED()
  parse pull line
  say line
end
end
exit
```

- Close the alert

This example uses the ADDRESS NETMASTR host environment to close the alert created above:

```
/* Close the alert */
address NETMASTR "Alert callTYPE(CLO)" ,
  "APPLID(OPSA)" ,
  "TRACKID(MyTrackingID)"
if RC <> 0 then
do
  say "address NETMASTR failed, RC="RC
do QUEUED()
  parse pull line
  say line
end
end
```

Note: In the above examples, only the APPLID and TRACKID keywords are used to correlate alerts.

ADDRESS NETMASTR Return Codes

The ADDRESS NETMASTR host environment produces these return codes:

0

The CA NetMaster alert was successfully passed to CA NetMaster.

1

An attempt to free TEXTVAR storage failed.

2

An attempt to free EXPLVAR storage failed.

3

An attempt to free RECMVAR storage failed.

4

An attempt to free SYSACVAR storage failed.

9

An attempt to free NM buffer storage failed.

10

GETMAIN for the NM buffer area failed.

11

GETMAIN for the TEXTVAR area failed.

12

GETMAIN for the EXPLVAR area failed.

13

GETMAIN for the RECMVAR area failed.

14

GETMAIN for the SYSACVAR area failed.

21

An abend occurred. See message OPS9600x and the associated mini-dump.

22

An abend occurred in the CA NetMaster interface. See message OPS9500x and the associated mini-dump.

23

A non-zero return code from the CA NetMaster interface (\$NMXEVT).

31

The command string is too long.

38

There is a syntax error in the command.

39

A scan error occurred while parsing the command.

41

An error occurred retrieving TEXTVAR variables.

42

An error occurred retrieving EXPLVAR variables.

43

An error occurred retrieving RECMVAR variables.

44

An error occurred retrieving SYSACVAR variables.

51

Invalid APPLID.

Note: The return code from ADDRESS NETMASTR does not provide any indication of whether CA NetMaster generated the alert. A zero return code means that a CA NetMaster system accepted the internal command passed to it.

ADDRESS OPER Host Environment

The ADDRESS OPER host environment permits you to issue operator commands from an OPS/REXX program or AOF rule. The ADDRESS OPER OPS/REXX host environment performs automated tasks with your CA OPS/MVS AOF rules and OPS/REXX programs, such as issuing z/OS, JES, subsystem, or product specific commands.

Use ADDRESS OPER to issue the following types of operator commands:

IMS Type 2 Commands

IMS Type-2 format commands are designed to be issued to the IMSPLEX Operations Manager, not any specific IMS system, and are identified to ADDRESS OPER by the presence of the keyword IMSPLEX. IMS provides a complete range of Type-2 commands, all of which must be issued in Type-2 format, and also allows some existing Type-1 commands to be issued directly to the Operations Manager using the Type-2 syntax. Upon receiving a Type-2 command, the manager distributes the work to participating IMS systems, and returns all collected output to the original issuer of the command.

The SYSID/SYSTEM parameters provide a method for remote execution. If a command is directed to a remote system through an MSF connection, the remote system must have the following configured properly:

- An active IMS configured for Type2 message execution
- Any IMSPLEX considerations unique to that system

System-related keywords such as the IMSPLEX name, apply to the remote system. The sending system does not perform any IMS functions, and does not require an active IMS to be running.

JES

Prefix JES2 commands with a dollar sign (\$). Prefix JES3 commands with an asterisk (*).

MVS (or MVS/JES3)

There is a limit to the number of output lines CA OPS/MVS can capture, but it is so large that truncation is unlikely.

VM

Prefix VM commands with a pound sign followed by the letters CP (#CP). The CA OPS/MVS product issues VM commands using the standard z/VM Diagnose Interface, which has these requirements:

- A buffer, which is used to return the response to the VM command, must be passed to the Diagnose Interface.
- The buffer must be contiguous in real memory (as seen by z/OS).

Because there is no z/OS mechanism for allocating contiguous real memory, VM command responses are limited to the size of a single real storage frame, which is 4 KB.

More information:

[REXX/EDQ Output from IMS Type 2 Message Responses](#) (see page 793)

Utilize the ADDRESS OPER Host Environment

Effective usage of this host environment is determined by the programmatic need to process or not to process the command response output. Meaning, should the automation:

- Dispatch a command and continue
- Issue a command and programmatically process the command responses

Use the guidelines in this section to utilize the ADDRESS OPER host environment to accommodate either method of issuing commands within your automation.

Dispatch a Command and Continue Processing

To issue a command without the need to programmatically process the command output:

- Utilize the NOOUTPUT keyword

```
ADDRESS OPER  
"Command(text of command) Noutput"
```

This can be abbreviated to:

```
ADDRESS OPER  
"C(text of command) Nbo"
```

- In a NOOUTPUT condition, specify (if needed) the keywords IMSID, MFORM, CONNAME, PROPRES (from within an AOF CMD rule only), and SYSTEM.
- The NOOUTPUT commands require CA OPS/MVS to use the console as set through the OCCONSOLENAME parameter. Obtain optimal efficiency for automation issuing the NOOUTPUT commands by following these rules:
 - Ensure that the setting of the OCCONSOLENAME parameter is that of a unique CA OPS/MVS extra extended console. For more information on the EXTRAEXTCONSOLES and EXTRAEXTPREFIX parameters that are used during initialization to allocate extended consoles, see the *Parameter Reference*.
 - Make the OCCONSOLENAME parameter change permanent by incorporating the change in the CA OPS/MVS start up REXX exec.
 - Authorize the issuing console, specified through the OCCONSOLENAME parameter, as required by the various products so the automation can issue NOOUTPUT commands to it. For example, in the case of a CICS command, the console may need to be defined to the CICS TCT table.

More information:

[ADDRESS OPER Keywords](#) (see page 231)

Issue a Command and Process the Command Response

Utilize the ADDRESS OPER host environment to issue a command and programmatically process the command responses

To issue a command and programmatically process the command output

1. Ensure that an OPS/REXX function does not exist for the command response output that you want to process.

OPS/REXX functions are an integral component of CA OPS/MVS that retrieve a wide variety of system data needed to make programmatic decisions within AOF rules and OPS/REXX programs. These functions are much more efficient than the equivalent system command, and in some cases retrieve data that cannot be retrieved with any system command. Additionally, these functions allow you to obtain the information synchronously from within AOF rules.

2. Since no waiting is allowed from within AOF rules, except in Request (REQ) rules, the NOOUTPUT option is always enforced and no command response output is collected for programmatic processing. To circumvent this and successfully collect and process command responses, you can dispatch or trigger an OPS/REXX program from an AOF rule to a CA OPS/MVS OSF TSO class server as shown in this example:

```
)MSG IST521I
)PROC
/* React to GBIND failures if the from node is A45PROD. */
/* The logic of this rule simply interrogates the message */
/* text to see if this is a GBIND failure for the A45PROD */
/* node. If it is, then we need to trigger an EXEC called */
/* IST521I to run in an OSF TSO server. This is needed */
/* because we need to issue VTAM commands and interrogate */
/* the command output, and waiting can't be done from */
/* within the AOF rule environment. */
/* */
/* IST521I GBIND failed for ISTCOS from A45PROD to A04X99 */
frommode = WORD(MSG.TEXT,7)
tonode = WORD(MSG.TEXT,9)
if frommode = 'A45PROD' then return /* NOT A45PROD */
/* Trigger OPS/REXX program IST521I to an OPSOSF server */
/* passing it the nodes as arguments. IST521I would be */
/* located in the OPSEEXEC or SYSEEXEC concatenation of the */
/* OPSOSF procedure. This program would then contain the */
/* Address OPER host environment command to issue the */
/* needed VTAM display command and successfully */
/* interrogate the command output. */
Address Osf
"OI PROGRAM(IST521I) ARG("frommode tonode") "
```


3. Specify one or more command output trapping keywords to override the default command response collection mechanisms. The required trapping keywords depend on how the command generates the command responses. Command responses can be any of the following:

- Single MLWTO command responses
- Multiple sets of MLWTO command responses
- Individual WTO command responses
- Command responses not directed back to the issuing console

For examples of specific usage of the ADDRESS OPER for each type of command response, see the Example section.

4. Utilize the OPSLOG display columns OPSFLAGS and CONSNAME to determine the type of command response that is being generated.

The values of the OPSFLAGS column identifies the message type (WTO, or MLWTO), and the CONSNAME column identifies if the responses are being directed back to the console that issued the command.

5. Determine the bit representations of the OPSFLAGS column by placing the cursor on the response line within the OPSLOG and hit enter.

The Display All Columns for MSG Event pop-up panel displays.

6. Display the field descriptions by placing the cursor on the OPSF field and hit enter.

The field description displays.

For a description of each bit representation of the OPSFLAGS column, see the chapter “Message Rules” of the *AOF Rules User Guide*.

Usage of ADDRESS OPER

These examples show specific usage of the ADDRESS OPER for each type of command response.

- Command and Collected Output in the Single Line MLWTO Form

A majority of command responses are in the form of single line MLWTOS as illustrated with the following D NET,ID command. For a single MLWTO command response, the most effective keywords are CMDWAIT(xx), and INTERVAL(0).

Single MLWTO-generated command response output as extracted from the OPSLOG:

```

OPSLOG Browse OPS11Q -CA11 --- OPSVIEW --- 07:17:12
OPSF CntlName ---1---2---3---4-
8000 OPS11X01 D NET,ID=A11IOPS,E
2408 OPS11X01 IST075I NAME = A11IOPS, TYPE = APPL SEGMENT
2408 OPS11X01 IST486I STATUS= ACTIV, DESIRED STATE= ACTIV
2408 OPS11X01 IST360I APPLICATIONS:
2408 OPS11X01 IST080I A11IOPSA CONCT A11IOPSB CONCT
2408 OPS11X01 IST080I A11IOPSD CONCT A11IOPSE CONCT
2408 OPS11X01 IST080I A11IOPSG CONCT A11IOPSH CONCT
2408 OPS11X01 IST080I A11IOPSJ CONCT A11IOPSK CONCT
2408 OPS11X01 IST080I A11IOPSM CONCT A11IOPSN ACT/S
2408 OPS11X01 IST080I A11IOPSP ACT/S A11IOPSQ ACT/S
2408 OPS11X01 IST080I A11IOPSS CONCT A11IOPST ACT/S
2408 OPS11X01 IST080I A11IOPSW ACT/S A11IOPSX CONCT
2408 OPS11X01 IST080I A11IOPSZ CONCT A11IOPSV ACT/S
2608 OPS11X01 IST314I END

```

Using this VTAM display command as an example, the following code illustrates issuing a command and collecting output that is returned as a SINGLE MLWTO (Primary line, followed by data lines, with one end line for end of response) back to the issuing CA OPS/MVS console.

```

/*-----*/
/* Start out with a clean EDQ before issuing the command. */
/*-----*/
clearedq = OPSLEDQ()
/*-----*/
/* Issue the VTAM display command using the necessary */
/* command response trapping keywords.
/*-----*/
ADDRESS OPER
"Command(D NET,ID=A11IOPS,E) Cmdwait(30) Interval(0)"
/*-----*/
/* Simply display back the captured command responses. The */
/* collected command responses are destined for the */
/* External Data queue of the program and can be extracted */
/* via the FULL instruction. */
/*-----*/
cmdoutlines = QUEUED()
do cmdoutlines
  pull cmdoutput
  say ">OUTPUT=cmdoutput
end

```

The following keywords are used within the above sample:

Command(D NET,ID=A11OPS,E)

Specifies the text of the command to be issued (VTAM Display command in this case).

CMDWAIT(30)

Sets a CONDITIONAL wait time of 30 seconds. Meaning the total wait time for collecting output will be up to 30 seconds or until some stop trapping condition occurs. The stop trapping condition in the case of the single line MLWTO response will be the collection of the end-line of the command response.

Interval(0) keyword

Disables the stop trapping mechanism that measures the delivery time of each collected command output line.

■ Command and Collected Output in the Multiple Line MLWTO Form

Some command responses are in the form of multiple MLWTOs as illustrated with the following JES2 \$DF command. For multiple MLWTO command responses that are directed back to the CA OPS/MVS console in which the command was issued, the most effective keywords are CMDWAIT(xx), INTERVAL(xx), and STOPEND(NO).

Multiple MLWTO-generated command response output as extracted from the OPSLOG:

```

OPSLOG Browse OPS11Q - CA11 --- OPSVIEW -- 09:50:25 05JUN2009 COLS 001 05
OPSF CnsName WTO ID ---1---2---3---4---5---
8000 OPS11X01 00000000 $DF
3008 OPS11X01 1B0724D2 $HASP621 OUT R=LOCAL
3408 OPS11X01 1B0724D2 $HASP621 OUT R=LOCAL F=DPLX C=*** T=*
3608 OPS11X01 1B0724D2 $HASP621 FRMODE=LINE CLASS A=24
3008 OPS11X01 1B0724D3 $HASP621 OUT R=LOCAL
3408 OPS11X01 1B0724D3 $HASP621 OUT R=LOCAL F=GARY C=*** T=*
3608 OPS11X01 1B0724D3 $HASP621 FRMODE=LINE CLASS A=1
3008 OPS11X01 1B0724D4 $HASP621 OUT R=LOCAL
3408 OPS11X01 1B0724D4 $HASP621 OUT R=LOCAL F=OBER C=*** T=*
3608 OPS11X01 1B0724D4 $HASP621 FRMODE=LINE CLASS B=12 M=5
3008 OPS11X01 1B0724D5 $HASP621 OUT R=LOCAL
3408 OPS11X01 1B0724D5 $HASP621 OUT R=LOCAL F=STD C=*** T=*
3608 OPS11X01 1B0724D5 $HASP621 FRMODE=LINE CLASS A=1578 B=14 G=1
3608 OPS11X01 1B0724D5 $HASP621 CLASS P=95 R=95 S=3 T=5 X=21
3008 OPS11X01 1B0724D6 $HASP621 OUT R=LOCAL
3408 OPS11X01 1B0724D6 $HASP621 OUT R=LOCAL F=STD C=*** T=*
3608 OPS11X01 1B0724D6 $HASP621 FRMODE=LINE CLASS X=1

```

Using this JES2 \$DF display command as an example, the following code illustrates issuing a command and collecting output that is returned as a multiple MLWTO (Primary line, followed by data lines, an end-line, then another sequence of primary line, data line, end-line etc. Notice the WTOID OPSLOG display in the \$DF output that changes with each MLWTO occurrence.)

```
/*-----*/
/* Start out with a clean EDQ before issuing the command. */
/*-----*/
clearedq = OPSQLEDQ()
/*-----*/
/* Issue the JES2 $DF command using the necessary command */
/* response trapping keywords. */
/*-----*/
address Oper
"Command($DF) CMDWAIT(30) INTERVAL(300) STOPEND(NO)"
/*-----*/
/* Simply display back the captured command responses. The */
/* collected command responses are destined for the */
/* external data queue of the program and can be */
/* extracted and processed via the PULL instruction. */
/*-----*/
cmdoutlines = QUEUED()
do cmdoutlines
  pull cmdoutput
  say ">Output="cmdoutput
end
```

The above sample uses the following keywords:

Command(\$DF)

Specifies the text of the command to be issued (JES2 \$DF Display command in this case).

CMDWAIT(30)

Sets a CONDITIONAL wait time of 30 seconds. Meaning the total wait time for collecting output will be up to 30 seconds or until some stop trapping condition occurs. The stop trapping condition in the case of this JES2 \$DF command response will be the INTERVAL condition as noted below.

Interval(300) keyword

Sets the stop trapping mechanism that measures the delivery time between each collected command output line to be that of 300 centiseconds. Therefore, when the delivery time of 300 centiseconds occurs, the response trapping will terminate.

The STOPRESP(message ID) keyword could optionally be utilized if the specified message ID indicates that all the desired output has been issued. Specific use of the STOPRESP keyword can be found in the example titled Using the STOPRESP and STOPMSG keywords to terminate response processing.

■ Command and Collected Output in the SingleLine WTO Form

Some command responses are in the form of Singleline WTOs as illustrated with the following z/OS Reset command. For singleline WTO command responses that are directed back to the CA OPS/MVS console in which the command was issued, the most effective keywords are CMDWAIT(xx), INTERVAL(0) and STOPRESP(messageid).

Singleline WTO generated command response output as extracted from the OPSLOG:

```
OPSLOG Browse OPS11Q - CA11 --- OPSVIEW -- 09:50:25 05JUN2009 COLS 001 05
OPSF CnsName WTO ID ---1---2---3---4---5---
0000 OPS11X01 00000000 E PCPDA115,SRV/CLASS=BATSTWLM
8008 OPS11X01 1B0724D2 IEE304I PCPDA115 JOB RESET
```

Using this z/OS Reset command as an example, the following code illustrates issuing a command and collecting output that is returned in the form of singleline WTOs.

```
/*-----*/
/* Accept name of batch job from invoking MSG rule */
/*-----*/
arg jobname
/*-----*/
/* Start out with a clean EDQ before issuing the command. */
/*-----*/
clearedq = OPSCLEDDQ()
/*-----*/
/* Issue the OS/390 reset command using the necessary */
/* command response trapping keywords. */
/*-----*/
address Oper
"Command(E "jobname",SRV/CLASS=BATSTWLM)",
"CMDWAIT(30) INTERVAL(0) STOPRESP(IEE304I)"
/*-----*/
/* Simply display back the captured command responses. The */
/* collected command responses are destined for the */
/* external data queue of the program and can be */
/* extracted and processed via the PULL instruction. */
/*-----*/
cmdoutlines = QUEUED()
do cmdoutlines
pull cmdoutput
say ">Output="cmdoutput
end
```

The following keywords are used within the above sample:

Command(E "jobname",SRVCLASS=BATSTWLM)

Specifies the text of the command to be issued with variable substitution. In this case, the z/OS Reset command.

CMDWAIT(30)

Sets a CONDITIONAL wait time of 30 seconds. Meaning the total wait time for collecting output will be up to 30 seconds or until some stop trapping condition occurs. The stop trapping condition in the case of this z/OS reset command response will be the STOPRESP condition as noted below.

Interval(0) keyword

Disables the stop trapping mechanism that measures the delivery time between each collected command output line. This mechanism is being disabled since the command response in the case of the z/OS Reset command generates a specific message ID that can be used within the STOPRESP() keyword.

STOPRESP(IEE304I) keyword

Sets the stop trapping mechanism to be that of the command response message IEE304I. Meaning with CMDWAIT(30), the total time of command response collection will be up to 30 seconds or until the message IEE304I is collected as set using this STOPRESP keyword.

- Command and Collected Output for Responses not Directed Back to the Issuing Console

Typically, most command responses are internally directed back to the console that issued the command. However, there are some product specific commands in which the command responses are directed elsewhere; such as to the master console or the system log. For command responses that are *not* directed back to the CA OPS/MVS console in which the command was issued, the most effective keywords are CMDWAIT(xx), INTERVAL(0), CAPTURE(message, messages, or message prefix) and STOPMSG(message ID).

Single line WTO generated command response output as extracted from the OPSLOG:

```

OPSLOG Browse OPS11Q - CA11 -- OPSVIEW -- 09:50:25 05JUN2006 COLS 001 05
CrsName Message ID ---+---1---+---2---+---3---+---4---+---5---
OPS11X01 MODIFY F PTX0099,DISPLAY
NONE PXMD105 PXMD105 CA/XMANAGER MODIFY IN PROGRESS XMD=1154
NONE PXMD107 PXMD107 MODIFY TEXT = DISPLAY
NONE PXMD111 PXMD111 XMANAGER DISPLAY WORKLOAD IN PROGRESS
NONE PXMD112 PXMD112 XMANAGER CURRENTLY HAS NO WORKLOAD
NONE PXMD109 PXMD109 XMANAGER MODIFY PROCESSING COMPLETE

```

Using this CA XMANAGER Display command as an example, note the CONSNAME field of the OPSLOG for the command responses not being sent back to console that issued the DISPLAY command. The following code illustrates issuing a command and collecting output not routed back to the issuing console using this CA XMANAGER command as an example.

```
/*-----*/
/* Start out with a clean EDQ before issuing the command. */
/*-----*/
clearedq = OPSLEDQ()
/*-----*/
/* Issue the XMANAGER command using the necessary      */
/* command response trapping keywords.                */
/*-----*/
address Oper
"Command(F PTX0099,DISPLAY) Cmdwait(30)",
"Capture(PXM) Interval(0) Stopmsg(PXM0109)"

/*-----*/
/* Simply display back the captured command responses. The */
/* collected command responses are destined for the      */
/* external data queue of the program and can be        */
/* extracted and processed via the PULL instruction.     */
/*-----*/
cmdoutlines = QUEUED()
do cmdoutlines
  pull cmdoutput
  say ">Output="cmdoutput
end
```

The following keywords are used within the above sample:

Command(F PTX0099,DISPLAY)

Specifies the text of the command to be issued.

Cmdwait(30)

Sets a CONDITIONAL waittime of 30 seconds. This sets the total wait time for collecting output to up to 30 seconds or until some stop trapping condition occurs. The stop trapping condition in the case of this XMANAGER command response will be the STOPMSG condition as noted below.

Capture(PXM)

By default, only console directed command responses are collected. This keyword causes the code to collect or capture these all nonconsole directed PXM messages.

Interval(0) keyword

Disables the stop trapping mechanism that measures the delivery time between each collected command output line. This mechanism is being disabled since the command response in the case of the XMANAGER Display command generates a specific message ID that can be used within the STOPMSG() keyword.

Stopmsg(PXM0109) keyword

Sets the stop trapping mechanism to be that of the command response message PXM0109. This means that with CMDWAIT(30), the total time of command response collection will be up to 30 seconds or until the message PXM0109 is collected as set through this STOPMSG keyword.

- Using the STOPRESP Keyword

This example uses the STOPRESP Keyword to collect command responses issued back to the issuing console in the form of multiple MLWTOS with a unique ending message.

For command responses that are directed back to the issuing CA OPS/MVS console in the form of multiple MLWTOS, and that contain a unique message ID indicating all desired output has been sent, the most effective keywords are CMDWAIT(xx), INTERVAL(0), STOPEND(NO) and STOPRESP(messageid).

Multiple MLWTO command response with a unique message that indicates end of response as extracted from the OPSLOG:

```

OPSLOG Browse OPS11Q - CA11 --- OPSVIEW -- 10:33:54 12JUN2006 COLS 001 065
OPSF CnsName ---+1--+2--+3--+4--+5--+6--+
8000 OPS11X01 FPIADIS DB(DSNDB01) SPACENAM(DBD01) LIMIT(*) LOCKS
8008 OPS11X01 DSN360I FPIA *****
2808 OPS11X01 DSN361I FPIA * DISPLAY DATABASE SUMMARY
2E08 OPS11X01 * GLOBAL LOCKS
8008 OPS11X01 DSN360I FPIA *****
2808 OPS11X01 DSN362I FPIA DATABASE = DSNDB01 STATUS = RW
2E08 OPS11X01 DBD LENGTH = 8000
2808 OPS11X01 DSN397I FPIA
2008 OPS11X01 NAME TYPE PART STATUS CONND CORRID LOCKI
2008 OPS11X01 -----
2008 OPS11X01 DBD01 TS RW H,S,P
2008 OPS11X01 - MEMBER NAME FPIA
2008 OPS11X01 DBD01 TS RW H,S,P
2008 OPS11X01 - MEMBER NAME FPIA
2E08 OPS11X01 ***** DISPLAY OF DATABASE DSNDB01 ENDED *****
8008 OPS11X01 DSN8022I FPIA DSN3DDIS 'DISPLAY DATABASE NORMAL COMPLETION

```


Using this DB2 display command as an example, note the multiple MLWTO response and the unique DSN9022I message that indicates the response is complete. The following code illustrates issuing this type of command and collecting the output.

```

/*-----*/
/* Start out with a clean EDQ before issuing the command. */
/*-----*/
clearedq = OPSQLEDQ()
/*-----*/
/* Issue the DB2 Thread display command */
/*-----*/
address Oper
"CPP1ADIS DB(DSNDB01) SPACENAM(DBD01) LIMIT(*) LOCKS",
"CMDWAIT(30) STOPRESP(DSN9022I) INTERVAL(0) STOPEND(NO)"
/*-----*/
/* Simply display back the captured command responses. The */
/* collected command responses are destined for the */
/* external data queue of the program and can be */
/* extracted and processed via the PULL instruction. */
/*-----*/
cmdoutlines = QUEUED()
do cmdoutlines
pull cmdoutput
say **>Output=c:cmdoutput
end

```

Command(IPP1ADIS DB(DSNDB01) SPACENAM(DBD01) LIMIT(*) LOCKS)

Specifies the text of the command to be issued.

Cmdwait(30)

Sets a CONDITIONAL wait time of 30 seconds. Meaning the total wait time for collecting output will be up to 30 seconds or until some stop trapping condition occurs. The stop trapping condition in the case of this DB2 command response will be the STOPRESP condition.

Interval(0) keyword

Disables the stop trapping mechanism that measures the delivery time between each collected command output line. This mechanism is being disabled since the command response in the case of this DB2 Display command generates a specific message ID that can be used within the STOPRESP() keyword.

Stopresp(DSN9022I) keyword

Sets the stop trapping mechanism to be that of the command response message DSN9022I being sent back to the issuing console. This means that with CMDWAIT(30), the total time of command response collection will be up to 30 seconds or until the message DSN9022I is collected as set via this STOPRESP keyword.

- Issuing IMS Type 2 Messages

These examples both issue the QUERY IMSPLEX command in IMS Type 2 Message format.

1. The following example specifies an explicit IMSPLEX name and allows a wildcard IMSID:

`"C(QUERY IMSPLEX) CMDRESP(REXX) MSID(*) MSPLEX(IPLEX)"`
2. The following example specifies an explicit IMSID, IMSS, and will be handled by the IMSPLEX that system IMSS belongs to.

`"C(QUERY IMSPLEX) CMDRESP(REXX) MSID(IMSS) MSPLEX(*)"`

ADDRESS OPER Security With Audit Ability

Because you can use the ADDRESS OPER host command environment to enter commands that can crash JES, z/OS, or VM, we have built security measures into ADDRESS OPER. The default security authorization exit for ADDRESS OPER commands issues an error message and terminates if you do not have OPER authority. ADDRESS OPER also terminates if you specify any of the following commands:

- IMS command-/CHE
- JES3 commands-*DUMP, *RETUR
- z/OS command-QUIESCE
- VM commands-CP, #CP, I, IPL, LOG, LOGOFF, LOGOUT, SHUTDOWN, SYS, SYSTEM

All commands issued through the ADDRESS OPER host command environment go into the system log, prefaced with message ID OPS1181H.

ADDRESS OPER Keywords

The ADDRESS OPER host environment supports different keywords, depending on whether the ADDRESS OPER command is issued from within an OPS/REXX program or an AOF rule.

- In order to issue IMS Type 2 commands, access must be available to IBM modules CSLSRG00 and CSLSDR00. Those are shipped in IMS product libraries at release 9.1 or greater, and are usually named *hlq.SDFSRESL*. The entire IMS product library can be allocated, or a private data set with just those isolated modules, and may be an explicit allocation, or a LNKLIST entry.
- If you use the ADDRESS OPER host environment to issue an operator command from an OPS/REXX program, you may include any of the keywords listed under Format 2 in your command string.
- If you use the ADDRESS OPER host environment to issue an operator command from an AOF rule that *allows WAITs* (a REQ, SCR, or TOD rule), you may include any of the keywords listed under Format 2 in your command string.
- If you use the ADDRESS OPER host environment to issue an operator command from an AOF rule that *does not allow WAITs* (any rule other than REQ, SCR, or TOD), the only keywords listed under Format 2 that you may specify in your command string are IMSID, MFORM, NAME/CONNAME, and SYSTEM /SYSID.
- You cannot issue IMS Type 2 message from a rule. Type 2 message implementation is based on internal event WAIT and POST completions, and as such is not appropriate within a rule environment.
- The IMS Type 2 message interface is a direct API call to the IMSPLEX manager, and not an operator message. All responses are returned through the same API. As a result, BMP is not involved, and keywords dealing with response disposition and routing are not applicable.

Formats for ADDRESS OPER Commands

Use either of the following formats when using the ADDRESS OPER host command environment. Use format 1 when you do not want to specify any additional keywords.

1. You may use this format for ADDRESS OPER commands:

```
ADDRESS OPER "keywords" {text}
```

2. You may also use this format for ADDRESS OPER commands. If you specify the COMMAND keyword, it must precede any other keywords you specify.

```
ADDRESS OPER "keywords"  
[BMPCMDOUT(OPSLOG|WTO|NONE)]  
[COMMAND(text)]  
[CAPTURE(msgtextlist)]  
[CVDECHO(YES|NO)]  
[CVDLOG(YES|NO)]  
[CVDRESP(REXX)]  
[CVDWAIT(seconds)]  
[CONTYPE(ANY|EXTCONS|SSCONS|XTRACONS)]  
[DELAY(seconds)]  
[MSID(name*|list)]  
[MSPLEX(name*|*)]  
[MSREPLY]  
[INTERVAL(centiseconds)]  
[LOCALONLY]  
[LOG(YES|NO|OFF)]  
[MAXCMDOUT(number)]  
[NAME|CONNAME(consolename)]  
[OUTPUT|NOOUTPUT]  
[PREFIX(name)]  
[PROPRESP]  
[STOPEND(YES|NO)]  
[STOPMSG(msgtextlist)]  
[STOPRESP(msgtextlist)]  
[SYSID|SYSTEM(systemids)]  
[SYSWAIT(seconds)]  
[VERBOSE(YES|NO)]  
[WAIT(seconds)]  
[XML(YES|NO)]
```

BMPCMDOUT Keyword

(Optional) Controls echoing of the current IMS command output. This keyword only applies if you are using an IOF BMP for issuing IMS commands. Possible values are:

OPSLOG

Echoes all output associated with the current IMS command back to OPSLOG as trace messages. This results in a small amount of additional overhead to the command (assuming that the number of lines of output is not large). No output is sent back to any z/OS console or to the subsystem interface.

WTO

Echoes all output associated with the current IMS command as Write To Operator messages to the z/OS console. Note that using this option can add a significant amount of overhead to IMS command processing and has the potential of flooding the consoles with command output messages.

NONE (Default)

Sends IMS commands to the BMP and returns the responses to the issuing program (if so requested). This recommended approach results in the lowest amount of overhead.

The use of this keyword does not affect the way in which the IMS command output is returned to the command issuer.

COMMAND(text) or text Keyword

(Optional) Indicates the text of the operator command you want to issue.

Note: If you specify the COMMAND keyword, it must precede any other keywords you specify.

CAPTURE(msgtextlist)

(Optional) Captures command responses that are not internally routed back to the CA OPS/MVS console that issued the command. For correct usage of this keyword, see Utilizing the ADDRESS OPER Host Environment.

The CAPTURE keyword captures command response messages based on text segments matching the first characters (starting with column 1) of the messages. The command response messages captured by specification of this keyword are in addition to the usual command response messages captured by ADDRESS OPER, based on other specified keywords. You can specify up to ten message text segments with this keyword, and each text segment can be up to 124 characters.

For example:

```
CAPTURE(xyz, jstr0004,IST02)
```

captures these messages:

```
xyz003
xyztest
xyz77 Fred is at lunch
xyztest This is a test
jbst0004AX10
ist02test
ist02job
IST02 Today is Wednesday
```

If a text segment contains blanks, commas, quotes, or parentheses, you must enclose that segment in quotes. For example, the second segment in the following list contains blanks, so it is enclosed in single quotes:

```
CAPTURE(IST02,' JOBS MS TS')
```

All message text segments specified on this keyword will match both uppercase and lowercase text in potential matching messages. For example, this segment:

```
CAPTURE('OPX0996I END OF')
```

Matches both of these messages:

```
OPX0996I end of exec
OPX0996I END OF JOB
```

The **CAPTURE** keyword accepts the wildcard character *. If you specify **CAPTURE(*)**, all messages are trapped.

Use the **CAPTURE** keyword to trap command responses that are not directed to the console where the command was issued. You can specify the **CAPTURE** keyword with either the **STOPMSG** keyword or the **STOPRESP** keyword.

CMDECHO(YES|NO)

(Optional) Instructs ADDRESS OPER to capture or omit the command echo line from the command response. You can specify either YES, which is the default, or NO.

Default: YES

CMDLOG(YES|NO)

(Optional) Determines whether the echo line from the command issued by ADDRESS OPER goes into the SYSLOG data set and the command response. You can specify either YES, which is the default, or NO.

Default: YES

CMDRESP(REXX)

When used to issue an IMS Type 2 message, **CMDRESP(REXX)** can be specified to create REXX variables. If **CMDRESP** is omitted, the output will be written to the EDQ. This option is only valid if IMS Type 2 message syntax is used.

For more information, see the section **REXX Output from IMS Type 2 Message Responses**.

CMDWAIT(seconds)

(Optional) Specifies, in seconds, how long ADDRESS OPER waits for command output collection to complete. ADDRESS OPER examines the number of lines collected every few hundredths of a second, based on the value specified with the INTERVAL keyword. If no new output lines are forthcoming after at least two output lines have been received, ADDRESS OPER terminates the response before the CMDWAIT period expires. The CMDWAIT period can be 1 to 600 seconds.

If the ADDRESS OPER includes the STOPEND(YES) keyword, the end line of a multi-line WTO message also stops response collection. The main difference between the CMDWAIT and WAIT keywords is that WAIT specifies an unconditional wait period, while CMDWAIT specifies a conditional wait based on a continuous, timely collection of response lines. For correct usage of this keyword, see the 'Utilizing the Address Oper' and the 'Examples' section.

Default: The value of the CA OPS/MVS OCWAIT parameter.

When used in IMS Type 2 message syntax, the CMDWAIT and WAIT keywords are identical, and specify the maximum wait time in seconds for message completion. IBM documentation on IMSPLEX usage implies that the delay in gathering results from multiple IMS systems can be considerable, and should be considered normal. The default if omitted is 120, or 2 minutes.

CONTYPE(ANY|EXTCONS|SSCONS|XTRACONS)

(Optional) Specifies the appropriate type of console for the command you are issuing. The default for CONTYPE is the current value of the OCCONTYPE parameter. Values are:

ANY

CA OPS/MVS selects the first available console to issue the command. CA OPS/MVS selects consoles in the order listed here. The search order may change in a future release.

- Subsystem consoles
- XTRACONS consoles
- Extended consoles

EXTCONS

CA OPS/MVS selects the first available extended console. Extended consoles are controlled by the EXTENDEDCONSOLES parameter.

SSCONS

CA OPS/MVS selects the first available subsystem console. Subsystem consoles are controlled by the SUBSYSDEFAULT parameter.

XTRACONS

CA OPS/MVS selects the first available extra extended console. Extra extended consoles are controlled by the EXTRAEXTCONSOLES parameter.

Notes:

- If the CONTYPE keyword is not specified for a cross-system command, then the value of OCCONTYPE on the target system is used.
- The CONTYPE keyword is mutually exclusive with the NAME/CONNAME keyword.

DELAY(*seconds*)

(Optional) Specifies, in seconds, how long ADDRESS OPER pauses before processing the current system command. You can specify any number of seconds between 1 and 300.

This keyword can only be used within AOF rules. If logically possible, avoid using long delay values. Utilize dynamic AOF rules within automated applications that require a long delay or wait to occur prior to the issuing of a command or after the command has been issued. For examples of using dynamic TOD rules (time interval delay or wait) and dynamic MSG rules (event triggered delay wait), see the sample OPS/REXX programs SHUTCA7, SHUTDB2, and SHUTIMS in the hlq.CCLXSAMP CA OPS/MVS dsn.

Default: There is no default.

IMSID(*name*|*|list)

(Optional) If you use the IMS Operation Facility at your site, you can use the ADDRESS OPER command processor to issue IMS commands.

If you are running only one copy of IMS on the system where you issue the ADDRESS OPER command processor, you can omit the IMSID keyword. However, if several copies of IMS (such as a production version and a test version) reside on the system, use the IMSID keyword to specify the IMS control region that should receive the command.

IMS IDs can contain from one to four characters. IBM ships IMS with a default IMS ID of IMSA. If you do not know the IMS ID of the IMS you want CA OPS/MVS to control, ask your systems management department.

When used in IMS Type 2 message syntax, the command is issued to the IMSPLEX manager, and distributed from there. The IMSID is used to establish eligibility of participating IMS systems to receive the command. If the IMSID is omitted, the first IMS system capable of responding will receive the command. If IMSID is specified as a wildcard *, all participating IMS systems will receive the command. IMSID can also be specified as a list, separated by spaces or commas, of specific IMS system names to receive the command. The IMSID wildcard * should not be used unless the IMSPLEX keyword is specified with an explicit name.

Note: A list or a wildcard in the IMSID keyword may only be used in communication with the IMSPLEX manager. The IMSPLEX keyword must be specified in order for the command to be issued to more than one IMS.

IMSPLEX(*name*|*)

The presence of the IMSPLEX keyword implies IMS Type 2 message syntax. It can be specified as a specific IMSPLEX name, in which case the IMSID keyword controls selection of participating IMS systems. IMSPLEX can also be specified as a wildcard *, meaning the IMSID determines the IMSPLEX name. In this case, the IMSID cannot be specified as a wildcard *, but must contain a valid IMS system name. The IMSID, or the first of a list, will be used to locate the name of its associated IMSPLEX.

IMSREPLY

(Optional) Causes the current IMS command to be issued using the IMS WTOR instead of the BMP. This keyword applies only if you use the IOF BMP to issue IMS commands. Other IMS commands issued using OPSCMD or ADDRESS OPER without this keyword will still go to the BMP.

Note: If the keyword is used for non-IMS commands, it is ignored. It will also be ignored when the IOF BMP facility is not in use. In that case, the IMS WTOR is already being used to issue IMS commands; therefore, the keyword is meaningless.

INTERVAL(*centiseconds*)

(Optional) Specifies, in centiseconds, how frequently ADDRESS OPER tests for command response lines to see if the response has ended. The INTERVAL value temporarily overrides the value of the CA OPS/MVS OCINTERVAL parameter.

The default for INTERVAL is the value of the OCINTERVAL parameter, but you can specify any number between 10 and 300, or a value of 0 to bypass interval testing.

For correct usage of this keyword, see the "Utilizing the Address Oper" and the "Examples" section.

LOCALONLY

(Optional) Indicates that only messages from the local system are considered as potential command response candidates.

This keyword should be used in conjunction with the CAPTURE and STOPMSG keywords to prevent output from other systems possibly being returned as command output.

LOG(YES|NO|OFF)

(Optional) Determines whether response lines from the command issued by ADDRESS OPER go into the SYSLOG and/or OPSLOG data sets. Valid options are:

YES

(Default) Command responses and command echo will be logged in both SYSLOG and OPSLOG.

NO

SYSLOG will only contain the command echo, responses will not be logged. OPSLOG will contain both the command echo and command responses.

OFF

Command echo and command responses will not be logged to the SYSLOG. OPSLOG will contain the command responses, command echo will not be logged. If command output is being collected within an OPS/REXX program, the command echo will be omitted.

NOWHERE

Command echo and command responses will not be logged in both SYSLOG and OPSLOG. If command output is being collected within an OPS/REXX program, the command echo will be omitted.

Default: YES

Notes:

- Whether command response lines are logged depends on how the CA OPS/MVS AOFDELETE parameter is set. The LOG keyword affects the logging of only the command response lines returned to the originating console.
- If CA MIC is running at your site, specifying LOG(OFF) does not always have the desired effect. In some cases, CA MIC still sends response lines to the SYSLOG data set.

MAXCMDOUT(*number*)

(Optional) Defines how many command output lines ADDRESS OPER collects before terminating the command response. If the command output contains more lines than the MAXCMDOUT value, ADDRESS OPER stops collecting command output. You can specify any value between 1 and 32767 for MAXCMDOUT.

If the command is issuing IMS Type2 messages and generating REXX variables, the amount of output can become heavy when the VERBOSE option is specified in a busy IMSPLEX environment, and all participating systems respond. This parameter can limit the REXX output volume to a maximum of 32K variables.

Default: The value of the CA OPS/MVS OCMAXMSG parameter. If used to control REXX output from IMS Type 2 messages, the default is 16384.

NAME|CONNAME(*console name*)

(Optional) The NAME keyword and its equivalent keyword, CONNAME, define the name of the console to receive the command that ADDRESS OPER issues. The console name you provide can contain as many as eight characters. The NAME or CONNAME keyword and the ID keyword are mutually exclusive.

You can retrieve command output when the specified console is allocated to CA OPS/MVS. If the console is not allocated or does not exist, ADDRESS OPER issues a return code of 190.

OUTPUT|NOOUTPUT

(Optional) The mutually exclusive OUTPUT and NOOUTPUT keywords determine whether you receive output from the command issued by ADDRESS OPER.

Default: OUTPUT

The NOOUTPUT keyword requests that no command output be returned. ADDRESS OPER executes the specified command but creates no output variables and displays no command output lines. The NOOUTPUT and WAIT keywords are mutually exclusive. For correct usage of this keyword, see Utilizing the ADDRESS OPER Host Environment.

PREFIX(*name*)

(Optional) Defines a prefix name for created variables.

Default prefix name: IMSTYP2

PROPRES

(Optional) This keyword is useful only when ADDRESS OPER is used in CMD rules. In all other environments, this keyword is ignored. When used in CMD rules, this keyword causes the console name and CART (Command and Response Token) associated with the command that triggered the CMD rule to be propagated to the CMD being issued by ADDRESS OPER. Use this keyword when you want the output of the command being issued through ADDRESS OPER to be associated with the original command that triggered the CMD rule.

STOPEND(YES|NO)

(Optional) Determines whether the end line of a multiline WTO message stops ADDRESS OPER from collecting further command output. For correct usage of this keyword, see Utilizing the ADDRESS OPER Host Environment.

YES

(Default) The command response terminates at the first end line of a multiline WTO message or at the time specified through the CMDWAIT keyword or the OCWAIT parameter.

NO

ADDRESS OPER continues collecting command output until one of the following occurs:

- The time interval specified by the CMDWAIT keyword or OCWAIT parameter expires
- No new command output lines are collected in the time interval specified by the INTERVAL keyword

Default: YES

STOPMSG(*msgtextlist*)

(Optional) Specifies a list of one to ten message text segments matching the first characters (starting with column 1) of the messages that terminate the collection of command response lines. When ADDRESS OPER detects any of these text segments, it stops collecting response lines. The message segment or segments you specify need not be directed to the console receiving the command response. Each item in the list of message text segments you specify can contain no more than 124 characters.

All message text segments specified on this keyword will match both uppercase and lowercase text in potential matching messages. For correct usage of this keyword, see Utilizing the ADDRESS OPER Host Environment.

Note: STOPMSG does not work with IMS Type 2 commands.

STOPRESP(msgtextlist)

(Optional) Specifies a list of one to ten message text segments matching the first characters (starting with column 1) of the messages that terminate the collection of command response lines. The message segments you specify must be directed to the console receiving the command response, and each item in the segment list can contain no more than 124 characters. You can use the STOPRESP keyword to terminate the collection of a long command response once the desired response line is found.

All message text segments specified on this keyword will match both uppercase and lowercase text in potential matching messages.

The STOPRESP and STOPMSG keywords are mutually exclusive. For correct usage of this keyword, see Utilizing the ADDRESS OPER Host Environment.

SYSID or SYSTEM(systemids)

(Optional) For more information on the SYSID or SYSTEM keywords, see Specifying an MSF System ID on a POI Command Processor in this chapter.

Note: If more than one system is specified or implied, then the NOOUTPUT keyword is implied.

SYSWAIT(seconds)

(Optional) Usually, when ADDRESS OPER issues a command to a remote system, the total wait time for a response to the command is the sum of two values:

- The time specified with the WAIT or CMDWAIT keyword of ADDRESS OPER, or the OCWAIT parameter default value
- The delay time, if any, that the CA OPS/MVS MSF component imposes, up to the value specified on the DELAY keyword for the ADDRESS OPSCTL MSF DEFINE command issued on the local (source) system to define the remote (target) system, or the CA OPS/MVS MSFSYSWAIT parameter if the DELAY keyword was not specified

The SYSWAIT keyword specifies a value, in seconds, that temporarily overrides the MSF delay time. You can specify a value from 1 to 600.

VERBOSE(YES|NO)

(Optional) Generates individual REXX variables to identify individual elements of the structured response and header data returned when an IMS Type2 command is issued. This option can lead to considerable volume in cases when several IMS systems under IMSPLEX control all respond to a command.

Default: NO

WAIT(seconds)

(Optional) Specifies that the ADDRESS OPER is to unconditionally wait for a period of time to receive output from the current command. In most cases a conditional wait using the CMDWAIT keyword in conjunction with the INTERVAL and various STOPxxxx keywords can be used to effectively collect command responses, thus eliminating the need for a unconditional WAIT. For details of how to successfully collect command response traffic, see Utilizing the ADDRESS OPER Host Environment.

When used in IMS Type 2 message syntax, the WAIT keyword is identical to CMDWAIT, and described there.

XML(YES|NO)

IMS Type 2 responses are all returned as text units within XML framing. The XML keyword can be specified as NO to strip both leading and trailing framing XML characters, or YES to include them with the output.

Default: NO

Usage Notes:

Keep these things in mind when you issue a command through the ADDRESS OPER host command environment:

- When you use ADDRESS OPER to route a command to a remote system through an AOF rule, the NOOUTPUT keyword is implied.
- Although WAITs are allowed in TOD rules, they are not recommended because they may delay the execution of other TOD rules.

ADDRESS OPER Return Codes

ADDRESS OPER commands produce these return codes:

Return Code	Message Number	Message Text
0	OPS1100	COMMAND EXECUTION SUCCESSFUL
0	n/a	NO REPLY COMMAND OUTPUT
4	OPS1101	COMMAND GENERATED NO OUTPUT
8	OPS1102	OUTPUT RETRIEVAL MECHANISM DOWN
12	OPS1103	NO MVS/JES/VM COMMAND ENTERED
16	OPS1104	MVS/JES/VM CMD LENGTH EXCEEDS MAXIMUM

Return Code	Message Number	Message Text
20	OPS1105	MVS/JES/VM CMD OUTPUT BUF OVERFLOW. When RC 20 is returned from a command containing an IMS Type2 command, it indicates a condition where the number of REXX output variables exceeded the allowable limit set by the MAXCMDOUT keyword.
24	OPS1106	VM COMMAND INVALID - MVS NOT RUNNING UNDER VM
28	OPS1107	AUTHORIZATION CHECK FAILED
32	OPS1108	NO JES3 COMMAND OUTPUT ON JES3 LOCALS
36	OPS1109	INVALID KEYWORD COMBINATION ENTERED
40	OPS1110	INVALID LOCAL/REMOTE SYSTEM ID
44	OPS1111	SYSTEM ID IS NOT ACTIVE
48	OPS1112	COMMANDS DISABLED AT THIS TIME
52	n/a	SOME TYPE OF SYSTEM NOT ACTIVE
56	n/a	SOME TYPE OF THING NOT FOUND
60	OPS1115	SOME TYPE OF ERROR OCCURRED
64	OPS1116	PRODUCT MUST BE ACTIVE TO USE COMMAND
88	OPS1122	TSO/E IS NOT INSTALLED
92	OPS1123	COMMAND NOT ISSUED WITH AUTHORIZATION
96	OPS1124	NO CNSLS AVAILABLE FOR CMD OUT RETRIEVAL
100	OPS1125	NO CONSOLES OF THE CORRECT TYPE EXIST
104	OPS1126	MAIN PRODUCT ADDRESS SPACE TERMINATED
108	OPS1127	MULTI-SYSTEM FACILITY NOT INSTALLED
112	OPS1128	MULTI-SYSTEM FACILITY NOT ACTIVE
120	OPS1130	MULTIPLE IMS IDS MISSING IMSPLEX KEYWORD
124	OPS1131	HOST SYSTEM INACTIVE
128	OPS1132	CONSOLE CONTROL BLOCK TAG CHECK
132	OPS1133	MESSAGE QUEUE SEND ERROR
136	OPS1134	MESSAGE QUEUE RECEIVE ERROR
160	OPS1140	CONSOLE CONTROL BLOCK ENABLE FAILED
164	OPS1141	CONSOLE BLOCK RELEASE LOGIC ERROR

Return Code	Message Number	Message Text
172	OPS1143	COMMAND FUNCTION AREA GET/FREE FAILED
176	OPS1144	COMMAND OUTPUT BUFFER GET/FREE FAILED
180	OPS1145	COMMAND OUTPUT QUEUE GET/FREE FAILED
184	OPS1146	ABEND FAILURE IN USER EXIT
186	OPS1146	ABEND FAILURE IN FUNCTION ROUTINE
188	OPS1147	MGCR (SVC 34) FAILURE
189	OPS1147	MGCRC (SVC 34) FAILURE
190	OPS1153	CONVCON FAILURE
200	OPS1150	COMMAND SCAN AREA ALLOCATION FAILED
204	OPS1151	COMMAND SCAN AREA RELEASE FAILED
212	OPS1153	COMMAND LENGTH IS ZERO
216	OPS1154	INVALID COMMAND BUFFER FORMAT
220	OPS1155	MASTER CONTROL BLOCK ERROR
224	OPS1156	IKJSCAN ERROR
240	OPS1160	CLIST VARIABLE ACCESS ERROR
244	OPS1161	WORD TOKENIZATION ERROR
248	OPS1162	SYSOUTTRAP CONVERSION ERROR
252	OPS1164	COMMAND BUFFER PARSE ERROR
253	OPS1183	KEYWORD NOT SUPPORTED IN CURRENT MVS LEVEL
254	OPS1184	INVALID CONSOLE ID SPECIFIED
255	OPS1153	NAME AND ID ARE MUTUALLY EXCLUSIVE
256	OPS4363	KEYWORD NOT VALID IN THIS ENVIRONMENT

IMS Type 2 Message Return and Reason Codes

The IMSPLEX manager API interface can generate a variety of unique return and reason codes, documented in IBM manual SC18-9967, *IMS V10 System Programming API Ref.* Errors associated with API registration can be found under the section CSLSCREG Return and Reason Codes and those with API dialog under CSLOMI Return and Reason Codes.

The ADDRESS OPER implementation utilizes only two of the API facilities. In the case of an error, OPS message 1137 is issued for a registration failure, and message 1138 will accompany a request/response error.

Both messages contain return and response codes, which can be used to look up the specific error condition in the above documentation, in the following format:

return/reason codes = X'aaaaaa'/X'bbbbbb'

Please note that a registration error will not allow the command to continue. A dialog error, however, can occur after the IMSPLEX manager starts data accumulation and might have incomplete results to return, thus producing a combination of both an error message and valid, but possibly partial, results.

The following error is an indication that the appropriate IMS product load library is not allocated:

X'01000010'/X'00004004' CSLSRG00 could not be loaded.

ADDRESS OPSCTL Host Environment

Using the ADDRESS OPSCTL host environment, your rules and OPS/REXX procedures can control the CA OPS/MVS COF, ECF, OSF, OPSLOG, and MSF components. This host environment is used by OPSVIEW options 4.2, 4.3, 4.4, 4.13.

The ADDRESS OPSCTL environment returns command output to the OPS/REXX external data queue, with blanks separating the individual output fields.

ADDRESS OPSCTL Commands

The ADDRESS OPSCTL host environment controls the ECF, MSF ONLY, MSF, and COF, and OSF components.

- To control the ECF component, the ADDRESS OPSCTL host environment uses the following command:

ECF LIST

Returns information about each ECF user logged onto a console.

- To control the MSF component, the ADDRESS OPSCTL host environment uses the following commands:

MSF DEFAULT

Enables you to specify a default system name and system wait value for the currently executing REXX program or rule. Only those OPS/REXX functions that support cross system operations and do not allow the explicit setting of a system name and system wait value (for example: OPSVALUE), are applicable.

MSF START

Tells CA OPS/MVS to initialize the MSF on the local MSF system, to open its VTAM access control block, and to begin communicating with copies of CA OPS/MVS running on remote systems.

MSF STOP

Instructs the local copy of CA OPS/MVS to end its sessions with remote CA OPS/MVS copies and to close its VTAM access control block.

- To control the MSF and COF components, the ADDRESS OPSCTL host environment uses the following commands:

MSF|COF ACTIVATE

This command does one of the following:

- Causes the MSF to try to activate a VTAM session with the MSF running on another system
- Causes an association between a transient data queue and the COF

MSF|COF DEACTIVATE

Ends the session between local and remote copies of the MSF, or ends the association between a transient data queue and the COF.

MSF|COF DEFINE

Defines to the MSF the systems it can communicate with, or defines to the COF a list of CICS transient data queue names to be selected for AOF processing.

MSF|COF DELETE

Deletes either specific or all MSF or COF resources that you have defined.

MSF|COF LIST

Displays all MSF or COF resources currently defined and their status.

- To control the OSF component, the ADDRESS OPCTL host environment uses the following commands:

OSF EXECSTATS

Returns performance information about the OSF server component. This information is useful for tuning OSF-related parameters to meet the performance objectives of your installation.

OSF LIST

Returns information about active servers to the external data queue.

OSF QUEUES

Returns status and historical information about the server execution queue to the external data queue.

OSF RESETQ

Causes all pending commands waiting on the OSF execute queue to be immediately discarded. As a result of this command, all existing OSF execute queue statistics are reset to 0 (zero).

OSF STOP *nnnn*

Stops the server that has the address space ID *nnnn*.

- To control the OPSLOG component, the ADDRESS OPCTL host environment uses the following commands:

DEFINE

Defines a new OPSLOG for future use.

ACTIVATE

Makes a previously defined OPSLOG definition active for use. In the cases where the OPSLOG is DIV-backed, the data set must exist at the time it is activated.

SETLIVE

Makes a previously active, read/write, OPSLOG the live log. This means that new events processed by the CA OPS/MVS subsystem are added to this OPSLOG. There can only be one live OPSLOG. Other than at product startup where there is no prior live log, the prior live log becomes an active log again.

RESET

Empties an active, read/write, OPSLOG (it cannot be the live log), which makes it available for use as an empty log.

LIST

Lists all of the defined OPSLOGs.

DEACTIVATE

Changes the status of an OPSLOG back to defined.

DELETE

Deletes an OPSLOG definition.

ADDRESS OPSCTL Return Codes

All ADDRESS OPSCTL host commands set a return code. Each command returns a code of 0 if it completes successfully. If a command does not execute successfully, you receive a non-zero return code and CA OPS/MVS writes an error message into the OPS/REXX external data queue.

The ADDRESS OPSCTL MSF LIST host command returns a code of 4 if no MSF system is defined, or a code of 8 when CA OPS/MVS cannot perform the list request. These LIST commands do not generate error messages.

In the following sections, return codes specific to each particular type of OPSCTL command (OSF, COF, ECF, and MSF) are listed. In addition, the following return codes apply to all OPSCTL commands:

0

The command completed successfully.

12

The command was too long (>32768) or too short.

13

The command length exceeded an internal limit.

16

The command contained an unknown request (a request other than OSF, COF, ECF, or MSF).

20

The CA OPS/MVS subsystem is not active.

28

An abend occurred in the user security exit.

32

The authorization exit or a security rule rejected the OPSCTL command.

36

A required verb is missing.

44

A syntax error was detected during command string parsing.

ADDRESS OPSCTL Commands for the COF

ADDRESS OPSCTL COF commands enable you to tailor lists of CICS transient data queue names that the CICS XTDOU global exit will intercept and pass to the AOF for rule processing.

Each copy of CA OPS/MVS that is active on your system may use a different list of queue names for AOF processing, or may ignore transient data queue message traffic altogether.

Optionally, the AOF may suppress or reword a transient data queue message.

Wildcards in OPSCTL COF Commands

Follow these guidelines when using wildcards in an OPSCTL COF command:

- For certain keywords that you specify on the DELETE, ACTIVATE, DEACTIVATE, and LIST commands, you can use the asterisk (*) character as a wildcard suffix.

For example, this keyword value matches CICSPROD, CICSTEST, CICSSYS2, and any other job name beginning with the characters CICS:

`JOBNAME(CICS*)`

Where wildcard suffixes are valid, the ability to use them is noted in the keyword descriptions.

- For the JOBNAME and STEPNAME keywords on the DELETE command, you can specify the asterisk (*) character by itself to match all job names or step names.

Generate the DEFAULT List

The sections that follow describe the commands you use to tailor and manipulate lists of transient data queue destination names. In addition to the lists you generate by using the COF DEFINE command, the CICS XTDOU exit generates a DEFAULT list.

Once the XTDOU exit is installed in a CICS region, the next transient data record that is written causes the exit to create a transient data queue name list for each active CA OPS/MVS subsystem that has its INITAOF and CICS AOF parameters set to YES.

If no predefined name list with a matching job name is found, then the DEFAULT list is used. `JOBNAME(DEFAULT)` is reserved for this purpose.

You can use any of the ADDRESS OPSCTL COF commands to manipulate the DEFAULT list, but only if you explicitly identify it by specifying DEFAULT as the value for the JOBNAME keyword for the command.

Note: The DEFAULT list is never selected if you use a wildcard suffix when you specify JOBNAME.

After a CICS region has built its destination name list by copying the DEFAULT list, the list is identified by the CICS job name.

If the DEFAULT list is not defined in advance, CICS regions build empty destination name lists, and no messages are sent to the AOF. You can use the COF ACTIVATE command to add destination names to the empty list. Alternatively, use the COF DEFINE command to build the DEFAULT list.

If you use the COF DELETE command to remove the current list that is associated with a particular CICS region, the next transient data write request from that region builds a new list from the now defined DEFAULT list.

Destination IDs

When you issue the COF DEFINE, COF ACTIVATE, and COF DEACTIVATE commands, you may specify a list of destination IDs for the DESTIDS keyword. Destination IDs are the names of CICS transient data queue destinations, as defined in the CICS destination control table, or DCT. If no DESTIDS are specified on the DEFINE command, then CICS regions build empty destination name lists, and no messages are sent to the AOF from that CICS region. You can use the COF ACTIVATE command to add destination names to the empty list at a later time.

You can use the COF ACTIVATE command to add destination names to the empty list at a later time.

You can use the CICS CEMT command to display the names of CICS destination queues.

For your convenience, you can use the following destination name in any OPSCTL COF command to see the DEFAULT destination ID list that is described in the previous section:

%%%%%%%%

For example, consider the following:

```
ADDRESS OPSCTL "COF DEFINE JOBNAME(DEFAULT)",  
               "DESTIDS(CSMT,CSSL,CADL,CSDL,CSKL)"
```

```
ADDRESS OPSCTL "COF DEFINE JOBNAME(CICSPROD)",  
               "DESTIDS(%%%%%%%%,CSPL,CSNE)"
```

Indirect Destination IDs

Indirect transient dataset queue names can be added to the explicit DESTIDS list just like any other intra-partition queue name. Because of the potentially large number and volatility of indirect queue names, the INDIRECT(YES) keyword is available. This keyword causes all indirect transient data queue names whose target queue names are explicitly defined in the DESTIDS keyword to be treated as implicitly defined. This keyword eliminates the need to constantly monitor the indirect queue name definitions in order to update the DESTIDS list.

COF ACTIVATE—Add Transient Data Queue Names

This command adds the specified transient data queue names to any destination list that matches the selection criteria you specify.

This command has the following format:

```
ADDRESS OPSCTL "COF ACTIVATE keywords"  
{JOBNAME(jobname)}  
[STEPNAME(stepname|taskid)]  
[STATUS(ACTIVE|INACTIVE)]  
[DESTIDS(destidlist)]  
[INDIRECT(YES|NO)]  
[OUTPUT|NOOUTPUT]  
[SYSTEM(sysname)]  
[SYSWAIT(seconds)]
```

JOBNAME(jobname)

Defines the one- to eight-character job name of the CICS region.

Position: This required keyword and must be first in the command syntax.

When specifying the job name, you may use the wildcard suffix.

STEPNAME(stepname|taskid)

(Optional) Defines the one- to eight-character step name of the CICS region (or task ID for a started task).

Position: It must follow the required JOBNAME keyword and precede the optional DESTIDS keyword.

This keyword intended for sites that use a shared CICS JCL PROC to start CICS regions. Since all CICS regions have the same job name at such sites, use the STEPNAME keyword to differentiate the various regions.

You must include the STEPNAME keyword in your COF ACTIVATE command to activate any transient data queue names whose definitions included the STEPNAME keyword.

When specifying the step name, you may use the wildcard suffix.

STATUS(ACTIVE|INACTIVE)

(Optional) Indicates the action of the ACTIVATE command is either ACTIVE or INACTIVE lists.

Position: It must follow the required JOBNAME keyword and precede the optional DESTIDS keyword.

ACTIVE

Specifies the lists in use by a CICS region.

INACTIVE

Specifies the lists not currently in use by a CICS region.

DESTIDS(*destidlist*)

(Optional) Defines the transient data queue destinations that you want to add, as defined in the CICS DCT (destination control table).

Position: This optional keyword must follow the optional keywords STEPNAME and STATUS (if they are used).

You can specify as many as 100 destination names in a list; each can contain from one to four characters.

The CA OPS/MVS product maintains the names in your lists in sorted order and eliminates any duplicate names. If the limit of 100 DESTIDS is exceeded as a result of an ACTIVATE command, then the list is sorted first and the last entries in the sorted sequence are discarded.

INDIRECT(YES|NO)

(Optional) Any indirect transient queue names that point to a defined transient queue name in the DESTIDS list can be processed by the COF interface. This indirect transient queue message is counted as messages issued to the corresponding DESTIDS defined destination name.

Position: This optional keyword must follow the optional keywords STEPNAME and STATUS (if they are used).

YES

Specifies that undefined indirect transient data queue names are processed by the COF interface.

NO

(Default) Specifies that only transient data queue names in the DESTIDS list are processed by the COF interface.

OUTPUT, NOOUTPUT, SYSTEM(*sysname*), and SYSWAIT(*seconds*)

(Optional) You may specify the keywords that are common to all ADDRESS OPSCTL COF commands.

Position: Place these optional keywords at the end of the command.

More information:

[Wildcards in OPCTL COF Commands](#) (see page 250)

[Keywords Common to All ADDRESS OPCTL COF Commands](#) (see page 263)

COF DEACTIVATE—Delete Transient Data Queue Names

This command deletes the specified transient data queue names from any destination list that matches the selection criteria you specify.

This command has the following format:

```
ADDRESS OPCTL "COF DEACTIVATE keywords"  
{JOBNAME(jobname)}  
[STEPNAME(stepname|taskid)]  
[STATUS(ACTIVE|INACTIVE)]  
[DESTIDS(destidlist)]  
[INDIRECT(YES|NO)]  
[OUTPUT|NOOUTPUT]  
[SYSTEM(sysname)]  
[SYSWAIT(seconds)]
```

JOBNAME(jobname)

Defines the one- to eight-character job name of the CICS region.

Position: This required keyword must be first in the command syntax.

When specifying the job name, you can use the wildcard suffix.

STEPNAME(stepname|taskid)

(Optional) Defines the one- to eight-character step name of the CICS region (or task ID for a started task).

Position: STEPNAME must follow the JOBNAME keyword and STEPNAME must precede the optional DESTIDS keyword.

This keyword is intended for sites that use a shared CICS JCL PROC to start CICS regions. Since all CICS regions have the same job name at such sites, use the STEPNAME keyword to differentiate the various regions.

The STEPNAME keyword must be included in your COF DEACTIVATE command to deactivate any transient data queue names whose definitions included the STEPNAME keyword.

When specifying the step name, you can use the wildcard suffix.

STATUS(ACTIVE|INACTIVE)

(Optional) Specifies the action of the DEACTIVATE command as either ACTIVE or INACTIVE lists.

Position: STATUS must follow the JOBNAME keyword and STATUS must precede the optional DESTIDS keyword.

ACTIVE

Specifies the lists in use by a CICS region.

INACTIVE

Specifies the lists not currently in use by a CICS region.

DESTIDS(*destidlist*)

(Optional) Defines the one- to four-character transient data queue destinations that you want to delete.

Position: This optional keyword must follow the optional keywords STEPNAME and STATUS (if they are used).

When specifying destination names to be deleted, you can use the wildcard suffix.

INDIRECT(YES|NO)

(Optional) Specifies whether any indirect transient data queue names that ultimately point to a defined transient data queue name in the DESTIDS list can be processed by the COF interface. This keyword is counted as messages issued to the corresponding defined destination name.

Position: This optional keyword must follow the optional keywords STEPNAME and STATUS (if they are used).

YES

Specifies that undefined indirect transient data queue names is processed by the COF interface.

NO

(Default) Specifies that only transient data queue names in the DESTIDS list are processed by the COF interface.

OUTPUT, NOOUTPUT, SYSTEM(*sysname*), and SYSWAIT(*seconds*)

(Optional) You can specify the common keywords described in Keywords Common to All ADDRESS OPSCTL COF Commands in this chapter.

Position: Place these optional keywords at the end of the command.

More information:

[Wildcards in OPSCTL COF Commands](#) (see page 250)

COF DEFINE—Create Transient Data Queue Name List

The COF DEFINE command creates a list of CICS transient data queue names. For the CICS region identified by the JOBNAME keyword, STEPNAME keyword, or both the CICS XTDOU exit selects the destinations on this list and passes them to the AOF for rule processing. If a current list exists that was not defined, but rather built, from the default list by the first message intercepted by the XTDOU exit, the old connection list will be deleted by the new defined list. Duplicate DEFINES for the same CICS JOBNAME/STEPNAME combination are not allowed. The old definition must be deleted first.

This command has the following format:

```
ADDRESS OPSCTL "COF DEFINE keywords"  
{JOBNAME(jobname)}  
[STEPNAME(stepname|taskid)]  
[DESTIDS(destidlist)]  
[INDIRECT(YES|NO)]  
[OUTPUT|NOOUTPUT]  
[SYSTEM(sysname)]  
[SYSWAIT(seconds)]
```

JOBNAME(jobname)

Defines the one- to eight-character job name of the CICS region.

Position: This required keyword must be first in the command syntax.

STEPNAME(stepname|taskid)

(Optional) Defines the one- to eight-character step name of the CICS region (or task ID for a started task).

Position: It must follow the required JOBNAME keyword and precede the optional DESTIDS keyword.

This optional keyword is intended for sites that use a shared CICS JCL PROC to start CICS regions. Since all CICS regions have the same job name at such sites, use the STEPNAME keyword to differentiate the various regions.

If you specify the STEPNAME keyword in the definition of a transient data queue name list, you must also specify it in any other COF commands (ACTIVATE, DEACTIVATE, DELETE, LIST) to select the transient data queues on that list.

For example, suppose that you issue this command to define a CICS connection:

```
ADDRESS OPSCTL "COF DEFINE JOBNAME(CICSTOR),  
"STEPNAME(TOR) DESTIDS(CSMT,CSSL)"
```

The following COF LIST command will not select the CICS connection defined above, because the COF LIST command does not include the STEPNAME keyword:

```
ADDRESS OPSCTL "COF LIST JOBNAME(*)"
```

But this COF LIST command will select the CICS connection:

```
ADDRESS OPSCTL "COF LIST JOBNAME(*) STEPNAME(*)".
```

DESTIDS(*destidlist*)

(Optional) Defines a list of transient data queue destinations, as defined in the CICS DCT (destination control table).

Position: This optional keyword must follow the optional STEPNAME keyword (if used).

You can specify as many as 100 destination names in a list; each name can contain from one to four characters.

CA OPS/MVS maintains the names in your lists in sorted order and eliminates any duplicate names.

INDIRECT(YES|NO)

(Optional) Specifies whether any indirect transient data queue names that ultimately point to a defined transient data queue name in the DESTIDS list can be processed by the COF interface. This keyword is counted as messages issued to the corresponding defined destination name.

Position: This optional keyword must follow the optional keywords STEPNAME and STATUS (if they are used).

YES

Specifies that undefined indirect transient data queue names is processed by the COF interface.

NO

(Default) Specifies that only transient data queue names in the DESTIDS list are processed by the COF interface.

OUTPUT, NOOUTPUT, SYSTEM(*sysname*), and SYSWAIT(*seconds*)

(Optional) You may specify the common keywords described in Keywords Common to All ADDRESS OPSCTL COF Commands in this chapter.

Position: Place these optional keywords at the end of the command.

COF DELETE—Delete Transient Data Queue Name List

This command deletes any defined or default transient data queue name lists that match the selection criteria specified on the command.

This command has the following format:

```
ADDRESS OPSCTL "COF DELETE keywords"  
{JOBNAME(jobname)}  
[STEPNAME(stepname|taskid)]  
[STATUS(ACTIVE|INACTIVE)]  
[OUTPUT|NOOUTPUT]  
[SYSTEM(sysname)]  
[SYSWAIT(seconds)]
```

JOBNAME(*jobname*)

Defines the one- to eight-character job name of the CICS region. When specifying the job name, you may use the wildcard suffix, or specify * by itself to match all job names.

STEPNAME(*stepname*|*taskid*)

(Optional) Defines the one- to eight-character step name of the CICS region (or task ID for a started task).

This keyword is intended for sites that use a shared CICS JCL PROC to start CICS regions. Since all CICS regions have the same job name at such sites, use the STEPNAME keyword to differentiate the various regions.

You must include the STEPNAME keyword in your COF DELETE command to delete any transient data queue names whose definitions included the STEPNAME keyword.

When specifying the step name, you may use the wildcard suffix, or specify * by itself to match all step names.

STATUS(ACTIVE|INACTIVE)

(Optional) Limits the action of the DELETE command to either ACTIVE or INACTIVE lists:

ACTIVE

Lists in use by a CICS region

INACTIVE

Lists not currently in use by a CICS region

OUTPUT, NOOUTPUT, SYSTEM(*sysname*), and SYSWAIT(*seconds*)

(Optional) You may specify the keywords common to all ADDRESS OPSCTL COF commands.

More information:

[Wildcards in OPSCTL COF Commands](#) (see page 250)

[Keywords Common to All ADDRESS OPSCTL COF Commands](#) (see page 263)

COF LIST—Display Transient Data Destination List

The COF LIST command displays the contents of any transient data destination list that matches the selection criteria you specify.

This command has the following format:

```
ADDRESS OPSCTL "COF LIST keywords"  
{JOBNAME(jobname,[SUMMARY])}  
[STEPNAME(stepname|taskid)]  
[STATUS(ACTIVE|INACTIVE)]  
[OUTPUT|NOOUTPUT]  
[SYSTEM(sysname)]  
[SYSWAIT(seconds)]
```

The optional keyword SUMMARY must follow the jobname value, and a comma must separate it from the jobname value as follows:

```
JOBNAME(TESTJOB,SUMMARY)
```

JOBNAME

Defines the one- to eight-character job name of the CICS region. If you include the SUMMARY positional parameter, the COF LIST command returns only one record per matching CICS connection, rather than one record for each destination name in the destination name list.

When specifying the job name, you may use the wildcard suffix.

STEPNAME

(Optional) Defines the one- to eight-character step name of the CICS region (or task ID for a started task).

This keyword is intended for sites that use a shared CICS JCL PROC to start CICS regions. Since all CICS regions have the same job name at such sites, use the STEPNAME keyword to differentiate the various regions.

You must include the STEPNAME keyword in your COF LIST command to list any transient data queue names whose definitions included the STEPNAME keyword.

When specifying the step name, you may use the wildcard suffix.

STATUS

(Optional) Limits the action of the LIST command to either ACTIVE or INACTIVE lists:

ACTIVE

Lists in use by a CICS region

INACTIVE

Lists not currently in use by a CICS region

OUTPUT, NOOUTPUT, SYSTEM, and SYSWAIT

(Optional) You may specify the keywords common to all ADDRESS OPSTCL COF commands.

More information:

[Wildcards in OPSTCL COF Commands](#) (see page 250)

[Keywords Common to All ADDRESS OPSTCL COF Commands](#) (see page 263)

COF List Command Output

The COF LIST command returns the following system information to the OPS/REXX external data queue as data fields separated by blanks. If you specify `JOBNAME(jobname,SUMMARY)` on your COF LIST command, only one record per matching CICS connection is returned. If you omit the SUMMARY parameter, one record is produced for each destination name contained in each selected list.

Word Number: 1

Length: 8

Contains the CICS job name.

Word Number: 2

Length: 8

Contains the CICS step name or for a started task, the task ID.

Word Number: 3

Length: 8

Contains the CICS generic VTAM application ID from the SIT.

Word Number: 4

Length: 4

Contains the CICS system ID from the SIT.

Word Number: 5

Length: 4

Contains the address space ID of the CICS region, in printable hexadecimal format.

Word Number: 6

Length: 1

Contains a flag indicating whether a match on the STEPNAME is required:

Y

OPSCTL COF DEFINE command generated this list which includes the STEPNAME keyword. In order to use this list, the step name of the CICS region must match.

N

The step name is informational only; no match is required.

Word Number: 7

Length: 8

Contains the status of the list:

- ACTIVE-A CICS region is using the list.
- INACTIVE-The list is not in use.

Word Number: 8

Length: 8

Contains the origin of the list:

- DEFINED-An OPSCTL COF DEFINE command generated the list.
- DEFAULT-A CICS region built this destination name list by copying the default list.

Word Number: 9

Length: 8

Contains the address of the XTDOU global exit area, in printable hexadecimal format for debugging

Word Number: 10

Length: 3

Contains the total number of transient data queue names in the list

Word Number: 11

Length: 3

Contains the index number of this transient data queue name in the list

Word Number: 12

Length: 4

SUMMARY omitted

This word contains the name of the transient data queue destination, or N/A if the transient data queue list is empty.

SUMMARY specified

This word contains the INDIRECT option value YES or NO.

Word Number: 13

Length: 7

SUMMARY omitted

This word contains a count of the messages sent to the AOF for this transient data queue name.

SUMMARY specified

This word contains the sum of all message counts for all destinations in the list.

OPSCTL COF Return Codes

OPSCTL COF commands produce the following return codes:

8

The COF found no matches to the selection criteria that were specified.

40

There is no command output from the remote command.

44

A command syntax error occurred.

48

A system service error occurred.

72

An invalid system name was specified.

76

No last message was received from the remote command.

92

The COF DEFINE command failed.

96

The COF DELETE command failed.

100

The COF ACTIVATE or COF DEACTIVATE command failed.

101

An ACTIVATE command added more DESTIDS than are allowed (over 100). Only the first 100 in alphabetic sequence are retained.

104

The COF interface is not initialized.

More information:

[ADDRESS OPSCTL Return Codes](#) (see page 249)

Keywords Common to All ADDRESS OPSCTL COF Commands

All ADDRESS OPSCTL COF host commands share the following keywords, which support cross-system execution:

OUTPUT or NOOUTPUT

Determines whether the command returns output to the external data queue. OUTPUT and NOOUTPUT are mutually exclusive. Specify OUTPUT to have output returned; otherwise, specify NOOUTPUT.

SYSTEM(*sysname*)

Defines the system name for the remote copy of CA OPS/MVS. The value of *sysname* can contain from one to eight characters.

SYSWAIT(*seconds*)

Defines the maximum time that the command waits for output from the remote system. You can specify a value from 1 to 300 seconds; the default is the value of the MSFSYSWAIT parameter.

Omit the SYSWAIT keyword if you specify NOOUTPUT, because there is no reason to wait unless you want to collect output.

Commands issued to the local system ignore the SYSWAIT keyword.

ADDRESS OPSCTL Commands for the ECF

There is only one ADDRESS OPSCTL command for the ECF. For a description of the ECF LIST command, see The ECF LIST Command in this chapter.

ECF LIST—List Console Users

The ECF LIST command lists information about console users logged onto the ECF.

This command has the following format:

ADDRESS OPSCTL "ECF LIST" (no keywords)

ECFLIST Command Output

The ECFLIST command returns the following output data, separated by blanks to the external data queue. OPSVIEW option 4.4 also displays this information.

Word Number: 1

Length: 4

Contains the ID of the address space (in hexadecimal format).

Word Number: 2

Length: 8

Contains the started task name for the address space.

Word Number: 3

Length: 8

Contains the four-character CA OPS/MVS subsystem ID, followed by the ID of the z/OS console at which the ECF logged on (for example, OPSS/0900).

Word Number: 4

Length: 5

Contains the number of transactions the address space processed since it was started

Word Number: 5

Length: 8

Contains the length of time that has passed since the address space began processing the current transaction

Notes: The format of this word depends upon its content:

- If the value is less than 1000 seconds, the format is sss.tttS, where sss is seconds and ttt is tenths of a second, and S is appended.
- If the value is less than 100 hours but greater than or equal to 1000 seconds, the format is hh.mm.ss, where hh is hours, mm is minutes, and ss is seconds.
- If the value is greater than or equal to 100 hours, the format is hhhhh.mm, where hhhhh is hours and mm is minutes.

Word Number: 6

Length: 8

Contains the amount of CPU time the address space has used so far to process the current transaction.

The format of word 6 depends upon its content. For details, see the note in the description of word 5.

Word Number: 7

Length: 7

Contains the number of lines of output produced so far by the processing of the current transaction.

Word Number: 8

Length: 8

Contains the name of the z/OS console at which the ECF user logged on (for example, XE03921).

If no consoles are logged onto the ECF, the ECF LIST command produces no output.

OPSCTL ECF LIST Return Codes

The OPSCTL ECF LIST command produces the following return codes:

24

The subsystem version is incompatible.

60

The ECF request contained an invalid verb.

64

The ECF request contained a syntax error.

68

A CA OPS/MVS service routine failure (SENDMSG) occurred.

More information:

[ADDRESS OPSCTL Return Codes](#) (see page 249)

ADDRESS OPSCTL Commands for the MSF

The next several sections describe ADDRESS OPSCTL commands for the MSF.

To specify the subsystem for ADDRESS OPSCTL MSF commands, use an ADDRESS AOF command.

MSF ACTIVATE—Activate a Session

This command causes the MSF to try to activate a session with the named system or with all defined remote systems.

This command has the following format:

```
ADDRESS OPSCTL "MSF ACTIVATE keywords"  
{msfid(sysname|ALL)}  
[OUTPUT|NOOUTPUT]  
[SYSTEM(sysid)]  
[SYSWAIT(seconds)]  
[WAIT(seconds)]
```

MSFID(*sysname* or ALL)

A required keyword specifying the name of the system to be activated. System names can contain one to eight characters. The first character must be @, #, or \$ or an alphabetic character, but the remaining characters can be any alphabetic or national character. If you specify ALL in place of *sysname*, the MSF tries to activate sessions with all defined remote systems that do not currently have active sessions.

OUTPUT, NOOUTPUT, SYSTEM, SYSWAIT, and WAIT

(Optional) You may specify the OUTPUT, NOOUTPUT, SYSTEM, SYSWAIT, and WAIT keywords.

More information:

[Keywords Common to ADDRESS OPSCTL MSF Commands](#) (see page 281)

MSF DEACTIVATE—End a Session

This command causes the MSF to end the session between the local system and the named remote system or all sessions between the local system and all defined remote systems.

This command has the following format:

```
ADDRESS OPCTL "MSF DEACTIVATE keywords"  
{msfid(sysname|ALL)}  
[OUTPUT|NOOUTPUT]  
[SYSTEM(sysid)]  
[SYSWAIT(seconds)]  
[WAIT(seconds)]
```

MSFID(*sysname* or ALL)

A required keyword specifying the name of the system to be deactivated. System names can contain one to eight characters. The first character must be @, #, or \$ or an alphabetic character, but the remaining characters can be any alphabetic or national character. If you specify ALL in place of *sysname*, the MSF tries to stop sessions with all defined remote systems that currently have active sessions.

OUTPUT, NOOUTPUT, SYSTEM, SYSWAIT, and WAIT

(Optional) You may specify the OUTPUT, NOOUTPUT, SYSTEM, SYSWAIT, and WAIT keywords.

More information:

[Keywords Common to ADDRESS OPCTL MSF Commands](#) (see page 281)

MSF DEFAULT—Specify System Name and Wait Defaults

This command enables you to specify a default system name and system wait value for the currently executing REXX program or rule.

This command has the following format:

```
ADDRESS OPSCTL "MSF DEFAULT keywords"  
[SYSTEM(sysname)]  
[SYSWAIT(seconds)]
```

SYSTEM(*sysname*)

(Optional) Defines the name of the remote or local MSF system that is to receive the command. The value of *sysname* can contain from one to eight characters.

If you issue the MSF DEFAULT SYSTEM command, the value you specify for the SYSTEM keyword overrides any SYSTEM values you may have previously specified for the REXX program or rule.

SYSWAIT(*seconds*)

(Optional) Defines the maximum time that the command waits for output from the remote system. You may specify a value between 1 and 300 seconds.

If you issue the MSF DEFAULT SYSWAIT command, the value you specify for the SYSWAIT keyword overrides any SYSWAIT values you may have previously specified for the REXX program or rule.

Note: ADDRESS OPER and ADDRESS SQL commands do not currently support the MSF DEFAULT SYSTEM(ALL) command. If you want to send ADDRESS OPER or ADDRESS SQL traffic to all systems, then you must specify the SYSTEM(ALL) keyword on the ADDRESS OPER and ADDRESS SQL commands.

MSF DEFINE—Define VTAM LU to LU Sessions

This command defines the VTAM LU to LU sessions that the MSF establishes and maintains with copies of CA OPS/MVS running on other systems. Typically, the MSF uses these sessions to send and receive commands and command responses.

This command has the following format:

```
ADDRESS OPSCTL "MSF DEFINE keywords"
{msfid(sysname)}
{APPLID(vtamname)}
[PASSWORD(password)]
[NORETRY | RETRY(seconds retries)]
[ALIAS(alias1,...alias8)]
[DELAY(seconds)]
[APPC | CCI | AP]
[SECURE | NOSECURE]
[LOGMODE(logmode)]
[OUTPUT | NOOUTPUT]
[SYSTEM(sysid)]
[SYSWAIT(seconds)]
[WAIT(seconds)]
```

MSFID(*sysname*)

Defines the system name. System names can contain one to eight characters. The first character must be @, #, or \$ or an alphabetic character, but the remaining characters can be any alphabetic, numeric, or national characters. The words ALL and EXT are reserved names and cannot be used as a system name or an alias.

APPLID(*vtamname*)

Defines one of the following:

- If you are defining a remote system and will specify the CCI keyword to indicate that the MSF should use the CAICCI communications services to communicate with the remote system, use the APPLID keyword to specify the CAICCI system ID of the remote system. The MSF checks to make sure that the value you specify is valid.
- If you are defining a remote system and will specify the AP keyword to indicate that MSF should use the CAICCI communication services to communicate with the remote system, use the APPLID keyword to specify the CCI host name of the remote CA Automation Point system.

For the value of the APPLID keyword to be valid, ensure that it has the following characteristics:

- When you specify the APPC keyword, the value of APPLID must be a valid VTAM application name.
- When you specify the CCI keyword, the value of APPLID must be the CAICCI system ID.
- When you specify the AP keyword, the value of the APPLID must be a CCI host name. If the CCI host name is longer than eight characters, the value of the APPLID can be either the first eight characters of the CCI host name or the CCI alias name, if one is specified.

The value that you specify for the APPLID keyword must match the value specified on the APPL statement you placed in your SYS1.VTAMLST data set for the MSF when you installed CA OPS/MVS. For information about installing the MSF, see the Installation Guide.

PASSWORD(*password*)

(Optional) Defines the password the MSF uses when opening its VTAM ACB. It is required only when you define the local MSF system and when the APPL specification in SYS1.VTAMLST requires a password. This password must match the password on the APPL statement.

NORETRY

(Optional) Instructs the MSF not to reestablish the session. Use this optional keyword only when defining a remote system.

RETRY(*seconds retries*)

(Optional) Tells the MSF on the local system to try to reestablish the session and, if the attempt fails, to try again. The seconds value specifies how many seconds the MSF waits between attempts to connect to the system being defined (if you are defining a remote system) or between attempts to start the local MSF. The value for seconds must be an integer between 30 and 86400, and the default is 30 seconds.

The retries value specifies the maximum number of times the MSF tries to start the local MSF or establish a remote session. This value must be an integer between 0 and 65535. Zero, which causes the MSF to make no retry attempt, is the default.

ALIAS(*alias,...alias8*)

(Optional) Defines one to eight alternate names that the MSF can use for this system. Use the same syntax rules for aliases as for system names. You can use an alias only when you issue cross-system commands using the OPSRMT command or the SYSID keyword of the OPSCMD command.

DELAY(seconds)

(Optional) Defines the network delay time between the local MSF system and the remote system being defined. The seconds value must be an integer between 0 and 60, and 1 second is the default.

Use the DELAY keyword together with the WAIT value for the OPSCMD command or the default wait value for the OCWAIT command, as follows:

- When you specify the WAIT value for OPSCMD, that value plus the value of DELAY seconds becomes the local wait time and the original wait time becomes the remote wait time.
- When you specify no wait value for OPSCMD, the sum of the OCWAIT value and the DELAY seconds value becomes the local wait time and the remote wait time is the OCWAIT time on the remote system.
- If you specify no WAIT value on the OPSCMD command processor, we strongly recommend that the OCWAIT values on the local and remote systems be identical.

APPC

(Optional) Tells the MSF to use LU 6.2 protocols to communicate with this remote system. The APPC keyword is valid only for releases of the CA OPS/MVS product after 2.2.0.

CCI

(Optional) If you are using CAICCI at your site, this value tells the MSF to use the CAICCI cross-platform communications services to communicate with the remote system. To use this value, you must have CCS for z/OS installed on your system, and the CAICCI interface must be enabled. For more information, see the CCS for z/OS documentation.

Each local CAICCI system must have a unique system identifier (sysid). ADDRESS OPSTL MSF uses the sysid to establish a connection to a remote system. For example:

```
ADDRESS OPSTL "MSF DEFINE MSFID(OPSS0B) APPLID(CCI0B) CCI"
```

Note: CCI0B is the sysid of CCI on system B.

AP

(Optional) An optional keyword, when specified on a remote system, which is used to identify a connection to a CA Automation Point system. This connection is actually a special case of a CCI connected system. However, since the remote CA Automation Point system has different capabilities than a remote CA OPS/MVS system, a different connection type designation is used to avoid sending commands to CA Automation Point that it does not support. To use this value, you must have the CCS for z/OS installed on your system, and the CAICCI interface must be enabled. For more information, see the CCS for z/OS documentation.

SECURE and NOSECURE

(Optional) To help you secure your automation environment, specify the optional SECURE or NOSECURE keyword when using the MSF DEFINE command to define an MSF system. These keywords can be particularly useful in an environment where you want to connect production and test systems through the MSF.

For example, by specifying SECURE on your MSF DEFINE command, you indicate that the remote system you are defining is secure, and thus any commands received from that system can be processed by the local system. In contrast, specify NOSECURE to indicate that the remote system is not secure (that is, it is a test system). When the local system receives a command from a non-secure remote system, it processes the command only if it is a display command. Any commands that could do harm to the production system (such as an ENABLE command) are ignored. The SECURE and NOSECURE keywords are ignored for the local system, whose commands are always processed. SECURE is the default.

LOGMODE(logmode)

(Optional) Defines a logmode name. If you do not specify a logmode name, CA OPS/MVS uses the value of its MSFLOGMODE parameter.

OUTPUT, NOOUTPUT, SYSTEM, SYSWAIT, and WAIT

(Optional) You may specify the OUTPUT, NOOUTPUT, SYSTEM, SYSWAIT, and WAIT keywords described in Keywords Common to ADDRESS OPSCTL MSF Commands in this chapter.

MSF DELETE—Delete Definitions

This command causes the MSF to delete definitions for either the specified system or for all MSF resources.

This command has the following format:

```
ADDRESS OPSCTL "MSF DELETE keywords"  
{msfid(sysname|ALL)}  
[OUTPUT|NOOUTPUT]  
[SYSTEM(sysid)]  
[SYSWAIT(seconds)]  
[WAIT(seconds)]
```

MSFID(sysname or ALL)

Specifies the name of the system to be deleted. System names can contain one to eight characters. The first character must be @, #, or \$ or an alphabetic character, but the remaining characters can be any alphabetic or national character. If you specify ALL in place of sysname, the MSF deletes definitions for all inactive systems.

OUTPUT, NOOUTPUT, SYSTEM, SYSWAIT, and WAIT

(Optional) You may specify the OUTPUT, NOOUTPUT, SYSTEM, SYSWAIT, and WAIT keywords.

More information:

[Keywords Common to ADDRESS OPCTL MSF Commands](#) (see page 281)

MSF LIST—Return Session Information

This command returns information about defined MSF sessions, including session status.

This command has the following format:

```
ADDRESS OPCTL "MSF LIST keywords"  
{msfid(sysname|ALL)}  
[FILPLEX(plexname)]  
[FILTSTAT(A|N)]  
[FILTSUBS(subsys)]  
[FILTSYS(sysname)]  
[SYSTEM(sysid)]  
[SYSWAIT(seconds)]  
[WAIT(seconds)]
```

MSFID(*sysname* or ALL)

Specifies the name of the system whose information is to be returned. System names can contain one to eight characters. The first character must be @, #, or \$ or an alphabetic character, but the remaining characters can be any alphabetic or national character. If you specify ALL in place of *sysname*, CA OPS/MVS returns information about all defined sessions.

FILPLEX (*plexname*)

(Optional) Defines the sysplex name to be used as a filter.

FILTSTAT(A|N)

(Optional) Returns status information. Specify an A to return information only for remote MSF systems whose connections to the local system are ACTIVE. Specify an N to return information only for remote MSF systems whose connections to the local system are in a status other than ACTIVE.

FILTSUBS(*subsys*)

(Optional) Defines the CA OPS/MVS subsystem name used to select the remote system. This is useful when there are multiple copies of CA OPS/MVS running on the same remote system.

FILTSYS(*sysname*)

(Optional) Defines the z/OS SYSNAME as specified in logical parmlib member IEASYSxx used to select the remote MSF system. This is useful when multiple MSF links are defined to a particular system.

SYSTEM, SYSWAIT, and WAIT

(Optional) You may specify the SYSTEM, SYSWAIT, and WAIT keywords described in Keywords Common to ADDRESS OPSCtl MSF Commands in this chapter.

Notes:

- The local system is always excluded when any FILTxxx keyword is specified.
- When multiple FILTxxx keywords are specified, each filter is applied to each remote MSF system.
- FILTSTAT performs filtering on all supported versions of MSF-connected systems.
- Wildcarding on the filters is not supported.

MSF LIST Output

The MSF LIST command returns the following session information to the OPS/REXX external data queue as data fields separated by blanks:

Word Number: 1

Length: 1

Contains a system identifier, L for the local MSF or R for the remote MSF

Word Number: 2

Length: 8

The system name

Word Number: 3-10

Length: 8

Words 3 through 10 may contain up to eight system aliases (one alias per word, with each word having a length of eight). If you have defined no systems, the value N/A appears in each word.

Word Number: 11

Length: 8

Contains the VTAM applid for the system

Word Number: 12

Length: 8

Contains the VTAM logmode for the system, or N/A

Word Number: 13

Length: 2

Contains a VTAM return code, shown as two hexadecimal characters

Word Number: 14

Length: 2

Contains a VTAM feedback code, shown as two hexadecimal characters

Word Number: 15

Length: 4

Contains one of these values, depending upon how the MSF session was defined:

- AP
- APPC
- CCI
- N/A (for the local MSF system)

Word Number: 16

Length: 5

Contains the retry time, in seconds

Word Number: 17

Length: 5

Contains the maximum number of retry attempts allowed

Word Number: 18

Length: 5

Contains the current number of retries

Word Number: 19

Length: 3

Contains the expected network delay, in seconds

Word Number: 20

Length: 8

Contains one of the following session statuses: ACTIVE, INACTIVE, FAILED, RETRYING, or PND-INCT (pending inactive)

For the local system record, this word contains the status of the MSF connection to VTAM, and word number 23 contains the status of the MSF connection to CAICCI.

Word Number: 21

Length: 8

Contains the product name (OPS/MSF or Conserve)

Word Number: 22

Length: 8

Contains the version code of OPS/MSF

Word Number: 23

Length: 8

Contains the information in this word differs depending on whether the record applies to a local system or a remote system (see the description of word number 1).

For a remote system record, this word contains a value (SECURE or NOSECURE) indicating the security status that was assigned to the remote system when the system was defined through the MSF DEFINE command. If the MSF DEFINE command was used to define the remote system but the SECURE/NOSECURE keyword was omitted, the default is SECURE, and the value of word 23 is SECURE.

For the local system record, this word contains the status of the MSF connection to CAICCI. This status may be one of the following: ACTIVE, INACTIVE, FAILED.

Word Number: 24

Length: 8

Contains the SYSNAME as defined in the active IEASYSxx member of the Logical Parmlib Concatenation on the remote system. If this information has not yet been obtained from the remote system, then this field contains the value N/A.

Word Number: 25

Length: 4

Contains the SMF identifier as defined on the SID keyword in the SMFPRMxx member of the Logical Parmlib Concatenation on the remote system. If this information has not yet been obtained from the remote system, then this field contains the value N/A.

Word Number: 26

Length: 8

Contains the name of the sysplex in which the remote system is a member. If this information has not yet been obtained from the remote system, then this field contains the value N/A.

Word Number: 27

Length: 4

Contains the CA OPS/MVS subsystem identifier of the remote CA OPS/MVS system. If this information has not yet been obtained from the remote system, then this field contains the value N/A.

Word Number: 28

Length: 8

Contains the value of CA OPS/MVS parameter SSMPLEXNAME on the remote system. If no SSMplex name is defined, then the default value is NONE. If this information has not yet been obtained from the remote system, then this field contains the value N/A.

Word Number: 29

Length: 5

Contains the value of CA OPS/MVS parameter SSMPRIORITY on the remote system. If this information has not yet been obtained from the remote system, then this field contains the value 0.

Word Number: 30

Length: 1

Contains the first character of the value of CA OPS/MVS parameter SSMACTIVEGLOBAL on the remote system.

Specifies a value of Y if the remote system is an SSMGA active global status manager, or a value of N if not.

If this information has not yet been obtained from the remote system, then this field contains the value N.

MSF START—Start MSF

This command tells CA OPS/MVS to start the MSF on the local system. The MSF opens its VTAM access control block (ACB) and becomes able to send requests to or receive data from CA OPS/MVS copies running on remote systems.

This command has the following format:

```
ADDRESS OPSCTL "MSF START keywords"  
[NORETRY|RETRY(seconds retries)]  
[OUTPUT|NOOUTPUT]  
[WAIT(seconds)]
```

NORETRY

(Optional) Causes the MSF to execute the START command with no result if the MSF cannot open its VTAM ACB (for example, if VTAM is not active).

RETRY(*seconds retries*)

(Optional) If the MSF cannot open its VTAM ACB, RETRY causes the MSF to try again every *seconds* seconds to the maximum number of attempts specified with *retries*. You can specify a value between 30 and 86400 for seconds; the default is 30. You can specify a value between 0 and 65535 for retries; the default is 0.

Note: RETRY values are only applicable when the local system attempts to open its VTAM ACB. The CCI subtask, once started, automatically retries any failed attempts to connect to the CAICCI component of CCS for z/OS up to 1000 times every 30 seconds without causing any delay to the MSF main task. If the INITCCI parameter is changed to OFF or NO while the CCI retry is in progress, then the CCI subtask will terminate following the end of the next 30-second interval.

OUTPUT, NOOUTPUT, and WAIT

(Optional) You may specify the OUTPUT, NOOUTPUT, and WAIT keywords.

More information:

[Keywords Common to ADDRESS OPCTL MSF Commands](#) (see page 281)

MSF STOP—Stop MSF Sessions

This command tells CA OPS/MVS to end its sessions with all other copies of the MSF on other systems and to close its VTAM ACB.

This command has the following format:

```
ADDRESS OPCTL "MSF STOP keywords"  
[OUTPUT|NOOUTPUT]  
[SYSTEM(sysid)]  
[SYSWAIT(nnn)]  
[WAIT(seconds)]
```

OUTPUT, NOOUTPUT, SYSTEM, SYSWAIT, and WAIT

(Optional) You may specify the OUTPUT, NOOUTPUT, SYSTEM, SYSWAIT, and WAIT keywords.

More information:

[Keywords Common to ADDRESS OPCTL MSF Commands](#) (see page 281)

OPSCTL MSF Return Codes

OPSCTL MSF commands produce these return codes:

4

For the MSF DEFINE command, the system definition failed. One of the following conditions occurred:

- The name of the system, an alias, or a VTAM APPLID you are trying to define already exists.
- Either a system by the same name is already defined or the name is an alias for an existing system.
- A failure (for example, a virtual storage shortage) occurred while attempting to allocate the storage necessary to represent the MSF system.

For the MSF DELETE command, no MSF system was deleted. One of the following conditions occurred:

- The system is not defined.
- The system is still active. Active systems must first be deactivated before being deleted.
- A failure occurred while freeing the storage associated with the MSF system being deleted.

For the MSF LIST command, no systems were listed because none have been defined.

For the MSF ACTIVATE command, the system was already active or is not defined.

For the MSF DEACTIVATE command, no MSF system was deactivated. This could mean that the system specified was already inactive or has not been defined.

For the MSF START command, some VTAM related failure occurred (see messages).

For the MSF STOP command, the local system has not been defined.

8

An OPSCTL MSF LIST command was issued, but the CA OPS/MVS product cannot perform the LIST request.

16

For the MSF START command, the local MSF system has not been defined or is already active.

24

Destination does not support SYSTEM() command.

40

No command output returned.

44

The CA OPS/MVS INITMSF parameter is currently set to NO.

Note: When the INITMSF parameter is set to NO (its default value), the RC value for all OPSCTL MSF commands is 44.

70

Current system not defined or active to MSF.

76

Error in command output processing.

80

The MSF request contained an invalid verb.

84

The MSF request contained a command syntax error.

88

An error exists in the tokenization routine.

92

Command disallowed during product termination.

93

The system name specified on an ADDRESS OPSCTL MSF DEFAULT command does not exist or is not active.

99

The sending system is considered non-secure.

Keywords Common to ADDRESS OPSCTL MSF Commands

This section describes the SYSTEM, SYSWAIT, WAIT, OUTPUT, and NOOUTPUT keywords.

Note: OUTPUT/NOOUTPUT does not apply to the OPSCTL MSF DEFAULT and OPSCTL MSF LIST commands.

- Keywords Common to All OPSCTL MSF Commands

SYSTEM(sysid)

Defines the name of the remote or local MSF system that is to receive the command. The value of sysid can contain from one to eight characters.

CA OPS/MVS does not check to see whether the sysid system is active when the host command executes. In place of sysid, you may specify either of these values:

ALL

Indicates that the command should be routed to all active MSF systems (including the system on which the command was issued).

EXT

Indicates that the command should be routed to all active MSF systems, with the exception of the system on which the command was issued.

If you do not specify the SYSTEM keyword, the value defaults to the value specified on the OPSCTL MSF DEFAULT SYSTEM command.

SYSWAIT(nnn)

Defines the maximum time that the command waits for output from the remote system. You may specify a value between 1 and 300 seconds.

If you do not specify the SYSWAIT keyword, the value defaults to the value specified on the OPSCTL MSF DEFAULT SYSWAIT command.

- Keywords Common to Most OPSCTL MSF Commands

With the exception of the OPSCTL MSF DEFAULT and OPSCTL MSF LIST commands, all ADDRESS OPSCTL MSF host commands share the following keywords:

OUTPUT or NOOUTPUT

Determines whether the command returns output to the external data queue. OUTPUT and NOOUTPUT are mutually exclusive. Specify OUTPUT to have output returned; otherwise, specify NOOUTPUT.

With the exception of the OPSCTL MSF DEFAULT, all ADDRESS OPSCTL MSF host commands share the following keyword:

WAIT(seconds)

Defines the maximum time that the ADDRESS OPSCTL command waits for output. Omit the WAIT keyword if you specify NOOUTPUT, because there is no reason to wait unless you want to collect output. The CA OPS/MVS product ignores any command output generated *after* the indicated number of seconds has expired.

There is no default value for WAIT. CA OPS/MVS does not check to see if the value you specify is in a particular range of seconds.

ADDRESS OPSCTL Commands for OPSLOG

ADDRESS OPSCTL OPSLOG commands enable you to manage and control the OPSLOGs for use by your CA OPS/MVS subsystem. Each copy of CA OPS/MVS that is active on your system will use a unique set of OPSLOGs.

You may use the following types of OPSLOGs:

- DIV-backed OPSLOG data sets - the data is backed to disk and retained across product restarts.
- In-storage OPSLOGs - not backed to disk - the data is discarded when the log is inactivated or the product terminates.

Each OPSLOG can have the following formats:

- Read/write

These OPSLOGs are eligible to be the live OPSLOG.

- Read-only

These OPSLOGs contain restored data from the OPSLOG archival system.

Note: A *live* OPSLOG is the log currently being updated with new events as they occur in the system.

Warnings:

- DIV-backed OPSLOGs that are used as the live OPSLOG must be treated like a page data set.
- Do not use large in-storage OPSLOGs unless you are certain that your page data sets can handle the potential increased paging load.
- When introducing multiple OPSLOGs into your existing CA OPS/MVS configuration for the first time, and have the OPSLOG archive component previously implemented, you must ensure that CA OPS/MVS r11.6 (or higher) archive components are being utilized.

For information about migration, see the Release Notes.

Each copy of CA OPS/MVS that is active on your system must use unique OPSLOG data sets if DIV-backed OPSLOGs are being used. Even if you are able to do so, it is unsafe and unreliable to have an OPSLOG DIV data set being used as the live OPSLOG on more than one CA OPS/MVS subsystem.

OPSLOG ACTIVATE—Activates an OPSLOG

The OPSLOG ACTIVATE command makes a previously defined OPSLOG definition active for use. If the OPSLOG is DIV-backed, the data set must exist at the time it is activated.

This command has the following format:

```
ADDRESS OPSCTL "OPSLOG ACTIVATE(logname) optional-keywords"  
[BROWSEMAX(nnnnnn)][READONLY|READWRITE]  
[SYSTEM(sysid)]  
[SYSWAIT(nnn)]
```

ACTIVATE(logname)

Defines the log name of one to sixteen characters.

BROWSEMAX

(Optional) Defines the maximum number of messages in the OPSLOG message area.

It lets you override the BROWSEMAX value specified on the OPSLOG DEFINE command or defaulted from the BROWSEMAX product parameter.

Value: A numeric value in the range from 1 to 4925000

READONLY|READWRITE

(Optional) Overrides any settings from the earlier DEFINE command.

These keywords are mutually exclusive.

READONLY

Indicates that the OPSLOG cannot be updated. This keyword is typically used when activating a restored OPSLOG.

READWRITE

(Default) Indicates that the OPSLOG can be updated. An OPSLOG must have the READWRITE attribute to be eligible for the OPSLOG SETLIVE command.

SYSTEM and SYSWAIT

Lets you specify the common keywords described in Keywords Common to All ADDRESS OPSCTL OPSLOG Commands in this chapter.

Note: The OPSLOG ACTIVATE command execution is asynchronous. The return code from this command only indicates that the command syntax used is valid and that initial command checking did not find any errors. Actual command execution is performed later (after the issuing REXX program or AOF rule resumes execution) under a different task and most likely in a different address space than the one the command was issued from. One example of such an error is attempting to activate an OPSLOG where the specified DIV data set does not exist. The OPSLOG status will be pending active until the activation is complete. In the case where a large DIV-backed OPSLOG is being used for the first time, it may take a minute or even longer for the asynchronous execution to complete. REXX code that needs to validate that the activation has completed should wait for a short period of time and then check the output from an OPSLOG LIST command to validate that the activation completed successfully.

Example: ACTIVATE command

- This command activates an already defined OPSLOG with a log name of DivLog2 (the log name is not case sensitive and could be specified as DIVLOG2 with the same results). The BROWSEMAX and read/write settings specified or defaulted, on the prior DEFINE statement for this OPSLOG, are used.

```
ADDRESS OPSCTL "OPSLOG ACTIVATE(DivLog2)"
```

- This command activates an already defined OPSLOG with a log name of DivLog2. Any BROWSEMAX and READWRITE values set for this OPSLOG on the prior DEFINE statement are overridden by this command.

```
ADDRESS OPSCTL "OPSLOG ACTIVATE(DivLog2) BROWSEMAX(100000)READONLY"
```


OPSLOG DEACTIVATE—Deactivate an OPSLOG

The OPSLOG DEACTIVATE command makes a previously active OPSLOG definition inactive for use.

This command has the following format:

```
ADDRESS OPSCTL "OPSLOG {DEACTIVATE(logname)} optional-keywords"  
[SYSTEM(sysid)]  
[SYSWAIT(nnn)]
```

DEACTIVATE(logname)

Defines the log name of one to sixteen characters.

SYSTEM and SYSWAIT

(Optional) Lets you specify the common keywords described in Keywords Common to All ADDRESS OPSCTL OPSLOG Commands in this chapter.

Note: The OPSLOG DEACTIVATE command execution is asynchronous. The return code from this command only indicates that the command syntax used is valid and that initial command checking did not find any errors. Actual command execution is performed later (after the issuing REXX program or AOF rule resumes execution) under a different task and most likely in a different address space than the one the command was issued from. The status of the OPSLOG will be “pending inactive” until the deactivation is complete. In the case where a DIV-backed OPSLOG is being used, the asynchronous processing includes a final checkpoint to write the data from the OPSLOG data space to the DIV data set. This process may take a few seconds to complete. REXX code that needs to validate that the deactivation has completed should wait for a short period of time and then check the output from an OPSLOG LIST command to validate that the deactivation completed successfully.

Example: DEACTIVATE command

This command deactivates an active OPSLOG with a log name of DivLog2.

```
ADDRESS OPSCTL "OPSLOG DEACTIVATE(DivLog2)"
```

When completed, the OPSLOG named DivLog2 can no longer be browsed using either OPSLOG Browse or OPSLOG WebView.

OPSLOG DEFINE—Define an OPSLOG

The OPSLOG DEFINE command creates a new OPSLOG definition.

This command has the following format:

```
ADDRESS OPSCtl "OPSLOG {DEFINE(logname)} optional-keywords"  
[DSNAME(divdsname)]  
[BROWSEMEX(nnnnn)] [READONLY|READWRITE]  
[SYSTEM(sysid)]  
[SYSWAIT(nnn)]
```

DEFINE(logname)

Defines a unique log name of one to sixteen characters.

The name must be 1-16 characters in length, and must contain only alphabetic, numeric, or any of the following special characters:

- \$ - dollar sign
- * - pound sign or hash
- @ - at sign
- . - period
- _ - underscore
- ! - exclamation point
- ? - question mark

The name is not case sensitive. It cannot start with a numeric character, an underscore (_) or a question mark (?), and cannot end with an underscore (_). Imbedded blanks are not allowed. The LOGNAME must be unique for each CA OPS/MVS subsystem.

The following name is reserved and should not be used: INSTAUTODEFINED.

DSNAME

(Optional) For DIV data sets, defines the actual data set name of the physical data set. In this case, the data set name must conform to standard z/OS data set naming restrictions.

For an in-storage OPSLOG, you must start the pseudo- data set name with the character string 'IN-STORAGE' and it is not required to conform to standard z/OS data set naming restrictions.

Do not use the following reserved name: IN-STORAGEAUTO.

For a DIV-backed data set name, it is possible to define the OPSLOG prior to the data set being created. The data set must actually exist at the time the OPSLOG is activated.

BROWSEMAX

(Optional) Defines the maximum number of messages in the OPSLOG message area.

It lets you override the default value which is the BROWSEMAX parameter value.

For a DIV-backed OPSLOG that has already been used, the value you specify will be overridden by the existing BROWSEMAX value in the data set.

Value: A numeric value in the range from 1 to 4925000.

READONLY|READWRITE

(Optional) Indicates the mode in which the data set is to be accessed. These keywords are mutually exclusive. The value can be subsequently overridden on a future OPSLOG ACTIVATE command.

READONLY

Specifies that the OPSLOG cannot be updated. This keyword is typically used when defining an OPSLOG that will be used to hold a restored OPSLOG.

READWRITE

(Default) Specifies that the OPSLOG can be updated. An OPSLOG must have the READWRITE attribute to be eligible for the OPSLOG SETLIVE command.

SYSTEM and SYSWAIT

(Optional) You may specify the common keywords described in Keywords Common to All ADDRESS OPSCTL OPSLOG Commands in this chapter.

Example: DEFINE command

- This command defines a new in-storage OPSLOG named InstLog2. Unless the BROWSEMAX value is subsequently overridden on an ACTIVATE command, this OPSLOG will use the value specified by the BROWSEMAX product parameter. Unless the read/write status is subsequently overridden on an ACTIVATE command, this OPSLOG will be READWRITE.

```
ADDRESS OPSCTL "OPSLOG DEFINE(InstLog2) DSNAME(IN-STORAGE)"
```

- This command defines an OPSLOG with a log name of RestLog1 and a DIV data set name of OPSMVS.REST.OPSLOG for the purpose of being able to browse a restored OPSLOG. As recommended the restored log is defined as being READONLY. Assuming that the data set already contains a restored OPSLOG, the BROWSEMAX value will be automatically determined at activation from the values stored in the restored OPSLOG.

```
ADDRESS OPSCTL "OPSLOG DEFINE(RestLog1) DSNAME(OPSMVS.REST.OPSLOG) READONLY"
```

- This command defines an OPSLOG with a log name of DivLogTemp and a DIV data set name of OPSMVS.TMP.OPSLOG. The DIV data is not required to exist at the time the DEFINE command is executed. Unless overridden at activation time, the BROWSEMAX value is set at 100000 and the data set will be used as READWRITE.

```
ADDRESS OPSCTL "OPSLOG DEFINE(DivLogTemp) DSNAME(OPSMVS.TMP.OPSLOG)
BROWSEMAX(100000)"
```

OPSLOG DELETE—Delete an OPSLOG

The OPSLOG DELETE command tells CA OPS/MVS to delete the definition for an OPSLOG.

This command has the following format:

```
ADDRESS OPSTCL "OPSLOG {DELETE(logname)} optional-keywords"  
SYSTEM(sysid)  
SYSWAIT(nnn)
```

DELETE(logname)

Defines the one- to sixteen-character log name.

SYSTEM and SYSWAIT

(Optional) You may specify the SYSTEM and SYSWAIT keywords described in Keywords Common to ADDRESS OPSTCL OPSLOG Commands in this chapter.

Example: DELETE command

This command deletes the definition for an OPSLOG with a log name of TempLog. TempLog must be defined but not active or live at the time this command is executed.

```
ADDRESS OPSTCL "OPSLOG DELETE(TempLog)"
```

OPSLOG LIST—List All OPSLOG Definitions

The OPSLOG LIST command lists all of the existing OPSLOG definitions.

This command has the following format:

```
ADDRESS OPSTCL "OPSLOG LIST optional-keywords"  
SYSTEM(sysid)  
SYSWAIT(nnn)
```

SYSTEM and SYSWAIT

(Optional) You may specify the common keywords described in Keywords Common to All ADDRESS OPSTCL OPSLOG Commands in this chapter.

Output from OPSLOG LIST Command

The OPSLOG LIST command returns the following OPSLOG information to the OPS/REXX external data queue as data fields separated by blanks. One record is produced for each defined OPSLOG.

Word Number: 1

Length: 16

Contains the OPSLOG log name.

Word Number: 2

Length: 8

Contains the OPSLOG DDNAME. This value is internally generated except for the case where the OPSLOG DD is allocated to the main address space via JCL or dynamic allocation.

Word Number: 3

Length: 44

Contains the OPSLOG DIV data set name or a pseudo-data set name for an in-storage OPSLOG.

Word Number: 4

Length: 10

Contains the BROWSEMAX value defined or actually being used for this OPSLOG (depending on the status). This is a numeric value in the range of 0 to 4925000.

Word Number: 5

Length: 1

Contains a flag indicating whether or not this OPSLOG is read-only:

- R - Read-only
- W - Read/write

Word Number: 6

Length: 1

Contains a flag indicating whether this is the live OPSLOG:

- L- Live OPSLOG
- N-Non-live OPSLOG

Word Number: 7

Length: 1

Contains a flag indicating how this OPSLOG was defined:

- A - The OPSLOG was automatically defined
- D - The OPSLOG was explicitly defined by an ADDRESS OPSCTL "OPSLOG DEFINE" command

Word Number: 8

Length: 1

Contains a flag indicating how this OPSLOG is backed:

- D - The OPSLOG is DIV-backed
- S - The OPSLOG exists only in a data space attached to the OPS/MVS main address space and is not saved on disk

Word Number: 9

Length: 8

Contains the status of this OPSLOG definition. The possible status values are:

- Defined - The OPSLOG is defined.
- Active - The OPSLOG is active and eligible for Browsing.
- PendAct - The OPSLOG is pending activation. An OPSLOG ACTIVATE command has been issued but the activation process has not yet completed.
- PendIAct - The OPSLOG is pending deactivation. A OPSLOG DEACTIVATE command has been issued but the deactivation process has not yet completed.
- PendRest - The OPSLOG is pending reset. An OPSLOG RESET command has been issued but the reset process has not yet completed.

Word Number: 10

Length: 10

Lock value. This is an internal field that may be used for debugging and is not intended for customer use.

Note: The OPSLOG LIST command cannot be used with the SYSTEM keyword and the SYSID of a remote MSF system in a NOWAIT AOF rule. In that environment the command will fail with an RC of 116. Using a SYSID of * or the name (or alias) of the local system will be successful.

Example: LIST command

This command lists all of the defined OPSLOGs (including those that are active and live) to the external data queue of the OPS/REXX program or AOF rule that issued the command.

```
ADDRESS OPSCtl "OPSLOG LIST"
```

OPSLOG RESET—Reset Active OPSLOG

The OPSLOG RESET command tells CA OPS/MVS to reset an active OPSLOG.

This command has the following format:

```
ADDRESS OPSCtl "OPSLOG {RESET(logname)} optional-keywords"  
SYSTEM(sysid)  
SYSWAIT(nnn)
```

RESET(logname)

Defines the one- to sixteen-character log name.

SYSTEM and SYSWAIT

(Optional) You may specify the SYSTEM and SYSWAIT keywords described in Keywords Common to ADDRESS OPSCtl OPSLOG Commands in this chapter.

Usage Notes:

- Do not reset an OPSLOG until you are certain that you no longer need to browse the events in that log.
- If you are archiving OPSLOG data you should wait until after the final archive has completed for this OPSLOG before resetting the log. The archival system schedules the final archive some time (up to two minutes) after the SETLIVE command was executed that switched the live OPSLOG away from this log.

- The OPSLOG RESET command execution is asynchronous. The return code indicates that the command syntax used is valid and that initial command checking did not find any errors. Actual command execution is performed later (after the issuing REXX program or AOF rule resumes execution) under a different task and in a different address space than the one the command was issued from. The OPSLOG status will be pending reset until the reset is complete. The reset process is normally very fast and unless there is a delay in the OPSLOG Execute Processor subtask in the OPS/MVS main address space it is unlikely that any delays will be noticed. Nevertheless REXX code that needs to validate that the reset has completed should wait for a short period of time and then check the output from an OPSLOG LIST command to validate that the reset completed successfully and that the OPSLOG is in Active status.

Example: RESET command

Reset empties the OPSLOG with a log name of TempLog. If this OPSLOG is made live, it will begin recording with event sequence number 1. After a previously live log completes its final archive and is no longer needed for browsing, you should reset this log before making it live again.

```
ADDRESS OPSCTL "OPSLOG RESET(TempLog)"
```

OPSLOG SETLIVE—Make OPSLOG Live

The OPSLOG SETLIVE command tells CA OPS/MVS to make the requested OPSLOG the live OPSLOG. The live OPSLOG is the log currently used to record events as they occur in the system.

This command has the following format:

```
ADDRESS OPSCTL "OPSLOG {SETLIVE(logname)} optional-keywords"
```

SYSTEM(sysid)

SYSWAIT(nnn)

SETLIVE(logname)

Defines the one- to sixteen-character log name.

SYSTEM and SYSWAIT

(Optional) You may specify the SYSTEM and SYSWAIT keywords described in Keywords Common to ADDRESS OPSCTL OPSLOG Commands in this chapter.

Usage Notes:

- The OPSLOG SETLIVE command execution is somewhat asynchronous. The return code from this command indicates that the command syntax used is valid and that the log switch has occurred to the new live OPSLOG. However as part of the SETLIVE processing the final checkpoint for the previously live OPSLOG must be completed asynchronously under a task in the OPS/MVS main address space. By that time the REXX program or AOF rule that issued this command has resumed execution. The status of the new live OPSLOG is reflected immediately upon completion of this command.
- Immediately prior to the OPSLOG switch a message is issued that is recorded in the previously live OPSLOG indicating the switch to the new live OPSLOG and immediately following the switch a message is issued that is recorded to the new live OPSLOG indicating the name of the previously live OPSLOG. For example the previously live log will contain a message similar to the following:

OPS4603H NewLogName is now the live OPSLOG and the new live log will contain a message similar to the following:

OPS4603H OldLogName was the live OPSLOG

Example: SETLIVE command

This command makes TempLog the live OPSLOG. TempLog must be active and read/write for this command to succeed.

```
ADDRESS OPSCTL "OPSLOG SETLIVE(TempLog)"
```

OPSCTL OPSLOG Return Codes

OPSCTL OPSLOG commands produce these return codes:

111

The ADDRESS OPSCTL OPSLOG command string is too short. No command verb was supplied.

Examples of valid command verbs are: DEFINE, ACTIVATE and SETLIVE.

112

A syntax error was detected while parsing the OPSCTL OPSLOG command.

113

A syntax error was detected while scanning the OPSCTL OPSLOG command.

114

The log name specified is invalid. Log names must be from 1 to 16 characters in length and may contain any characters that are valid in OPS/REXX variable names. They cannot be all numeric and cannot start with a numeric character.

115

Either MSF is not active or the SYSTEM keyword was specified with an MSF SYSID that is invalid or inactive.

116

The OPCTL OPSLOG LIST command cannot be used with the SYSTEM keyword in a NOWAIT environment.

117

A cross-system OPCTL OPSLOG command was rejected because the target system is at a release below 11.06.00 and does not support this command.

121

A DEFINE command failed because the log name specified is already defined. Log names must be unique.

122

A DEFINE command failed because the data set name or pseudo-data set name specified is already defined. Data set names must be unique.

123

A DEFINE command failed because the maximum number of OPSLOGs are already defined.

124

A DEFINE command failed because of an internal 'lock' failure.

131

An ACTIVATE command failed because the maximum number of OPSLOGs are already activated. The maximum number of active OPSLOGs is controlled by the BROWSEACTIVEMAX product parameter.

132

An ACTIVATE command failed because the requested log name is already the live OPSLOG.

133

An ACTIVATE command failed because the requested log name is either already active or is in transition.

141

A SETLIVE command failed because the requested log name is already the live OPSLOG.

142

A SETLIVE or RESET command failed because the named OPSLOG is read-only.

143

A SETLIVE or RESET command failed because the named OPSLOG is not currently active and is therefore ineligible for this command.

144

A SETLIVE command failed because of an internal lock failure.

145

A SETLIVE command failed because of an internal failure. OPS_DSP address is zero.

146

A SETLIVE command failed because a non-normal UCBLEVEL value has been set for the device on which the OPSLOG data set resides. This is typically a short-lived condition and may be caused by EMC AutoSwap processing. Wait for a short time before retrying the command.

147

A RESET command failed because the named OPSLOG is currently the live OPSLOG.

148

A RESET command failed because of an internal lock failure.

149

A RESET command failed because of an internal unlock failure.

151

A DEACTIVATE command failed because the named OPSLOG is currently the live OPSLOG.

152

A DEACTIVATE command failed because the named OPSLOG is not currently active and is therefore ineligible for this command.

153

A DEACTIVATE command failed because there is a checkpoint pending for the named OPSLOG. A checkpoint writes the data in the OPSLOG data space to a DIV data set. Wait for the checkpoint to complete before retrying this command.

154

A DEACTIVATE command failed because there is a reset pending for the named OPSLOG. Wait for the reset to complete before retrying this command.

155

A DEACTIVATE command failed because there is a high checkpoint pending for the named OPSLOG. A high-checkpoint only occurs when a new DIV-backed OPSLOG is first used and may take some time to complete. Wait for the high checkpoint to complete before retrying this command.

161

A DELETE command failed because the named OPSLOG is currently the live OPSLOG.

162

A DELETE command failed because the named OPSLOG was not defined using an OPCTL OPSLOG DEFINE command.

163

A DELETE command failed because the named OPSLOG is not currently inactive and is therefore ineligible for this command.

169

The current command failed due to a failure in unlocking the internal OPSLOG control block.

171

A LIST command failed because there are no OPSLOG entries to list.

177

A LIST command failed because of a missing buffer address. This internal error can only occur if there is an error in OPSLOG WebView.

178

A LIST command failed because of an internal buffer error (zero entries). This internal error can only occur if there is an error in OPSLOG WebView.

179

A LIST command failed because of an internal buffer overflow. This internal error can only occur if there is an error in OPSLOG WebView.

191

The log name specified on the current command does not exist on this OPS/MVS subsystem.

200

An attempt to send a message to a message queue failed. Check the external data queue and the OPSLOG for messages relating to a "send failure".

201

An attempt to obtain storage failed. Check the external data queue and the OPSLOG for messages relating to storage failures.

202

No output was received as the result of a cross-system command.

203

Message queue allocation failed.

204

Message queue deletion failed.

205

Error in command output processing.

206

An error occurred attempting to send a message to an MSF queue.

207

The control blocks required for OPCTL OPSLOG processing have not been created yet.

208

An ABEND occurred during OPCTL OPSLOG processing.

Keywords Common to ADDRESS OPCTL OPSLOG Commands

The following SYSTEM and SYSWAIT keywords can be used in any OPCTL OPSLOG command.

SYSTEM(sysid)

Defines the name of the remote or local MSF system that is to receive the command. The value of sysid can contain from one to eight characters. See the detailed syntax restrictions for an MSF system name in the section titled “MSF DEFINE Command” in this chapter.

SYSWAIT(nnn)

Defines the maximum time that the command waits for output from the remote system. You may specify a value between 1 and 300 seconds.

ADDRESS OPCTL Commands for OSF

The next several sections describe ADDRESS OPCTL commands for the OSF.

OSF EXECSTATS—Return Performance Statistics

This command returns information about the performance of the OSF component to the OPS/REXX external data queue. You can use this information to help you to fine-tune your OSF-related parameters to meet the performance objectives of your installation.

This command has the following format:

```
ADDRESS OPSCTL "OSF EXECSTATS keywords"  
[OUTPUT|NOOUTPUT]  
[SYSTEM(sysid)]  
[SYSWAIT(seconds)]  
[CLASS(class)]
```

OUTPUT, NOOUTPUT, SYSTEM, SYSWAIT, and CLASS

(Optional) You may specify the OUTPUT, NOOUTPUT, SYSTEM, SYSWAIT, and CLASS keywords described in Keywords Common to All ADDRESS OPSCTL OSF Commands in this chapter.

Output from OSF EXECSTATS Command

The OSF EXECSTATS command returns the following output data, separated by blanks, in the external data queue. OPSVIEW option 4.3 also displays this information.

Word Number: 1

Length: 7

Contains the total number of times a server was started.

Word Number: 2

Length: 7

Contains the number of times a server was started to maintain the number of servers specified by the OSFMIN, OSFTSLMIN, OSFTSPMIN, or USSMIN parameter as appropriate.

Word Number: 3

Length: 7

Contains the number of times a server was started because the OSF execution queue depth was greater than or equal to the value of the OSFQADD, OSFTSLQADD, OSFTSPQADD, or USSQADD parameter as appropriate.

Word Number: 4

Length: 7

Contains the number of times a server was started because an earlier server start command failed to complete successfully; the OSFALLOWRESTART and USSALLOWRESTART parameters control server restarts.

Word Number: 5

Length: 7

Contains the number of OSF commands that were executed.

Word Number: 6

Length: 10

Contains the average elapsed time, in seconds, that each OSF command took to complete.

Word Number: 7

Length: 10

Contains the minimum elapsed time, in seconds, that an OSF command took to complete.

Word Number: 8

Length: 10

Contains the maximum elapsed time, in seconds, that an OSF command took to complete.

Word Number: 9

Length: 7

Contains the number of times an OSF server has been terminated due to the number of completed transactions in that server being equal to or greater than the value of the OSFRECYCLE parameter.

Word Number: 10

Length: 7

Contains the highest number of non-terminating OSF servers that have been in service since CA OPS/MVS was started.

OSF LIST—Return Active Server Statistics

This command returns information about active OSF servers to the OPS/REXX external data queue.

This command has the following format:

```
ADDRESS OPSCTL "OSF LIST keywords"  
[OUTPUT|NOOUTPUT]  
[SYSTEM(sysid)]  
[SYSWAIT(seconds)]  
[CLASS (class)]
```

OUTPUT, NOOUTPUT, SYSTEM, SYSWAIT, and CLASS

(Optional) You may specify the OUTPUT, NOOUTPUT, SYSTEM, SYSWAIT and CLASS keywords described in Keywords Common to All ADDRESS OPSCTL OSF Commands in this chapter.

Output from OSF LIST Command

For each OSF server, the OSF LIST command returns two output records to the external data queue. OPSVIEW option 4.3 displays these records, inserting blanks between fields and displaying UNAVAIL or N/A in place of fields inappropriate for the status of the server. The first record contains the following data:

Word Number: 1

Length: 4

Contains the address space ID in hexadecimal format.

Word Number: 2

Length: 8

The server job name.

Word Number: 3

Length: 8

Contains one of the following server status values:

- ACTIVE, indicating that the server is processing a request.
- IDLE, indicating that the server is waiting for work.
- INIT, indicating that the server is initializing.
- TERM, indicating that the server is shutting down.
- FAILED, indicating that the server failed to initialize. If the FAILED status persists, contact CA Customer Support at <http://ca.com/support>.
- UNAVAIL, indicating that the server has shut down while processing a request. Should this occur, the remaining fields in this output record will contain N/A, the transaction count, the elapsed time, CPU time used, output lines, and the command text. If the UNAVAIL status persists, contact CA Customer Support at <http://ca.com/support>.

Word Number: 4

Length: 5

Contains a count of total transactions executed on this server.

Word Number: 5

Length: 8

If the server is active, word 5 contains the server elapsed time for the current transaction. Otherwise, the word contains the cumulative elapsed time for all of the transactions this server processed since it started. The data in word 5 will be in one of the following formats:

- sss.tttS, indicating seconds.milliseconds
- hh.mm.ss, indicating hours.minutes.seconds
- hhhh.mm, indicating hours.minutes
- NOTAVAIL, indicating that there is no valid time format

Word Number: 6

Length: 8

If the server is active, word 6 contains the amount of CPU time the server has used so far to process the current transaction. Otherwise, the value of word 6 is cumulative for all transactions the server has processed since it started. The data in word 6 will be in one of the following formats:

sss.tttS, indicating seconds.milliseconds

hh.mm.ss, indicating hours.minutes.seconds

hhhhh.mm, indicating hours.minutes

NOTAVAIL, indicating that there is no valid time format

Word Number: 7

Length: 7

If the server is active, the value of word 7 indicates the number of output lines produced for the current transaction. Otherwise, the value is cumulative for all transactions the server has processed since it started.

Word Number: 8

Length: 1

Contains either Y or N, indicating whether this server acquired a JES job ID.

Word Number: 9

Length: 27

Contains debugging information for use by CA customer support.

The second record that the OSF LIST command returns contains 256 bytes of the current (or last) command text. This text occupies a separate record to allow you to use the OPS/REXX PARSE statements, WORD function, or both to extract data from the first record.

OSF QUEUES—Display Server Queue Status

This command displays the status and history of the OSF server queue.

This command has the following format:

```
ADDRESS OPSTCL "OSF QUEUES keywords"  
[OUTPUT|NOOUTPUT]  
[SYSTEM(sysid)]  
[SYSWAIT(seconds)]  
[CLASS(class)]
```

OUTPUT, NOOUTPUT, SYSTEM, SYSWAIT, and CLASS

(Optional) You may specify the OUTPUT, NOOUTPUT, SYSTEM, SYSWAIT, and CLASS keywords described in Keywords Common to All ADDRESS OPSTCL OSF Commands in this chapter.

Output from OSF QUEUES

The OSF QUEUES command returns the following output data, separated by blanks, in the external data queue. OPSVIEW option 4.3 also displays this information.

Word Number: 1

Length: 10

Contains the average length of time, in seconds, that a command remains on the queue before the OSF dispatches it to a server.

Word Number: 2

Length: 8

Contains the number of commands currently on the queue.

Word Number: 3

Length: 8

The maximum number of commands that have been on the queue at one time (since CA OPS/MVS was last started).

Word Number: 4 - 24

Length: Variable, each with a length between 1 and 8

Contains a total of 21 transaction counters—one for each time the OSF execute queue was empty, had one transaction waiting, had two transactions waiting, and so on up to 19 transactions waiting. There is also a single counter for each time the queue had 20 or more pending transactions.

The counters illustrate the frequency distribution of observed queue depths. The queue depth is sampled each time a command is removed from the top of the queue for execution by a server.

Word Number: 25

Length: 10

Contains the minimum length of time, in seconds, that a command remained on the queue before being dispatched to a server.

Word Number: 26

Length: 10

The maximum length of time, in seconds, that a command remained on the queue before being dispatched to a server.

Word Number: 27

Length: 10

Contains the number of commands that were added to the OSF server queue.

OSF RESETQ—Reset Execute Queue

This command causes all pending commands waiting on the OSF execute queue to be immediately discarded. As a result of this command, all existing OSF execute queue statistics are reset to 0 (zero).

Use this command only in emergency situations, such as if a looping rule or REXX program erroneously fills up the OSF execute queue.

This command has the following format:

```
ADDRESS OPSCTL "OSF RESETQ keywords"  
[OUTPUT|NOOUTPUT]  
[SYSTEM(sysid)]  
[SYSWAIT(seconds)]  
[CLASS(class)]
```

OUTPUT, NOOUTPUT, SYSTEM, SYSWAIT, and CLASS

(Optional) You may specify the OUTPUT, NOOUTPUT, SYSTEM, SYSWAIT, and CLASS keywords described in Keywords Common to All ADDRESS OPSCTL OSF Commands in this chapter.

OSF STOP—Stop the Server Address Space

This command stops the server with address space ID *nnnn*.

This command has the following format:

```
ADDRESS OPSCTL "OSF STOP keywords"  
  {nnnn}  
  [OUTPUT|NOOUTPUT]  
  [SYSTEM(sysid)]  
  [SYSWAIT(seconds)]  
  [CLASS(class)]
```

nnnn

Defines the one- to four-character address space ID of the server you want to stop, expressed hexadecimal format.

OUTPUT, NOOUTPUT, SYSTEM, SYSWAIT, and CLASS

(Optional) You may specify the OUTPUT, NOOUTPUT, SYSTEM, SYSWAIT, and CLASS keywords described in Keywords Common to All ADDRESS OPSCTL OSF Commands in this chapter.

Terminate a Server Immediately

When an OSF STOP command is issued to a server that is currently running a transaction (ACTIVE status), the server will not be terminated until the current transaction completes.

To terminate an active server immediately, issue the z/OS CANCEL command.

For example, a server executing this REXX program terminates as usual:

```
asid = C2X(OPSINFO('ASID'))  
ADDRESS OPSCTL "OSF STOP" asid  
exit
```

In this example, the server transaction will *not* be terminated until the 30 second OPSWAIT completes and the transaction terminates (assuming that OSFWAIT and OSFRUN are not exceeded):

```
asid = C2X(OPSINFO('ASID'))  
ADDRESS OPSCTL "OSF STOP" asid  
temp = OPSWAIT(30)  
exit
```

OPCTL OSF Return Codes

OPCTL OSF commands produce these return codes:

Return Code	Description
24	The subsystem version is incompatible
40	No command output was received
44	The OSF request contained a syntax error
48	A CA OPS/MVS service routine failure (SENDMSG or ECB POST) occurred
52	Invalid server control block address
56	The address space that you specified on a STOP command is not a server
72	The system ID that you specified is invalid or inactive
76	The last message was not received for command output
80	The server class is invalid
84	The server class is not active

For additional return codes that apply to all OPCTL commands, see ADDRESS OPCTL Return Codes in this chapter.

Keywords Common to All ADDRESS OPCTL OSF Commands

All ADDRESS OPCTL OSF host commands share the following keywords:

CLASS(class)

(Optional) Specifies which class of OSF server to which the command refers. This optional keyword may be specified on any of the ADDRESS OPCTL OSF host commands.

The only possible values for the CLASS keyword are:

- TSL
- TSO
- TSP
- USS

TSO is the default if the CLASS keyword is not specified.

OUTPUT or NOOUTPUT

(Optional) Determines whether the command returns output to the external data queue of the system that originated the command. Specify OUTPUT to have output returned; otherwise, specify NOOUTPUT.

When running in an AOF rule environment, NOOUTPUT is always forced. In all other environments, these defaults exist:

- OSF EXECSTATS-OUTPUT
- OSF LIST-OUTPUT
- OSF QUEUES-OUTPUT
- OSF RESETQ-NOOUTPUT
- OSF STOP-NOOUTPUT

Do not specify NOOUTPUT in conjunction with the SYSWAIT keyword.

Commands issued to the local system ignore the OUTPUT and NOOUTPUT keywords.

SYSTEM(sysid)

(Optional) If your site uses the CA OPS/MVS product on multiple systems connected by the MSF, you can use this keyword to send commands between the systems. CA OPS/MVS routes the command to the system with the specified MSF sysid. You may also specify an asterisk (*) for sysid to route the command to the local system only. If the SYSTEM keyword is not specified, then by default the command is routed to the local system only, regardless of whether MSF is running.

SYSWAIT(seconds)

(Optional) Specifies the maximum time (from 1 to 300 seconds) that the command waits for output from the remote system.

Omit the SYSWAIT keyword if you specify NOOUTPUT, because there is no reason to wait unless you want to collect output.

Commands issued to the local system ignore the SYSWAIT keyword.

ADDRESS OPSDYNAM Host Environment

The ADDRESS OPSDYNAM host environment provides dynamic allocation, unallocation, concatenation, deconcatenation, and information retrieval functions for data sets in the current address space. It is designed to be a functional replacement for the TSO ALLOCATE and FREE commands in an OPS/REXX program. In addition, it provides functionality for concatenating existing allocations and controlling the attributes of allocated data sets. It supports the handling of GDG data sets by relative generation number. It also enhances dynamic allocation performance in an OPS/REXX program.

This command has the following format:

```
ADDRESS OPSDYNAM "keywords"
```

OPSDYNAM Used as a TSO/REXX or OPS/REXX Function

OPSDYNAM may also be executed as a TSO/E or OPS/REXX function with a single character string argument that is the same as the OPSDYNAM command syntax. The return code of the request is returned.

Use the following format to execute a TSO/E or OPS/REXX function with a single character string argument:

```
FRC = OPSDYNAM("ALLOC FILE(xyz) DSN(myfile) SHR")
```

ADDRESS OPSDYNAM Compared to TSO ALLOCATE

Using ADDRESS OPSDYNAM provides the following advantages over TSO ALLOCATE:

- OPSDYNAM does not require the TSO TMP. It can be executed in a batch program or in a CA OPS/MVS TOD, SCR, or REQ rule.
- Execution speed improves five to ten times in an OPS/REXX program using OPSDYNAM.
- OPSDYNAM gives you more control over the output message generation, including REXX/CLIST variable return values.
- OPSDYNAM works in TSO/E as a REXX function.
- OPSDYNAM supports GDG relative generation numbers in user selectable mode (GDGLOCATE keyword).
- OPSDYNAM allows you to allocate subsystem data sets such as CA OPS/MVS GDI data sets.
- OPSDYNAM includes the INFO, CONCAT, and DECONCAT functions of dynamic allocation

ADDRESS OPSDYNAM Commands

The next several sections describe ADDRESS OPSDYNAM commands.

OPSDYNAM ALLOCATE—Allocate Data Set

The ALLOCATE command allocates an existing or new data set, a list of existing data sets, a dummy data set, a subsystem data set, or a SYSOUT data set to a z/OS ddname with selected attributes.

This command has the following format:

```
ADDRESS OPSDYNAM "ALLOCATE keywords"
[FILE|DDNAME(ddname1,ddname2,... ddname65)]
[DATASET|DSNAME(dsname1*,dsname2,... dsname16)]DUMMY]
```

FILE|DDNAME(ddname1, ddname2,...ddname65)

(Optional) Defines a valid z/OS ddname or list of ddnames for the FREE and CONCAT commands. If a ddname is not supplied for the ALLOCATE command, the system generates a ddname of the form SYSnnnnn, which is returned in the REXX variable OPSDD. A maximum of 65 ddnames may be specified.

DATASET|DSNAME (dsname1|*, dsname2, ...dsname16)

(Optional) Specifies a single z/OS data set name or a list of data set names for the ALLOCATE and FREE commands. When a list of data set names is specified for ALLOCATE, the data sets are concatenated to a single ddname. A maximum of 16 data set names may be specified. If the data set name is not enclosed in quotes, the TSO data set prefix is added to the name. A PDS member name or relative GDG number may be appended to the data set name delimited by matched parentheses. If a data set name is not supplied for the ALLOCATE command, z/OS generates a temporary data set name. If DSN (*) is specified for the ALLOCATE command, the TSO terminal device is allocated. This is equivalent to specifying the TERM(TS) keyword.

DUMMY

(Optional) Allocates a null data set. This keyword is mutually exclusive with the DSNAME keyword.

MEMBER

(Optional) Provides a PDS member name or relative GDG number (0, +1,...). When this keyword is specified with a data set list, it applies only to the first data set name in the list.

```
ALLOC DSN('OPSMVS.CCLXLOAD(OPSDYNAM))...
```

is equivalent to:

```
ALLOC DSN('OPSMVS.CCLXLOAD') MEMBER (OPSDYNAM) ...
```

TERM(TS)

(Optional) Allocates the TSO terminal as an input or output data set. Allocation keywords other than DCB-related attributes are ignored.

SUBSYS

(Optional) Allocates a subsystem data set such as a CA OPS/MVS generic interface data set. The first parameter must be the 4-character subsystem name. One to six subsystem parameters may optionally be specified. The maximum length of a parameter is 67 characters.

REUSE

(Optional) Replaces any old allocation with a new allocation for a ddname. This is equivalent to issuing a FREE command for the old file, and then allocating the new file.

CONVERTIBLE(YES|NO)

Specifies whether an allocation can be modified and reused by a subsequent allocation.

YES

Assigns the convertible attribute to the new allocation.

NO

(Default) Does not convert the allocation by a subsequent allocation.

PERMANENT(YES|NO)

(Optional) Specifies whether the allocation can be deallocated to meet the control limit.

YES

(Default) The allocation will not be deallocated by the system to stay in the control limit of the number of dynamically allocated data sets specified by the DYNAMNBR parameter on the JCL EXEC statement. YES is the default for the ALLOCATE command.

NO

The system can deallocate this data set when it is not in use to meet the control limit.

ALLOWCONVERT

(Optional) Allows dynamic allocation to search for a current convertible allocation that is not in use and alter it to meet the new allocation criteria. This capability helps dynamic allocation to stay in the control limit.

GDGLOCATE

(Optional) Allows the GDG catalog entry to be reread prior to resolution of the relative generation number. When GDGLOCATE is not specified, and when a GDG is allocated with a relative generation number such as +1, subsequent allocations of the same generation number refer to the same data set. When GDGLOCATE is specified, if the +1 generation is now reallocated, a new data set is created since the prior +1 generation is now generation 0 in the catalog.

NEW|MOD|OLD|SHR

(Optional) Specifies the current status of the allocation and corresponds with the first operand of JCL DD statement DISP=keyword.

SYSOUT(class)

(Optional) Allocates a JES spool data set of the specified output class. SYSOUT is mutually exclusive with current status keywords.

SYSOUT(class,pgm,name)

The abbreviated form of SYSOUT(class) WRITER(pgm) FORM(name).

UNCATLG|CATLG|DELETE|KEEP

(Optional) Specifies the normal disposition of the data set when it is deallocated and corresponds to the second operand of the JCL DD statement DISP=keyword.

COND(UNCATLG|CATLG|DELETE|KEEP)

(Optional) Specifies the conditional disposition of the data set when an abnormal termination of the job step or address space occurs. This corresponds to the third operand of the JCL DD statement DISP=keyword.

FREE(CLOSE)

(Optional) Causes the data set to be deallocated when the program closes it. This eliminates the need to issue a FREE command to deallocate the data set.

SPACE(prime, secondary)

(Optional) Defines the primary and secondary space allocation amounts for DASD data sets. The units of allocation are specified by a separate keyword.

DIR(dirblocks)

(Optional) Determines the number of directory blocks for PDS data sets.

SPACE(prime, secondary, dir)

(Optional) Specifies an expanded form of SPACE that includes the number of directory blocks for PDS data sets.

TRACKS

(Optional) Defines units of allocation space.

CYLINDERS

(Optional) Specifies the units of allocation space.

BLOCK(*size*)

(Optional) Defines the units of allocation space. If the BLKSIZE keyword is not specified, then the block operand size is also the BLKSIZE value. This allocation type is device-independent.

AVBLOCK(*size*)

(Optional) Defines the units of allocation space. The BLKSIZE operand is not set by this keyword. When specified in an SMS environment, AVBLOCK can be used to allocate space by the number of logical records with an SMS determined block size. AVGREC should also be specified with this keyword.

AVGREC(U|K|M)

(Optional) Specifies the multiplier for space units when BLOCK, BLKSIZE, or AVBLOCK is used for device-independent space allocation.

- U-Multiply space units by 1
- K-Multiply space units by 1024
- M-Multiply space units by 1048576

RELEASE

(Optional) Releases the unused space of a new data set when a program closes it.

ROUND

(Optional) Specifies that space allocation on a DASD device should be rounded up to the next cylinder boundary for improved performance.

UNIT(*unitname*)

(Optional) Defines the generic or esoteric device name on which the data set is to be allocated.

UCOUNT(*unitcount*)

(Optional) For multi-volume data sets, defines the maximum number of devices to be allocated to the data set.

PARALLEL

(Optional) Indicates that one device should be allocated for each volume of a multi-volume data set. This keyword is mutually exclusive with the UCOUNT keyword.

VOLUME(*vol1*,...,*vol6*)

(Optional) Defines a list of one to six volume serial number names on which a new data set is to be placed or an old data set exists.

REFVOL(*dsname*)

(Optional) Allocates a new data set on the same volume as an existing cataloged data set. This keyword is mutually exclusive with the VOLUME keyword.

VSEQ(*volume sequence number*)

(Optional) Defines the maximum number of volumes that may be used by a data set.

POSITION(0-9999)

(Optional) Specifies the relative file number of the data set on multi-file tape volume. The maximum number is 9999.

PRIVATE

(Optional) Indicates that any mountable volumes that are not permanently resident or reserved should be marked with the private storage attribute.

DEFER

(Optional) Indicates that any mountable volumes used by a data set should not be physically mounted until they are referenced.

LABEL(NL|SL|NSL|SUL|BLP|LTM|AL|AUL)

(Optional) Specifies the type of label processing to be performed when the data set is opened.

PASSWORD

(Optional) Indicates that you must supply a password to acquire input and output access to the data set.

NOPWREAD

(Optional) Indicates that you must supply a password to acquire output access to the data set. Input access is permitted without a password.

DSPASSWORD(*password*)

(Optional) Defines the password required to access the data set. If not specified, the system operator must supply the password. SMS-managed data sets do not use password protection.

INPUT

(Optional) Indicates that the data set may be opened for input only. This keyword is equivalent to the JCL DD statement IN parameter of the LABEL operand.

OUTPUT

(Optional) Indicates that the data set may be opened for output only. This keyword is equivalent to the JCL DD statement OUT parameter of the LABEL operand and is mutually exclusive with the INPUT keyword.

RETPD(0-9999)

(Optional) Defines the number of days from the current date that must pass before the data set can be deleted or overwritten.

EXPDT(ccyyddd|yyddd)

(Optional) Defines the Julian date on which the data set may be deleted or overwritten. The long form that includes the century is recommended. This keyword is mutually exclusive with RETPD.

DEN(556|800|1600|6250)

(Optional) Indicates the recording density for magnetic tape data sets.

REFDSN(dsname)

(Optional) Defines the name of a catalogued DASD data set from which DCB attributes should be copied.

REFDD(ddname)

(Optional) Defines the ddname of a currently allocated data set from which DCB attributes should be copied. This keyword is mutually exclusive with REFDSN.

USING(attr)

(Optional) Defines the name of an attribute data set created by the TSO ATTRIB command for copying of DCB parameters. This is equivalent to using the REFDD keyword that points to a null data set allocation with DCB attributes.

DSORG(VSAM|PO|POU| DA|DAU|PS|PSU|IS|ISU)

(Optional) Specifies the data set organization.

RECFM(FB|VB|VBS|U|DA|FBA|VBA|FBM|VBM...)

(Optional) Specifies the record format of the data set. Unlike TSO ALLOCATE, there are no spaces between the record format designation characters.

BLKSIZE(0-32760)

(Optional) Defines the block size for the data set. The maximum block size is 32760. When allocation space units are not specified, it is also the average block size (AVBLOCK) value.

LRECL(0-32760)

(Optional) Defines the logical record length of the data set. The maximum length is 32760.

STORCLAS(storage class)

(Optional) Specifies the SMS storage class name.

MGMTCLAS(management class)

(Optional) Specifies the SMS management class name.

DATACLAS(*data class*)

(Optional) Defines the SMS data class for allocation.

LIKE(*dsname*)

(Optional) Defines a model data set name whose space and DCB attributes are to be copied as defaults when corresponding keywords are not explicitly specified.

DSNTYPE(LIBRARY|PDS)

(Optional) Specifies the type of partitioned data set to be allocated. LIBRARY indicates a PDSE.

KEYOFF(0-32760)

(Optional) Defines the offset of the key field in the record for a VSAM KSDS allocation. This function requires SMS.

KEYLEN(0-255)

(Optional) Defines the length of the KEY field for a VSAM KSDS or other direct access data set.

RECORD(KS|ES|RR|LS)

(Optional) Specifies the data set organization for a VSAM data set. Possible organizations:

- KS-Keyed sequence data set
- ES-Entry sequence data set
- RR-Relative record data set
- LS-Linear data set

RLS(NRI|CR)

(Optional) Allows record level sharing for sysplex VSAM files using the coupling facility with SMS.

- NRI-No read integrity
- CR-Consistent read integrity

SPIN(UNALLOC|NO)

(Optional) Specifies when the SYSOUT data set should be made available for printing.

- UNALLOC-Available when deallocated
- NO-Not available until step termination

HOLD

(Optional) Places the SYSOUT data set on the JES hold queue when deallocated.

DEST(*node|userid*)

(Optional) Specifies the name of a JES node or TSO user ID to which an output data set is to be routed.

WRITER(*pgm*)

(Optional) Defines the name of an external writer program that prints a SYSOUT data set instead of the JES subsystem. If you are submitting JCL to an internal reader, the value of *pgm* may be specified as INTRDR.

FORMS(*name*)

(Optional) Defines the name of the form that is mounted on the printer when printing a SYSOUT data set.

UCS(*charset*)

(Optional) Defines the universal character set (font) to be used when printing a SYSOUT data set.

FCB(*image*)

(Optional) Defines the forms control buffer to be used when printing a SYSOUT data set.

FOLD

(Optional) Specifies whether lower case characters are to be translated to upper case for printing a SYSOUT data set.

COPIES(0-255)

(Optional) Defines the number of copies of a SYSOUT data set to print. This number is subject to the installation limit. The maximum number of copies permitted is 255.

OUTLIM(0-16777215)

(Optional) Defines the number of output lines for a SYSOUT data set. The maximum number of lines permitted is 16777215.

UNALLOCATE

(Optional) Indicates that the data set is to be deallocated even if it is permanently allocated. This is the default for the OPSDYNAM FREE command.

REMOVE

(Optional) Indicates that only the in-use attribute is to be removed from the data set being freed, even if the data set is permanently allocated. This keyword is mutually exclusive with UNALLOCATE.

PATH(*pathname*)

(Optional) Specifies the 1- to 255-character fully qualified name of an HFS or zFS in compatibility mode file. Pathname must begin with a / and include all sub-directory names down to the filename.

PATHDISP(normal, abnormal)

(Optional)

normal

The normal disposition of the HFS or zFS in compatibility mode file when deallocated. Valid values are KEEP or DELETE.

abnormal

The conditional disposition of the HFS or zFS in compatibility mode file when deallocated due to an abnormal termination of the job step or address space. Valid values are KEEP or DELETE.

Default: PATHDISP(KEEP,KEEP)

FILEDATA(TEXT|BINARY)

(Optional)

BINARY

The HFS or zFS in compatibility mode file records are not delimited by the new line character X'15'. This is the default.

TEXT

The HFS or zFS in compatibility mode file records are delimited by the new line character.

PATHMODE(access1,access2,...access14)

(Optional) access nn = 1 to 14 file access specifications to set the read/write/execute access attributes of a new HFS or zFS in compatibility mode file. The valid values are:

SRUSR	SMUSR	SXUSR	SRWXU
SRGRP	SMGRP	SXGRP	SRWXG
SROTH	SMOTH	SXOTH	SRWXO
SSUID	SSGID		

The meanings of these values are documented in the section on the TSO ALLOCATE command.

PATHOPS(option1,option2,...option8)

(Optional) option n = 1 to 8 file access and status options that determine how the HFS or zFS in compatibility mode file is opened and processed.

One file option from the access group and up to seven file options from the status group may be specified from the following list:

Access Group:

- ORDONLY
- OWRONLY
- ORDWR

Status Group:

- OAPPEND
- OCREAT
- OEXCL
- ONOCTTY
- ONONBLOCK
- OSYNC
- OTRUNC

The meanings of these options are documented in the section describing the TSO `ALLOCATE` command.

EATTR(OPT|NO)

(Optional) Specifies whether the data set can support extended attributes. By definition, a data set with extended attributes can reside in EAS (extended address space) on an EAV (extended address volume). These attributes can be specified for non-VSAM data sets as well as for VSAM data sets.

OPT

Extended attributes are optional, and will be created if the data set is created on an EAV. In addition, the data set can be created in the EAS of the EAV.

NO

No Extended Attributes allowed.

SMSHONOR

(Optional) Specifies that the system must honor the device name or the esoteric specified on the unit name for an SMS tape library request.

Example: OPSDYNAM Allocate

The following allocates a new text file:

```
Address OPSDYNAM "Allocate ddname(CRHFS)
Path('/u/users/ops/mvs/rexxpgm1')",
"Pathopts(ORDWR,OCREAT)",
"Pathmode(SIRWXU,SIRWXS,SIXOTH) Filedata(TEXT)"
```

Common Keywords for `ALLOCATE`

The following common keywords may be specified for the `ALLOCATE` command:

DELAY, CMDRESP, and MSG

You may specify the `DELAY`, `CMDRESP`, and `MSG` keywords described in *Keywords Common to All ADDRESS OPSDYNAM Commands* in this chapter.

Examples: ALLOCATE keyword

- Example 1: Allocate an existing data set:

```
/* Allocate an existing data set */
"Allocate DD(oldrule) DSN('sys1.old.rules') SHR Reuse"
if rc > 0 Then Signal DYNERROR
Return rc
```

```
DYNERROR
Say Sourceline(sigl-1)
Say "OPSDYNAM error RC='rc' "Error code='opsercd,
    "Info code='opsifcd
Do While Queued() > 0
Full errsng
Say errsng
End
Return rc
```

- Example 2: Allocate a new rules partitioned data set:

```
/* Allocate a new rules pds data set */
"Allocate DD(newrule) DSN('sys1.new.rules') New Catlg",
"Like('opsdev.o.rules') Recfm(FB) Block(3120) Space(100,100,40)"
if rc > 0 Then Signal DYNERROR
Return rc
```

```
DYNERROR
Say Sourceline(sigl-1)
Say "OPSDYNAM error RC='rc' "Error code='opsercd,
    "Info code='opsifcd
Do While Queued() > 0
Full errsng
Say errsng
End
Return rc
```

OPSDYNAM CONCAT—Concatenate DDNames

The CONCAT command either permanently or temporarily concatenates a list of ddnames to the first ddname specified. If you specify PERMANENT(YES), the concatenation is permanent and may not be deconcatenated. The default is NO.

This command has the following format:

```
ADDRESS OPSDYNAM "CONCAT keywords"  
[FILE|DDNAME(ddname1,ddname2,...ddname65)]  
[PERMANENT(YES|NO)]  
[DELAY]  
[CMDRESP(type)]  
[MSG(ON|OFF)]
```

FILE|DDNAME(*ddname1*, *ddname2*,...*ddname65*)

(Optional) Defines a list of currently allocated ddnames that are to be concatenated to the first ddname. A maximum of 65 ddnames may be specified.

PERMANENT(YES|NO)

(Optional) Specifies whether the allocation will be deallocated to meet the control limit.

YES

(Default) Specifies the allocation will not be deallocated by the system to stay in the control limit of the number of dynamically allocated data sets specified by the DYNAMNBR parameter on the JCL EXEC statement.

NO

Specifies the system deallocates this data set when it is not in use to meet the control limit.

The following common keywords may be specified for the CONCAT command:

DELAY, CMDRESP, and MSG

(Optional) You may specify the DELAY, CMDRESP, and MSG keywords described in Keywords Common to All ADDRESS OPSDYNAM Commands in this chapter.

Example: OPSDYNAM CONCAT

The following temporarily concatenates data sets to the ddname of an old rule:

```
/* Temporarily concatenate the data sets to the oldrule ddname */
"Concat DD(oldrule newrule)"
If rc > 0 Then Signal DYNERROR
Return rc
```

```
DYNERROR
Say Sourceline(sig|-1)
Say "OPSDYNAM error RC='rc' "Error code='opsercd,
    "Info code='opsifcd
Do While Queued() > 0
Pull errsmsg
Say errsmsg
End
Return rc
```

OPSDYNAM DECONCAT—Deconcatenate DDNames

The DECONCAT command deconcatenates a non-permanent, existing concatenation (by ddname) back to the originally allocated ddnames of the concatenated data sets.

This command has the following format:

```
ADDRESS OPSDYNAM "DECONCAT keywords"
[FILE|DDNAME(ddname)]
[DELAY]
[CMDRESP(type)]
[MSG(ON|OFF)]
```

FILE|DDNAME(*ddname*)

(Optional) Specifies the ddname of an existing concatenation of data sets with the non-permanent attribute.

The following common keywords may be specified:

DELAY, CMDRESP, and MSG

(Optional) You may specify the DELAY, CMDRESP, and MSG keywords described in Keywords Common to All ADDRESS OPSDYNAM Commands in this chapter.

Example: OPSDYNAM DECONCAT

The following removes concatenation and restores the original ddnames:

```
/* Remove the concatenation and restore the original ddnames */
"Deconcat DD(oldrule)"
If rc > 0 then Signal DYNERROR
Return rc

DYNERROR
Say Sourceline(sig|-1)
Say "OPSDYNAM error RC='rc' "Error code='opsercd,
    "Info code='opsifcd
Do While Queued() > 0
Pull errsmsg
Say errsmsg
End
Return rc
```

OPSDYNAM FREE—Free DD and Data Set Names

The FREE command frees a list of ddnames, data set names, or both. It provides the option to remove only the in-use attribute, or to deallocate a permanently allocated data set. The default is to deallocate a permanently allocated data set.

This command has the following format:

```
ADDRESS OPSDYNAM "FREE keywords"
[FILE|DDNAME(ddname1,ddname2,...ddname65)]
[DATASET|DSNAME(dsname1,dsname2,... dsname16)]
[UNALLOCATE|REMOVE]
[DELAY]
[CMDRESP(type)]
[MSG(ON|OFF)]
[PATH(pathname)]
```

FILE|DDNAME (*ddname1, ddname2... ddname65*)

(Optional) Defines a valid z/OS ddname or list of ddnames to be freed. A maximum of 65 ddnames may be specified.

DATASET|DSNAME (*dsname1, dsname2, ...dsname16*)

(Optional) Defines a single z/OS data set name or a list of data set names to be freed. A maximum of 16 data set names may be specified. If the data set name is not enclosed in quotes, then the TSO data set prefix is added to the name. A PDS member name or relative GDG number may be appended to the data set name delimited by matched parentheses.

UNALLOCATE

(Optional) Indicates that the data set is to be deallocated even if it is permanently allocated. This is the default for the FREE command.

REMOVE

(Optional) Indicates that only the in-use attribute is to be removed from the data set being freed, even if the data set is permanently allocated. This keyword is mutually exclusive with the UNALLOCATE keyword.

PATH(*pathname*)

(Optional) Defines the 1-255 character fully qualified name of an HFS or zFS in compatibility mode file. Pathname must begin with a / and include all sub-directory names down to the filename.

The following common keywords may be specified:

DELAY, CMDRESP, and MSG

(Optional) You may specify the DELAY, CMDRESP, and MSG keywords described in Keywords Common to All ADDRESS OPSDYNAM Commands in this chapter.

Example: OPSDYNAM FREE

The following frees and unallocates files:

```
/* Free and unallocate the files */
"Free DD(oldrule,newrule)"
If rc > 0 Then Signal DYNERROR
Return rc

DYNERROR:
Say Sourceline(sig|-1)
Say "OPSDYNAM error RC='rc' Error code='opsercd,"
    "Info code='opsifcd"
Do While Queued() > 0
Pull errsmsg
Say errsmsg
End
Return rc
```

OPSDYNAM INFO—Display Data Set Information

The INFO command returns the data set name, ddname, and data set organization of a currently allocated data set or ddname.

This command has the following format:

```
ADDRESS OPSDYNAM "INFO keywords"  
[FILE|DDNAME(ddname) OR DATASET|DSNAME(dsname)]  
[DELAY]  
[CMDRESP(type)]  
[MSG(ON|OFF)]  
[PATH(pathname)]
```

FILE|DDNAME (*ddname*)

(Optional) Defines a currently allocated z/OS ddname for which information should be returned. If the ddname points to a concatenated list of data sets, the INFO function only returns information on the first data set in the concatenation.

DATASET|DSNAME(*dsname*)

(Optional) Defines the name of a currently allocated data set for which information should be returned. This keyword is mutually exclusive with the DDNAME keyword. If the data set name is not enclosed in quotes, then the TSO data set prefix is added to the name. A PDS member name or relative GDG number may be appended to the data set name delimited by matched parentheses.

PATH(*pathname*)

(Optional) Defines the 1- to 255-character fully qualified name of an HFS or zFS in compatibility mode file. Pathname must begin with a / and include all sub-directory names down to the filename.

The following common keywords may be specified:

DELAY, CMDRESP, and MSG

(Optional) You may specify the DELAY, CMDRESP, and MSG keywords described in Keywords Common to All ADDRESS OPSDYNAM Commands in this chapter.

Example: OPSDYNAM INFO

The following displays allocation information:

```
/* Display information about the allocation */
"Info DD(aldrule)"
If rc > 0 The Signal DYNERROR
Say "DDname" opsddl "allocated to" opsdsn "is dsorg" opsdsorg
Return rc

DYNERROR
Say Sourceline(sigl-1)
Say "OPSDYNAM error RC='rc' "Error code='opsercd,
    "Info code='opsifcd
Do While Queued() > 0
Pull errsng
Say errsng
End
Return rc
```

OPSDYNAM Command Return Codes

The following return codes are produced:

0

Command executed with no errors. The information code variable may be greater than zero, indicating that a non-critical error was encountered.

4

Command did not execute successfully. The error code indicates the nature of the error and a DAIRFAIL message is generated if MSG(ON) is active.

8

Dynamic allocation installation exit denied the request.

12

Dynamic allocation parameter list is invalid. Some keyword combinations permitted by OPSDYNAM may not be acceptable to dynamic allocation.

40

Command syntax or data set name invalid.

Keywords Common to All ADDRESS OPSDYNAM Commands

The following optional keywords are shared by OPSDYNAM host commands:

CMDRESP(*type*)

Defines the type of response desired from the request. Possible values are:

- **TERMINAL**-Issues any error message to the TSO terminal.
- **NOWHERE**-Does not issue any execution error messages. Syntax error messages are still issued.
- **XDQ**-Puts the messages in the REXX external data queue
- **CLIST/REXX**-Sets the following result variables:
 - OPSIFCD = SVC99 INFO CODE
 - OPSERCD = SVC99 ERROR CODE
 - OPSMSCD = SMS reason code
 - OPSDSN = DSNAME (INFO request)
 - OPSDSORG = DSORG (INFO request)
 - OPSVOL = VOLUME (ALLOC/INFO request)
 - OPSMEMBER = Member name (INFO request)
 - OPSDISP = Data set status (INFO request)
 - OPSSTORCLAS = Storage class (INFO request)
 - OPSMGMTCLAS = Management class (INFO request)
 - OPSDATACLAS = Data class (INFO request)
 - OPSAVGREC = Allocation unit (INFO request)
 - OPSDSNTYPE = Data set type (INFO request)
 - OPSPATHOPTS = z/OS UNIX file options (INFO request)
 - OPSPATHMODE = z/OS UNIX file access attributes (INFO request)
 - OPSFILEDATA = z/OS UNIX file organization (INFO request)

Default: REXX

DELAY(*seconds*)

Defines the number of seconds to wait before issuing the OPSDYNAM request. The maximum time for delay is 300 seconds.

Default: 0.

MSG(ON|OFF)

Indicates whether dynamic allocation failure messages (DAIRFAIL) are to be generated. These messages indicate the reason for the failure. Possible values are:

- ON-Generates failure messages. The default for CMDRESP(TERMINAL|XDQ) is ON.
- OFF-Does not generate failure messages. The default for CMDRESP(NOWHERE|REXX|CLIST) is OFF.

Example: Common Keywords

The following shows the common ADDRESS OPSDYNAM keywords:

```
address OPSDYNAM
/*Allocate an existing data set */
"Allocate DD(oldrule) DSN('sys1.old.rules') Shr Reuse"
if RC > 0 then signal DYNERROR
/* Display information about the allocation */
"Info DD(oldrule)"
if RC > 0 then signal DYNERROR
say "DDname" opsdld "allocated to "opsdsn "is dsorg" opsdsorg
/* Allocate a new rules pds data set */
"Allocate DD(newrule) DSN('sys1.new.rules') New Catlg" ,
"Like('opsdev.o.rules') Recfm(FB) Block(3120) Space(100,100,40)"
if RC > 0 then signal DYNERROR
say "New rule data set allocated on" opsvol
/* Temporarily concatenate the data sets to the oldrule DD name */
"Concat DD(oldrule newrule)"
if RC > 0 then signal DYNERROR
/* Remove the concatenation and restore the original DD names */
"Deconcat DD(oldrule)"
if RC > 0 then signal DYNERROR
/* Free and unallocate the files */
"Free DD(oldrule,newrule)"
if RC > 0 then signal DYNERROR
return 0

DYNERROR:
say SOURCELINE(sigl-1)
say "OPSDYNAM error RC="rc "Error code="opsercd "Info code="opsifcd ,
"SMS reason code="opsmrscd
do while QUEUED() > 0
  pull errmsg
  say errmsg
end
return RC
```

ADDRESS OSF—Route Commands to TSO Server

The ADDRESS OSF instruction routes commands to CA OPS/MVS TSO server address spaces. Output from TSO commands directed to the OSF host environment is not returned to rules or the OPS/REXX program. Host commands sent to a server through ADDRESS OSF cannot contain more than 6144 bytes. This includes the command text and any additional parameters.

This command has the following format:

```
ADDRESS OSF "OSF command"
```

To schedule an OPS/REXX EXEC to run asynchronously in an OSF TSO server address space, use code similar to the following:

```
ADDRESS OSF  
"OI servprog argument"
```

ADDRESS OSF Return Codes

The ADDRESS OSF instruction produces the return codes listed in the following table.

Note: The ADDRESS OSF return codes indicate whether a command was successfully sent to the OSF execute queue. They do not imply that a command executed successfully in a server.

0

The OSF command was successfully added to the OSF execute queue.

4

The OSF execute queue has overflowed; the command cannot be queued.

20

Either of the following:

- The product is not active. See message OPS4340S.
- The OSF facility is not active.

32

The authorization exit or a security rule rejected the OSF command. See message OPS4346E.

36

An abend occurred in the user security exit. See message OPS4347S.

44

The OSF command is too long. See message OPS4342E.

ADDRESS OSFTSL—Route Commands to TSL Server

The ADDRESS OSFTSL instruction routes commands to CA OPS/MVS TSL server address spaces. Output from TSO commands directed to the OSF TSL host environment is not returned to rules or the OPS/REXX program. Host commands sent to a server through ADDRESS OSFTSL cannot contain more than 6144 bytes. This includes the command text and any additional parameters.

This command has the following format:

```
ADDRESS OSFTSL "TSO command"
```

To schedule an OPS/REXX EXEC to run asynchronously in an OSF TSL server address space, use code similar to the following:

```
ADDRESS OSFTSL  
"OI servprog argument"
```

ADDRESS OSFTSL Return Codes

The ADDRESS OSFTSL return codes indicate whether a command was successfully sent to the OSFTSL execute queue. They do not imply that a command executed successfully in a server.

The ADDRESS OSFTSL instruction produces the following return codes:

0

The OSFTSL command was successfully added to the OSFTSL execute queue.

1

The OSFTSL command was successfully added to the OSFTSL execute queue. However, the OSFTSLMIN and OSFTSLMAX parameters are both set to zero. Unless these parameters are changed, the command will never be processed and if additional commands are sent, the queue will likely overflow.

4

The OSFTSL execute queue has overflowed. The command cannot be queued.

20

Either one of the following is true:

- The product is not active. See message OPS4340S.
- The OSF facility is not active.

32

The authorization exit or a security rule rejected the OSFTSL command. See message OPS4346E.

36

An abend occurred in the user security exit. See message OPS4347S.

44

The OSFTSL command is too long. See message OPS4342E.

ADDRESS OSFTSP—Route Commands to TSP Server

The ADDRESS OSFTSP instruction routes commands to CA OPS/MVS TSP server address spaces. Output from TSO commands directed to the OSF TSP host environment is not returned to rules or the OPS/REXX program. Host commands sent to a server through ADDRESS OSFTSP cannot contain more than 6144 bytes. This includes the command text and any additional parameters.

This command has the following format:

```
ADDRESS OSFTSP "TSO command"
```

To schedule an OPS/REXX EXEC to run asynchronously in an OSF TSP server address space, use code similar to the following:

```
ADDRESS OSFTSP  
"OI servprog argument"
```

ADDRESS OSFTSP Return Codes

The ADDRESS OSFTSP instruction produces the return codes listed below.

Note: The ADDRESS OSFTSP return codes indicate whether a command was successfully sent to the OSFTSP execute queue. They do not imply that a command executed successfully in a server.

0

The OSFTSP command was successfully added to the OSFTSP execute queue.

1

The OSFTSP command was successfully added to the OSFTSP execute queue. However, the OSFTSPMIN and OSFTSPMAX parameters are both set to zero. Unless these parameters are changed, the command will never be processed, and if additional commands are sent, the queue will likely overflow at some future time.

4

The OSFTSP execute queue has overflowed. The command cannot be queued.

20

Either one of the following is true:

- The product is not active. See message OPS4340S.
- The OSF facility is not active.

32

The authorization exit or a security rule rejected the OSFTSP command. See message OPS4346E.

36

An abend occurred in the user security exit. See message OPS4347S.

44

The OSFTSP command is too long. See message OPS4342E.

ADDRESS SERVDESK—Create Service Desk Requests

The ADDRESS SERVDESK host environment lets you create a request on CA Service Desk and is available from OPS/REXX programs or AOF rules. The ADDRESS SERVDESK host environment uses CAISDI/med server and CAISDI/soap server to create the request.

Note: For more information, see the appendix “CCS for z/OS Component Requirements” in the *CA OPS/MVS Administration Guide*.

This command has the following format:

```
ADDRESS SERVDESK "keywords"
```

Keywords for ADDRESS SERVDESK:

REQUEST(CREATE)

Creates a request.

REQUEST(CREATE)

Note: CREATE is the only keycode available.

SUMMARY('text')

Defines the summary text to be displayed on CA Service Desk. This field is visible in the list of all requests.

Limit: 240 characters

SUMMARY('text')

DESCRIPTION('text')

Defines the description to be displayed on CA Service Desk. This field is visible in details of request.

Limit: 4000 characters

DESCRIPTION('text')

text

This parameter is translated to XML so that symbols, such as & and <, are replaced with an appropriate XML entity. For example, the ampersand (&) is translated to &#amp; and < is translated to <. Therefore, allow for the extra bytes.

Example: An XML translation from symbols to text

This example shows 14 bytes being sent but address SERVDESK converted the ampersand character to &#amp;. Therefore 18 bytes are sent to CA Service Desk instead of 14.

'Test & Results' converts to 'Test &#amp; Results'

Specify text normally, the address SERVDESK does the translation.

WAIT(n)

(Optional) Defines the wait time in seconds to create requests, either synchronous or asynchronous. If omitted, the AOF rule default value of 0 is set, and OPS/REXX default value of 60 seconds is set.

0

An Asynchronous request, valid value in the AOF rule only.

10

Minimum value, valid in OPS/REXX only.

3600.

Maximum value, valid in OPS/REXX only

9999

Special value, unlimited timeout to create a synchronous request, valid in OPS/REXX only.

To set the timeout for 100 seconds to create a synchronous request, enter WAIT(100).

DEBUG(YES|NO)

(Optional) Specifies whether to print debug information to OPSLOG.

YES

Prints debug information to OPSLOG.

NO

Does not print debug information to OPSLOG.

Default: NO

OPS/REXX Output Variables

Four OPS/REXX variables are created as output. These variables are accessible from OPS/REXX and AOF rule.

SERVDESK.RETURN

Contains the return code. In some cases, the reason code can contain more information.

SERVDESK.REASON

Contains more information as to the reason for the return code.

SERVDESK.NUMBER

Contains the number of created requests.

SERVDESK.HANDLE

Specifies how to handle the requests from ADDRESS SERVDESK.

ADDRESS SERVDESK Formats

The following examples show different REQUEST(CREATE) formats.

- Example 1: OPS/REXX code to create a service desk request

The following OPS/REXX code produces a detail screen of the create request. All requests are created as priority two.

```
address SERVDESK "REQUEST(CREATE) SUMMARY('Testing request')",
"DESCRIPTION('This is the test of address SERVDESK') WAIT(30) DEBUG(YES)"
if rc = 0 then say "Request was created: #"&SERVDESK.NUMBER
```

Produces the following detail screen:

1942 Request Detail

Affected End User: [System_SD_User](#) Request Area: Status: Open Priority: 2

Detail

Reported By	Assignee	Group	Configuration Item
System_SD_User	System_SD_User		
Severity	Urgency	Impact	Active?
		None	YES
Change	Charge Back ID	Call Back Date/Time	Root Cause

Summary Information

Summary	Total Activity Time
Testing request	00:00:00
Description	
This is the test of address SERVDESK	
Open Date/Time	Last Modified
09/12/2008 01:33 pm	09/12/2008 01:33 pm
Resolve Date/Time	Close Date/Time

Request Activity Log List

Created By / Description	On *	Time Spent	Type
System_SD_User	09/12/2008 01:33 pm		In Progress
create a new request/incident/problem/change/issue			

1 request activity log found

- Example 2: OPS/REXX code

```
address SERVDESK "REQUEST(CREATE) SUMMARY('Test 2')
DESCRIPTION('Test number two.')
```

Produces the following detail screen:

1944 Request Detail - Unicenter Service Desk - Microsoft Internet Explorer provided by CA

Unicenter® Service Desk

Logged in as: [ServiceDesk](#) (Log Out)

File View Activities Actions Search Reports Window Help

1944 Request Detail

Request Area: [System_SD_User](#) Status: Open Priority: 2

Detail

Reported By	Assignee	Group	Configuration Item
System_SD_User	System_SD_User		
Severity	Urgency	Impact	Active?
		None	YES
Change	Change Back ID	Call Back Date/Time	Root Cause

Summary Information

Summary	Total Activity Time
Test 2	00:00:00
Description	
Test number two.	
Open Date/Time	Last Modified
09/12/2008 02:30 pm	09/12/2008 02:30 pm
Resolve Date/Time	Close Date/Time

Request Activity Log List

Created By / Description	On	Time Spent	Type
System_SD_User	09/12/2008 02:30 pm		Initial
create a new request/incident/problem/change/issue			

1 request activity log found

- Example 3a: AOF Rule code with Wait set to 0.

```
)CMD SERVRULE
)NT
)PROC
address SERVDESK "REQUEST(CREATE) SUMMARY('AOF Rule Test') WAIT(0)",
DESCRIPTION('AOF Rule Test with WAIT(0) parameter.')
```

```
)TERM
)END
```

■ Example 3b: OPS/REXX code to run the rule

```
address AOF "SUBSYS OPSX"
address AOF "ENABLE RULES.SERVRULE"
address OPER "C(SERVRULE)"
address AOF "DISABLE RULES.SERVRULE"
```

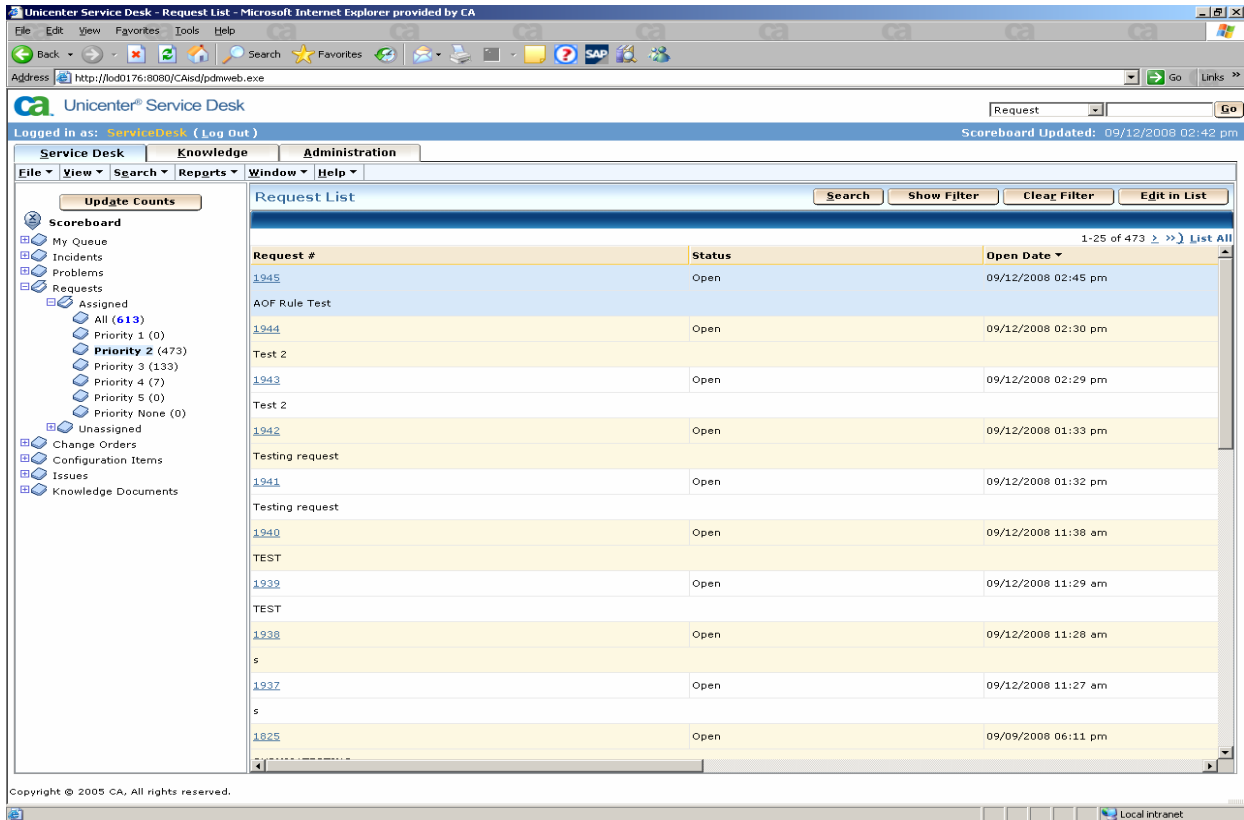
Produces the following detail screen:

The screenshot displays the Unicenter Service Desk interface in a Microsoft Internet Explorer browser window. The page title is "1945 Request Detail - Unicenter Service Desk - Microsoft Internet Explorer provided by CA". The interface includes a navigation bar with menus like File, View, Activities, Actions, Search, Reports, Window, and Help. Below the navigation bar, there are buttons for Edit, Create Change Order, Create Incident, and Profile Browser. The main content area is titled "1945 Request Detail" and contains several sections:

- Affected End User:** System_SD_User
- Request Area:** Open
- Status:** Open
- Priority:** 2
- Detail Section:**
 - Reported By:** System_SD_User
 - Assignee:** System_SD_User
 - Group:** System_SD_User
 - Configuration Item:** System_SD_User
 - Severity:** System_SD_User
 - Urgency:** System_SD_User
 - Impact:** System_SD_User
 - Active?:** YES
 - Change:** System_SD_User
 - Charge Back ID:** System_SD_User
 - Call Back Date/Time:** System_SD_User
 - Root Cause:** System_SD_User
- Summary Information:**
 - Summary:** AOF Rule Test
 - Description:** AOF Rule Test with WAIT(0) parameter.
 - Open Date/Time:** 09/12/2008 02:45 pm
 - Last Modified:** 09/12/2008 02:45 pm
 - Resolve Date/Time:** 09/12/2008 02:45 pm
 - Close Date/Time:** 09/12/2008 02:45 pm
- Request Activity Log List:**
 - Created By / Description:** System_SD_User
 - On:** 09/12/2008 02:45 pm
 - Time Spent:** 00:00:00
 - Type:** Initial

Example: Request List

The following display shows the list of service desk requests:



Command Return and Reason Codes

The following codes are common to most functions. In addition to the actual return code, the reason codes can also be examined.

0

Indicates the command executed successfully and terminated properly.

1

Indicates an error occurred during a parsing scan. The SERVDESK.REASON normally contains a zero, but may contain one of the following specific codes to indicate the cause of certain errors:

- 1 - Indicates an error occurred during a parsing scan.
- 2 - Indicates an error occurred during a scan.

2

WAIT(n) in the rule, n>0

3

Indicates an error occurred during the DESC XML translation.

The SERVDESK.REASON normally contains a zero, but may contain one of the following specific codes to indicate the cause:

- 1 - Encountered a DESCription XML translation error.
- 2 - Encountered a SUMMary XML translation error.
- 3 - Indicates the DESCription contains only blanks.
- 4 - Indicates the SUMMary contains only blanks.

4

Indicates that there was an error in processing the API request.

- 4 - Found no matching API request for CANCEL.
- 8 - The request was not processed because it timed out.
- 12 - Could not obtain the virtual storage needed to process the request.
- 16 - Detected an unauthorized use of the CAISDI/med interface.
- 20 - Encountered an unexpected Name/Token service routine error.
- 24 - CAISDI/med address space is not active.
- 28 - Encountered a permanent CAISDI/soap server error while processing the request.
- 32 - Detected an invalid API parameter list value.
- 36 - An unexpected ABEND occurred.
- 40 - An unexpected error occurred while retrieving data.
- 44 - An unexpected error occurred while processing a CANCEL request.
- 48 - An unexpected error occurred while processing a PURGE request.
- 52 - The product value is not defined to the CAISDI/med interface.
- 56 - The CAISDI/med did not have a defined server to process the request.
- 60 - The caller canceled the request.

5

Indicates a variable creation error.

6

Indicates an abend occurred

ADDRESS SOF—Manage Devices

The ADDRESS SOF command is issued internally within the Switch Operations Facility (SOF) ISPF Interface. Communication between the ADDRESS SOF host command and the CA SOF server uses the CA CCI interface.

The ADDRESS SOF host environment interfaces with the IBM I/O subsystem to provide advanced management capabilities for devices that are attached through the ESCON directors and FICON switches.

For detailed interface information, see the IBM manual SA23-0356, titled *ESCON Director Programming Interface*.

The ADDRESS SOF host environment also provides an interface to Peer-to-Peer Remote Copy commands through its PPRCCMD request type. With this set of commands, you can set up and can manage the PPRC option of remote copy, which is based on disaster recovery needs. This option includes creating and deleting paths and volume pairs or querying existing setup of the DR PPRC environment.

The command has the following format:

```
"request 'specific keywords' common keywords"
```

Execution Considerations

The ADDRESS SOF command cannot be executed either as a rule or in cross-memory access mode. An authorization test requires that a security rule must be active. The rule must comply with code applicable to OPS RULES, and should be subtype OPSOF.

The command has the following format:

```
)SEC SOF
```

Common Keywords of ADDRESS SOF

These keyword variables are common to all ADDRESS SOF functions. Function-specific variables are described separately in their corresponding sections.

SERVER(ccisysname|*,cciapplname)

Defines the SERVER name consisting of a CCI system name, followed by a CCI application name. The system name can be one to eight characters, or wildcards, where permitted. The CCI application name identifies the SOF server to which the request is directed. Enter the name as a character string of up to 20 characters.

Defaults: ccisysname='*' and cciapplname='SOF*'

STEM(string)

Defines the first qualifier of the stem variable name generated for all REXX variables produced by the ADDRESS SOF command. Enter the STEM keyword as a 1- to 64-character string. A trailing period is not necessary.

Default: OPSOF

SYSWAIT(seconds)

Defines the maximum wait time as a digit between 10 and 300 for the function to complete. The function will complete with a timeout error if the SYSWAIT time is exceeded.

Default: 30

DEBUG(YES|VAR|CB|ALL|NO)

Specifies the trace the status and display data to OPSLOG.

YES

Trace progress only.

VAR

Trace progress and display REXX variables.

CB

Trace progress and display control blocks.

ALL

Trace and display all of the above.

NO

Disable tracing and displays.

Default: NO

Common REXX Variables

Several common variables are returned by multiple functions. Variables unique to each function are described in their appropriate sections.

OPSOFRTN

Contains the return code, and can be examined to determine the results of command execution. In some cases, the reason code may contain additional information. The value is equivalent to the REXX return code RC.

OPSOFRSN

Contains reason code information triggered by some conditions in addition to the return code. The return code will indicate when the reason code may be examined.

stem.0

Contains the count of all following REXX variables names stem.1...stem.x. The stem.0 variable is created for every function. If no variables are created, stem.0 will contain 0.

Example: OPSOF.0 = 4

stemEMSG

Returns a 72-byte text string after each function call. When the result is successful, the text string contains all blanks; otherwise, it will contain the error description.

Note: In spite of the leading stem, this is a simple variable name without multiple qualifiers.

Example: OPSOFEMSG

Sample Rules and Programs

Sample rules and programs are supplied for your reference to help you use specific SOF functions.

Note: For samples, see the &HLQ.CCLXRULS and &HLQ.CCLXSAMP data sets and the appendixes in the *CA OPS/MVS User Guide*.

Return and Reason Codes

The following return codes are common to most functions.

0

Command was successful and terminated properly.

1

SOF completion error. The reason code contains the failing SOF return code.

- 2**
Abend occurred in the CCI API. The reason code contains the abend code.
- 3**
Abend occurred in the ADDRESS SOF code. The reason code contains the abend code.
- 4**
REXX variable access failure. The reason code contains the GREXX_VAR return code.
- 5**
REXX variable creation failure. The reason code contains the GREXX_VAR return code.
- 6**
Unsupported response type received from the SOF server.
- 7**
ADDRESS SOF is running in cross-memory mode.
- 8**
CCI RECEIVE error. Error notification may be immediate or delayed until after a WAIT ECB is posted, in which case the reason code will contain the ECB posting X'41' (decimal 65). If the notification is immediate, the reason code will contain the original CCI failing return code.
- 9**
CCI SEND error. The reason code contains the failing CCI return code.
- 10**
Omitted SERVER keyword, the SOF server cannot be identified.
- 11**
Command length error.
- 12**
Command syntax error.
- 13**
Parsing scan error. Check the External Data Queue for additional descriptor messages.
- 14**
CCI initialization failure. The reason code contains the CCI return code.
- 15**
Authorization failure. Reason code contains failing OPAUCK return code.

Special Character Handling Considerations

Certain long descriptor strings, typically 24-30 bytes, can contain any mix of characters, including imbedded blanks. Since the REXX word structure is based on blank separators, maintaining the original image requires two translation steps. The first translates all imbedded blanks to X'FE' characters when returned in a REXX variable. The next step is a user effort to reverse that back to blanks for display purposes or similar reasons. Please note that as long as REXX word boundaries need to be maintained, the second translation step should not be done.

The following fields are subject to the dual translation:

- Device ID
- Switch file name
- Port name
- Descriptor name

The following REXX function restores imbedded blanks:

```
NORMAL_FIELD = TRANSLATE(INCOMING_FIELD, 'FEX')
```

Device ID String Composition

A device ID is a unique system generated identifier for each device in a system. The device ID consists of the concatenation of the following six fields and may contain imbedded blanks:

Device type

Contains six characters (Example: 009020)

Device model number

Contains three characters (Example: T01)

Device manufacturer

Contains three characters (Example: IBM)

Device manufacturer plant code

Contains two characters (Example: 02)

Device serial number

Contains 12 characters (Example: 000012345678)

Device tag

Contains a binary half-word integer (Example: 001F)

The final device ID is the concatenation all components. The fields follow each other with no separating blanks, although any individual component may contain imbedded blanks.

Example: Sample device ID

type|model|manufacturer|plant|serial|tag

Q-PID(002094S28IBM020000000508512080)

Device ID input

ADDRESS SOF converts the tag portion of device ID operands from 4-byte printable hex back to 2-byte binary before sending them as input operands to the SOF commands.

Device ID output

The Device ID field is subject to the blank translation as described above. In addition, the Device ID tag field is also converted from a 2-byte hex value to its 4-byte decimal character equivalent. Individual components of the Device ID can be broken down using positional parsing as shown in the following example.

Example: Sample device ID parse:

Parse Var devid devtype 7 devmodel 10 devmfg 13 devplant 15 devserial
27 devtag

Say 'Device Plant is' Translate(devplant, ' ', 'FE')

Non-QUERY SOF Request Types

This section provides all ADDRESS SOF request types, except the QUERY request, and keywords.

COMMAND—Issue a Command to the SOF Server

Issues a command directly to the SOF server. The text is enclosed in quotes, may contain blanks, and contains any permissible command string. Command responses are returned as one or more server messages.

The SOF operator command has the following command format:

COMMAND 'text string to SOF server'

Created variables:

- Output stem variable records-Consists of one message response line per variable
- Output buffer-Consists of message responses lines delimited by the new line character

Message response: 0-128 characters

stemEMSG: 1-72 character description of outcome

DELETE FILES SWITCH—Delete Switch Configuration File

The DELETE FILES SWITCH command deletes a specific switch configuration file.

This command has the following format:

DELETE FILES SWITCH(*device*) **FILENAME**(*filename*)

Returns one stemEMSG variable containing a 1-72 character description of outcome.

FIND SOF—Find Active SOF Servers

The FIND SOF command finds active SOF servers.

This command has the following command format:

FIND SOF [OPTION(CCIplex)] **SERVER**(CCIsysname,CCIapplname)

FIND

Allows wildcarding to the SERVER definition. The FIND request is directed to the CCI API, not the SOF server.

OPTION(CCIplex)

(Optional keyword) Indicates that the CCI system name in the SERVER keyword is actually a CCI sysplex name. Only servers belonging to this CCI sysplex will be eligible for selection.

The optional SERVER keyword specifies a specific or generic CCI system name and an SOF server CCI application name. The full or generic application name that is specified should match the *cciapplname* parameter of an SOF server.

If the SERVER keyword or any arguments are omitted, the default values of *ccisysname*='*' and *cciapplname*='OPS*' are used.

Created stem variables:

Important! Due to concurrent parallel threads within the CCI server, the output of the FIND can result in duplicate server names.

The output stem variable records have the following word delimited fields for each matching CCI application name found:

- CCI system name - Contains one to eight characters
- CCI application name - Contains 1 to 20 characters
- L/R - Indicates L=on local system and R=on remote system
- Receive status - Indicates ACTIVE or INACTIVE
- CCI release level - Contains small integer 0-255
- CCI sysplex name - Contains one to eight characters or N/A if not in CCI sysplex
- Jobname - Contains one to eight characters
- Jobid - Contains eight characters
- Receive queuing on - Contains Y/N
- Number of messages queued - Contains integer
- Number of sends to this application - Contains integer
- Number of receives from this application - Contains integer
- Specific sender CCI system name - Contains eight characters or N/A
- Specific sender CCI application name - Contains 1-20 characters or N/A

LOG—Issue SOF Server Log Message

The LOG command issues a message to the SOF server log. A timestamp is prefixed to the message text by the SOF server unless NOPREFIX is specified.

This command has the following command format:

```
LOG "message text" [NOPREFIX]
```

NOPREFIX

(Optional) Indicates that no timestamp is prefixed to the message text by the SOF server.

Only the stemEMSG variable is returned from the server task.

READ FILES SWITCH—Read Switch Configuration

The READ FILES command reads a specific switch configuration file.

This command has the following command format:

```
READ FILES SWITCH(devic) FILENAME(filename)
```

The output stem variable records will have the same format as the output from the QUERY PORTS command, plus the following two additional simple variables:

- stemDESC - Contains a file description of 1 to 24 characters.
- stemEMSG - Contains an error message of 0 to 72 characters if the file is unable to be read.

TERMINATE—Terminate CCI Session

The TERMINATE command terminates the CCI session. It cancels all outstanding CCI functions and should be the last in a series of ADDRESS SOF commands.

This command has the following command format:

```
TERMINATE
```

No output is returned.

WRITE FILES—Create or Replace Switch Configuration File

The WRITE FILES command writes (creates or replaces) a specific switch configuration file and provides a 1- to 24-character description.

This command has the following command format:

```
WRITE FILES SWITCH(devic) FILENAME(filename) DESCRIPTION(filedesc)
```

Created a stem variable:

stemEMSG

Contains a description of the outcome from 1 to 72 characters when the write operation fails.

QUERY SOF Request Types

This section provides all ADDRESS SOF QUERY request types and the corresponding keywords.

A stemEMSG simple variable is returned after each function call.

QUERY CHPIDS—Query Global Channel Port ID

The QUERY CHPIDS command obtains global channel port ID (ChpID) information for the following:

- A specific ChpID by the device ID
- A range of ChpID based on the relative position number assigned by SOF for all known ChpID

The range form of this command can limit the amount of output per command. To continue the output, start at the next sequential relative position number. If CHPID is omitted and STARTPOS is not specified, then STARTPOS defaults to (0,65535).

This command has the following format:

```
QUERY CHPIDS [CHPID(devId) | STARTPOS(relative position number,  
count of ChpIDs to return) ]
```

Created stem variables

The output stem variable records contain the following word delimited fields for each GLOBAL ChpID information record returned by the SOF server.

- Global relative position number - Contains integer
- Number of connected LPARS - Contains integer
- ChpID channel path number - Contains two printable hex digits
- ChpID identifier - Contains the ChpID DevID string
- Switch port address - Contains two printable hex digits 00-FF
- Switch identifier - Contains the switch DevID string

QUERY CHPIDS LOCAL SYSNAME—Query Local System CHPID

The QUERY CHPIDS LOCAL command obtains local system ChpID information for a range of ChpIDs on a specific system. The SYSNAME keyword is required. CHPNUM defaults to (00,FF) when not specified.

```
QUERY CHPIDS LOCAL SYSNAME(ccisysname)  
[CHPNUM(first returned CHPID number, last returned ChpID number)]
```

Created stem variables: See QUERY CHPID LOCAL CHPID.

QUERY CHPIDS LOCAL CHPID—Query CHPID for Device ID

The QUERY CHPIDS LOCAL CHPID command obtains local system ChpID information for a particular ChpID device ID for all systems to which the ChpID is connected.

QUERY CHPIDS LOCAL CHPID(*devID*)

Created stem variables:

The output stem variable records contain the following word delimited fields for each local ChpID information record returned by the SOF server.

- CCI system name - Contains 1 to 8 characters
- Number of connected devices - Contains integer
- ChpID channel path number - Contains two printable hex digits
- ChpID identifier - Contains the ChpID DevID string
- Switch device number - Contains four printable hex characters
- Switch port address - Contains two printable hex digits 00-FF
- Switch identifier - Contains the switch DevID string

QUERY CONTROLUNITS—Query Global Control Unit

The QUERY CONTROLUNITS command obtains global control unit information for the following:

- A specific control unit or device by the DevID
- A range of control units based on the global relative position number assigned by SOF for all known control units.

You can use the range form of this command for limiting the amount of output per command. Output may be continued by starting at the next sequential relative position number. If CONTROLUNIT and DEVICE are omitted and STARTPOS is not specified, then STARTPOS defaults to (0,65535).

This command has the following format:

```
QUERY CONTROLUNITS  
[CONTROLUNIT(devic)][DEVICE(devic)]STARTPOS  
(global relative position number, count of CUs to return)]
```

Created stem variables:

The output stem variable records will have the following word delimited fields for each GLOBAL control unit information record returned by the SOF server.

- Global relative position number - Contains integer
- CU identifier - Contains the CU DevID string.
For more information see the section Device ID String Composition in this chapter.
- Connected devices count - Contains integer
- Connected switches count - Contains integer
- Connected systems count - Contains integer

QUERY CONTROLUNITS LOCAL SYSNAME—Query the Local System Control Unit

The QUERY CONTROLUNITS LOCAL SYSNAME command obtains local system control unit information for the following:

- All control units on a system
- A range control unit numbers on a system
- A single specific device attached to a control unit on a system

The SYSNAME keyword must be specified.

This command has the following format:

```
QUERY CONTROLUNITS LOCAL SYSNAME(ccisysname)
[CUNUM(first cu device number, count of CUs to be returned)DEVNUM(a
specific device number)]
```

Created stem variables: See the following QUERY CONTROLUNITS LOCAL CONTROLUNIT command.

QUERY CONTROLUNITS LOCAL CONTROLUNIT—Query a Specific Local System Control Unit

The QUERY CONTROLUNITS LOCAL CONTROLUNIT command obtains local system control unit information for a specific control unit device ID for each system that the CU is connected to.

This command has the following format:

```
QUERY CONTROLUNITS LOCAL CONTROLUNIT(device)
```

Created stem variables:

The output stem variable records have the following word delimited fields for each local control unit information record returned by the SOF server. In most cases the hardware identifier and the switch port attached node identifier are identical. However, small discrepancies may exist due to different hardware discovery mechanisms used by the operating system and the switch device. Usually, the difference is in the tag value of the device ID.

- CCI system name - Contains one to eight characters
- CU device number from I/O definition - Contains four printable hex characters
- CU hardware identifier - Contains the CU DevID string
- Connected devices count - Contains integer
- Connected switches count - Contains integer
- CU switch port attached node identifier - Contains the node DevID string.

QUERY CUSWITCH CONTROLUNIT—Query Global Switch

The QUERY CUSWITCH CONTROLUNIT command obtains global switch information for a specific control unit device ID.

This command has the following format:

```
QUERY CUSWITCH CONTROLUNIT(devic)
```

Created stem variables: See the QUERY SWITCHES command.

QUERY CUSWITCH LOCAL SYSNAME—Query Local System Switch

The QUERY CUSWITCH LOCAL SYSNAME command obtains local system switch information for a specific control unit number.

This command has the following format:

```
QUERY CUSWITCH LOCAL SYSNAME(ccisname) CUNUM(cu device number)
```

Created stem variables: See the QUERY SWITCHES LOCAL request.

QUERY DEVICES—Query Specific Global Device

The QUERY DEVICES command obtains global device information for a specific device or for a range of devices based on the global relative position number assigned by SOF for all known devices.

Filtering capabilities are as follows:

- The device ID components filter the device selection within the range of devices. Wildcard filtering is supported by the same wildcard matching characters and rules as the OPSLIKE REXX function.
- The range and filter form of this command limits the amount of output per command. Continue the output by starting at the next sequential relative position number. If DEVICE is omitted and STARTPOS is not specified, then STARTPOS defaults to (0,64).
- The optional BACKWARD keyword scans for devices with global relative position numbers less than or equal to the STARTPOS value.

This command has the following format:

```
QUERY DEVICES  
[DEVICE(devic)]STARTPOS(global relative position number,  
count of devices to return) BACKWARD]  
[FILTERBY(devtype,devmodel,devmfg,devplant,devserial,devtag)]
```

Created stem variables: See the QUERY DEVICES CONTROLUNIT command request.

QUERY DEVICES CONTROLUNIT—Query Range of Control Unit Attached Devices

The QUERY DEVICES CONTROLUNIT command obtains global device information for a range of devices attached to a specific control unit based on the global relative position number assigned by SOF for all known devices.

Selection of devices within the range of devices requested can be filtered by the device ID components. Wildcard filtering is supported using the same wildcard matching characters and rules as the OPSLIKE REXX function. The range and filter form of this command are useful for limiting the amount of output per command. Output may be continued by starting at the next sequential relative position number. If STARTPOS is not specified, then STARTPOS defaults to (0,64). The optional BACKWARD keyword results in scanning for devices with global relative position numbers less than or equal to the STARTPOS value.

This command has the following format:

```
QUERY DEVICES CONTROLUNIT(devic)  
[STARTPOS(global relative position number, count of devices to return)]  
[BACKWARD]  
[FILTERBY(devtype,devmodel,devmfg,devplant,devserial,devtag)]
```

Created stem variables:

The output stem variable records contain the following word delimited fields for each GLOBAL device information record returned by the SOF server. In addition the simple variable name, stemTOT will contain the total number of devices available for retrieval.

- Global relative position number - Contains integer data
- Device identifier - Contains the device DevID string.
- Device connected system count - Contains integer data
- Device CU connections count - Contains integer data

For more information see the section Device ID String Composition in this chapter.

QUERY DEVICES LOCAL SYSNAME—Query Starting Local Device

The QUERY DEVICES LOCAL SYSNAME command obtains local system information for a range of devices based on the starting local device number and count of desired devices. Both keywords are required. If device count is not specified, it defaults to 1. Selection of devices within the range of devices requested can be filtered by the 1-8 character device type produced by the z/OS EDTINFO macro and the last known volume serial number. Wildcard filtering is supported using the same wildcard matching characters and rules as the OPSLIKE REXX function.

This command has the following format:

```
QUERY DEVICES LOCAL SYSNAME(ccisysname) DEVNUM(first local device
number, count of devices to return)
[BACKWARD]
[CUNUM(local CU number)] [FILTERBY(device type,volser) ONLINE]
```

BACKWARD

(Optional) Scans for devices with local device numbers less than or equal to the STARTPOS value.

CUNUM

(Optional) Attaches the selected devices to a particular control unit number.

ONLINE

(Optional) Causes OFFLINE devices to be filtered out.

Created stem variables: Identical to the following request.

QUERY DEVICES LOCAL DEVICE—Query Device ID

The QUERY DEVICES LOCAL DEVICE command obtains local system device information for a specific device by device ID for each system on which the device is defined.

This command has the following format:

```
QUERY DEVICES LOCAL DEVICE(device)
```

Created stem variables:

The output stem variable records have the following word delimited fields for each local device information record that is returned by the S0F server. In addition, the simple variable name, stemTOT contains the total number of local devices available for retrieval.

CCI system name

Contains one to eight characters.

Device number

Contains four printable hex characters.

Device identifier

Contains the device DevID string.

Device UCB type

Contains eight printable hex characters. This value is mapped by the UCBTYP field in the IEFUCBOB macro.

Device Type

Contains the one- to eight-character device type determined by the EDTINFO macro using the UCB device type.

Device last known status

Contains two printable hex characters that contain the device status. This value is mapped by the UCBSTAT field in the IEFUCBOB macro.

Device last known volser

Contains one to six characters or N/A.

CU connection count

Contains small integer.

ChpID connection count

Contains small integer.

ChpID connections array

Contains 16 printable hex characters. Eight occurrences of ChpID where each occurrence is a two-character hexadecimal ChpID number or 00. The number of valid ChpIDs is determined by the ChpID connection count.

ChpID status bit flag array

16 printable hex characters. Eight occurrences of ChpID status bits flag where each occurrence is a two-character hexadecimal value. The value meanings are defined in the IOSDPATH macro at label PATHBITS.

ChpID type array

16 printable hex characters. Eight occurrences of ChpID channel type where each occurrence is a two-character hexadecimal value. Value meanings are defined in the IOSDPATH macro at label PATHINTTYPE.

ChpID port array

16 printable hex characters. Eight occurrences of port address where each occurrence is a two-character hexadecimal value.

CU port array

16 printable hex characters. Eight occurrences of switch port number where each occurrence is a two-character hexadecimal port number or 00.

CU connections array

32 printable hex characters. Eight occurrences of CU number where each occurrence is a four-character hexadecimal CU number or 0000. The number of valid CUs is determined by the CU connection count.

Logical path mask

Eight-byte string consisting of the letters Y and N representing bit settings 1 and 0 in the LPM.

CU switch array

Contains 16 printable hex characters. Eight occurrences of CU switch number where each occurrence is a two-character hexadecimal CU switch number or 00. The number of valid CU switch numbers is determined by the ChpID connection count.

CU switch devid array

Contains 240 printable hex characters. Eight occurrences of CU switch device ID where each occurrence is a thirty-character CU switch device ID or 'N/A-----'. The number of valid CU switch device ids is determined by the ChpID connection count.

QUERY FILES SWITCH—Query Configuration Files

The QUERY FILES SWITCH command obtains a list of configuration files stored on a switch device.

This command has the following format:

`QUERY FILES SWITCH(devic)`

Created stem variables:

The output stem variable records will have the following word delimited fields for each switch file information record returned by the SOF server.

- Switch file name - Contains one to eight characters. Valid name characters are upper case A-Z, 0-9, - and _
- Switch file description - Contains 1 to 24 characters
- Time file last written - Specified as HH:MM:SS
- Date file last written - Specified as yyyy/mm/dd
- File status flag - Specified as one printable hex character

'10'x = File is saved on disk

'08'x = File is open

'04'x = File is locked

'80'x = Last file in the switch file list, which can be combined with other flag settings.

QUERY PORTS PORTADDR—Query ESCON/FICON Switch Ports

The QUERY PORTS PORTADDR command obtains information on a range of ESCON/FICON switch ports for a specific switch device.

This command has the following format:

```
QUERY PORTS PORTADDR(fromport|*,toport|*) {SWITCH(device)}
```

PORTADDR

Specifies a range of port addresses (00-FF) such as PORTADDR(C0,DF). If the PORTADDR keyword is omitted, then the default is PORTADDR(*,*).

fromport|*

Defaults to 00 when * is specified.

toport|*

Defaults to FF when * is specified. If omitted, then the fromport specification is copied.

SWITCH

Contains the switch device ID obtained from the QUERY SWITCHES command or other switch-connected device commands.

Created stem variables:

The output stem variable records will have the following word delimited fields for each port information record returned by the SOF server.

- Port number - Two printable hex characters 00-FF.
- Port address - Two printable hex characters 00-FF.
- Port descriptor - Eight printable hex characters. Same structure as port descriptor words returned by QUERY SWITCH.
- Dedicated port address - Two printable hex characters 00-FF. Only valid when port is marked as dedicated in the port descriptor.
- Port prohibit mask - 256 character string of Y and N characters.

Y - Prohibits dynamic connections

N - Allows dynamic connections

The position of each character in the mask maps to the port address range 00-FF. For example, the second character in the string indicates (Y/N) whether the current port is prohibited from dynamically connecting to port address 01.

- Port name - 1-24 characters or N/A. Imbedded blanks in the name are translated to 'FE'x and need to be translated back to blanks for display and commands.

- Port DCM Flag - Two hex character flag byte with dynamic channel management information for the port.
 - '80'x = Port is installed
 - '40'x = Offline to DCM by command
 - '20'x = Offline to DCM by system
 - '10'x = Ineligible for use by DCM
 - '04'x = Attached to a channel
 - '02'x = Attached to control units
 - '01'x = Neither channel nor control unit attached
- Port Misc Flag - Two hex character flag byte with additional connection information for the port.
 - '80'x = Channel is known on a local system
 - '40'x = Port is an E-port connected to another switch.
- Port ChpID - Two hex character ChpID number when the port is connected to a channel. Only valid if the DCM flag indicates that a channel is attached to the port.
- Node device type - small integer
 - 0 = Unspecified
 - 1 = I/O Device
 - 2 = Control unit
- Node device class - small integer
 - 0 = Unspecified
 - 1 = DASD
 - 2 = Magnetic tape
 - 3 = Unit record (input)
 - 4 = Unit record (output)
 - 5 = Printer
 - 6 = Communications controller
 - 7 = Full screen terminal
 - 8 = Line mode terminal
 - 9 = Channel-to-channel adapter
 - 10 = Switch
- Node identifier - the attached device node ID string.

QUERY SWITCHES—Query ESCON/FICON Switch by Device ID

The QUERY SWITCHES command obtains global ESCON/FICON switch information for a specific switch by device ID or for a range of switches based on the global relative position number assigned by SOF for all known switches. Use the range form of this command to limit the amount of output per command. Output may be continued by starting at the next sequential relative position number. If SWITCH is omitted and STARTPOS is not specified, then STARTPOS defaults to (0,65535).

This command has the following format:

QUERY SWITCHES

[SWITCH(*devic*)]STARTPOS(*global relative position number*,
count of switches to return)

Created stem variables:

The output stem variable records will have the following word delimited fields for each GLOBAL switch information record returned by the SOF server.

- Relative position number - Specified as integer.
- Switch name - Contains 1 to 24 characters or N/A. Imbedded blanks in the name are translated to 'FE'x and need to be translated back to blanks for display and commands.
- Switch identifier - Contains the switch DevID string.
- Ports implemented - Contains small integer 0-255
- Ports installed - Contains small integer 0-255
- ChpIDs attached - Contains small integer 0-255
- Control units attached - Contains small integer 0-255
- Systems attached - Contains small integer 0-255
- 1 byte status flag expressed as 2 byte printable hexadecimal. Multiple flag settings may be combined. The flag indicators are defined as follows:

'80'x = switch could not be allocated

'40'x = switch is offline

'20'x = SW_SWNUM contains global switch number

'10'x = switch is virtual

'08'x = FICON switch

- 256 words of eight-character port descriptors in printable hexadecimal. The port descriptor consists of the following substring fields:
 - 1-2 Port attribute flag
 - '80'x = port is not installed
 - '40'x = port is blocked
 - '20'x = port is prohibited for dynamic connections
 - '10'x = port has a dedicated connection
 - '08'x = port number and address are valid
 - '05'x = FICON port
 - '04'x = external laser port
 - '03'x = external LED port
 - '01'x = internal port
 - '00'x = unknown or not installed port
 - 3-4 Port hardware status flag
 - '80'x = port not installed
 - '40'x = port link failure
 - '20'x = spare port
 - '10'x = port is offline
 - '08'x = port is in maintenance mode
 - '04'x = control unit is connected to port
 - '02'x = port requires service
 - '01'x = invalid port attachment
 - 5-7 Port number 00-FF
 - 7-8 Port address 00-FF

Note: Port addresses are not necessarily the same as their physical port numbers

QUERY SWITCHES LOCAL SYSNAME—Query Specific Local System Switches

The QUERY SWITCHES LOCAL SYSNAME command obtains a full or partial list of local system switches that are connected to the specified system.

This command has the following format:

```
QUERY SWITCHES LOCAL SYSNAME(ccisysname)
[SWNUM]
```

SWINUM

(Optional) Limits returned data by specifying the first switch device number and a count of the number of switches that should be returned.

Default count: 1

The created stem variables are the same as the QUERY SWITCHES LOCAL SWITCH command request.

QUERY SWITCHES LOCAL SWITCH—Query Local System Connected Switches

The QUERY SWITCHES LOCAL SWITCH command obtains the local system view of a single switch for all systems to which the switch is connected.

This command has the following format:

```
QUERY SWITCHES LOCAL SWITCH(device)
```

Created stem variables: The output stem variable records contain the following word delimited fields for each LOCAL switch information record returned by the SOF server:

- CCI system name - 1-8 characters
- Switch device number - 4 printable hexadecimal digits
- Switch name - 1-24 characters or N/A. Imbedded blanks in the name are translated to 'FE'x and need to be translated back to blanks for display and commands.
- Switch identifier - the switch DevID string.
- Ports implemented - small integer 0-255
- Ports installed - small integer 0-255
- ChpIDs attached - small integer 0-255
- Control units attached - small integer 0-255
- Devices attached - integer
- Switch status flag - 2 hexadecimal characters
- 256 words of 8-character port descriptors in printable hexadecimal. The port descriptor is the same as the GLOBAL switch information record.

QUERY SYSTEMS—Query Manager Systems

The QUERY SWITCHES LOCAL SWITCH command obtains a list of SOF z/OS managed systems that are connected to the specified SOF server and have the same CCI application name. The SYSNAME keyword designates a specific CCI system name or '*' for all system names.

For the SWITCH, CONTROLUNIT, DEVICE, and CHPID keywords, obtain a list of systems to which the specified device ID is connected.

This command has the following format:

```
QUERY SYSTEMS  
[SYSNAME(ccisysname*)][SWITCH(device)|CONTROLUNIT(device)|  
DEVICE(device)|CHPID(device)]
```

Created stem variables: The output stem variable records contain the following word delimited fields for each system information response block returned by the SOF server.

- CCI system name - 1-8 characters
- System Status Flag - 2 hexadecimal characters
- SMF name - 1-4 characters
- Central processing complex name - 1-8 characters
- LPAR name - 1-8 characters
- Sysplex name - 1-8 characters or N/A
- System identifier - the system DevID string.
- ESCON/FICON switch count - integer
- Control units count - integer
- Channel path count - integer
- Device count - integer
- Last system health check date - *yyyy/mm/dd* or N/A
- Last system health check time - *hh:mm:ss* or N/A

PPRCCMD Request Types

This section provides all ADDRESS SOF PPRCCMD request types and the corresponding keywords.

A stem EMSG simple variable is returned after each function call.

PPRCCMD DISPLAY GCU—Query Global Control Units (Storage Controllers)

The PPRCCMD DISPLAY GCU command obtains information about all Global Control Units that are detected by a system and have valid World Wide Node Name (WWNN) which indicates that a FICON path can be established.

This command has the following format:

```
PPRCCMD DISPLAY GCU
```

Created stem variables:

The output stem variable records contain these word delimited fields for each GCU information record that is returned by the SOF server.

- CU identifier which contains the CU DevID string
- World Wide Node Name (WWNN)

PPRCCMD DISPLAY PATH—Query all established PPRC paths

The PPRCCMD DISPLAY PATH command obtains all established paths between all Local Control Units (LCUs) that are detected by a system.

This command has the following format:

```
PPRCCMD DISPLAY PATH
```

Created stem variables:

The output stem variable records contain the following delimited fields for each path information record that are returned by the SOF server.

- Primary storage subsystem ID (SSID) which contains primary LCU ID
- Primary serial number which contains serial number of primary storage controller
- Primary logical subsystem (LSS) number
- Primary fibre channel adapter (FCA) number
- Primary world wide node name (WWNN)
- Secondary storage subsystem ID (SSID) which contains secondary LCU ID
- Secondary serial number which contains serial number of secondary storage controller
- Secondary logical subsystem (LSS) number
- Secondary fibre channel adapter (FCA) number
- Secondary world wide node name (WWNN)
- Path status is one byte

PPRCCMD DISPLAY VOLUME—Query all established PPRC volume pairs

The PPRCCMD DISPLAY VOLUME command obtains all established volume pairs that are detected by a system.

This command has the following format:

PPRCCMD DISPLAY VOLUME

Created stem variables:

The output stem variable records contain the following delimited fields for each volume pair information record that are returned by the SOF server.

- Primary volume device ID
- Primary serial number which contains serial number of primary storage controller
- Primary storage subsystem ID (SSID) which contains primary LCU ID
- Primary channel connection address (CCA)
- Primary logical subsystem (LSS) number
- Secondary storage subsystem ID (SSID) which contains secondary LCU ID
- Secondary channel connection address (CCA)
- Secondary logical subsystem (LSS) number
- Copy state
- Volume state

PPRCCMD SETUP PATH—Setup new PPRC path

The PPRCCMD SETUP PATH command establishes an FCP (fibre channel protocol) path between a primary site logical subsystem and a recovery (secondary) site logical subsystem.

When the primary site logical subsystem supports eight paths, each SETUP PATH command can establish up to eight paths from one primary site logical subsystem to a single recovery site logical subsystem. You can link up to four recovery site logical subsystems to one primary site logical subsystem. These links require that you issue a separate SETUP PATH command for each logical subsystem.

Note: To *replace* paths use the SETUP PATH command. The paths on the command are treated as a path set. Any subsequent SETUP PATH command removes paths that were previously established and replaces them with the new set of paths. To ensure that you do not remove paths that you want to keep, use caution when using this command.

This command has the following format:

```
PPRCCMD SETUP PATH DEVNUM(devid) LINK(up to 8 path addresses)
PWNN(wwnn) PSSD(ssid) [PLSS(ss)]
SWNN(wwnn) SSSD(ssid) [SLSS(ss)]
[CGROUP(YES|NO) SUBCHSET(0|1)]
```

DEVNUM(devid)

Specifies the device number of any simplex disk volume that is attached to the primary storage controller without directing any operation to the volume. The *devid* is the four-digit hexadecimal address of any disk volume behind the primary site storage control.

LINK(X'aaaabbbb' [X'ccccddd' X' xxxxyyyy' ...])

Specifies the addressing paths from the primary site logical subsystem to the recovery site logical subsystem.

You can specify up to eight Fibre Channel path addresses (if the primary site logical subsystem supports it), separated with a blank.

Note: Duplicate link addresses are not allowed.

Each path address is an 8-digit hexadecimal address. The format of each Fibre Channel path address has the hex digit form *aaaabbbb* where:

aaaa

Specifies the Fibre Channel adapter that is used for the path in the primary subsystem.

bbbb

Specifies the Fibre Channel adapter that is used for the path in the secondary subsystem.

PWNN(wwnn)

Specifies the World Wide Node Name of the primary storage controller and it is exactly 16-digit hexadecimal number.

PSSID(ssid)

Specifies the primary site storage subsystem ID and it is a four-digit hexadecimal number.

PLSS(lss)

(Optional) Specifies the primary site logical subsystem number and it is a two-digit hexadecimal number.

SWNN(wwnn)

Specifies the World Wide Node Name of the secondary storage controller and it is exactly 16-digit hexadecimal number.

SSSID(ssid)

Specifies the secondary site storage subsystem ID and it is a four-digit hexadecimal number.

SLSS(lss)

(Optional) Specifies the secondary site logical subsystem number and it is a two-digit hexadecimal number.

CGROUP(yes/no)

(Optional) Specifies whether a PPRC volume pair in a consistency group remains in a long-busy condition for a specified period following an error that is related to the associated volume pair. When an error occurs, and one of the volumes in that consistency group is suspended. The volumes in that consistency group are suspended when the next write occurs to a volume.

One of the following values must be specified with the CGROUP parameter:

NO

(Default) Specifies when an error occurs to a PPRC volume pair in a consistency group, the long-busy condition is not enabled for the associated volume pair.

YES

Specifies when an error occurs to a PPRC volume pair in a consistency group, the long-busy condition is enabled for the associated volume pair.

Note: Automation products can detect PPRC volume pair outages on these devices and can take the appropriate action.

SUBCHSET(0/1)

(Optional) Specifies the subchannel set where the command is issued. The subchannel set where the I/O is issued is the subchannel set as defined in the Hardware Configuration Dialog (HCD). The values are:

0

(Default) Subchannel set 0.

1

Subchannel set 1.

Created stem variables:

The output stem variable records contain the following delimited fields:

- Return code
- Reason code
- Error message

OPS--PPRCCMD DELETE PATH—Delete selected PPRC path--R12.1

The PPRCCMD DELETE PATH command deletes all established paths between a primary site logical subsystem and a recovery (secondary) site logical subsystem. Only active paths between the specified logical subsystems are affected; all other paths to other subsystems are unaffected.

Note: Before issuing a DELETE PATH command, issue a DELETE VOLUME command to all active PPRC volume pairs. The DELETE PATH command can cause the issuance of an ANTP0121I message when this sequence is not followed.

This command has the following format:

```
PPRCCMD DELETE PATH DEVNUM(devid)
PWNN(wwnn) PSSID(ssid) [PLSS(lss)]
SWNN(wwnn) SSSID(ssid) [SLSS(lss)]
[SUBCHSET(0/1)]
```

DEVNUM(devid)

Specifies the device number of any simplex disk volume that is attached to the primary storage controller without directing any operation to the volume.

The devid is the four-digit hexadecimal address of any disk volume behind the primary site storage control.

PWNN(wwnn)

Specifies the World Wide Node Name of the primary storage controller and it is exactly 16-digit hexadecimal number.

PSSID(ssid)

Specifies the primary site storage subsystem ID and it is a four-digit hexadecimal number.

PLSS(lss)

(Optional) Specifies the primary site logical subsystem number and it is a two-digit hexadecimal number.

SWNN(wwnn)

Specifies the World Wide Node Name of the secondary storage controller and it is exactly 16-digit hexadecimal number.

SSSID(ssid)

Specifies the secondary site storage subsystem ID and it is a four-digit hexadecimal number.

SLSS(lss)

(Optional) Specifies the secondary site logical subsystem number and it is a two-digit hexadecimal number.

SUBCHSET(0/1)

(Optional) Specifies the subchannel set where the command is issued. The subchannel set where the I/O is issued is the subchannel set as defined in the Hardware Configuration Dialog (HCD). The values are:

0

(Default) Subchannel set 0.

1

Subchannel set 1.

Created stem variables:

The output stem variable record contains the following delimited fields:

- Return code
- Reason code
- Error message

PPRCCMD SETUP VOLUME—Setup new PPRC volume pairs

The PPRCCMD SETUP VOLUME command specifies PPRC primary and secondary volumes. In order for this command to complete successfully, a PPRC link must be active between the primary logical subsystem and the secondary logical subsystem.

When establishing a pair of devices, the primary and secondary volumes must have the same number of tracks on each cylinder and the same number of bytes on each track. The secondary device must have the same number or a greater number of cylinders as compared to the primary device.

This command has the following format:

```
PPRCCMD SETUP VOLUME DEVNUM(devid) VOLCNT(a number of consecutive volumes to be processed)
PSERIAL(serial) PCCA(cca) PSSID(ssid) [PLSS(lss)]
SSERIAL(serial) SCCA(cca) SSSID(ssid) [SLSS(lss)]
[SUBCHSET(0|1)] [MODE(COPY|NOCOPY|INCRS|RESYNQ)] [CRIT(YES,NO)] [ONLINESEC(YES,NO)]
```

DEVNUM(devid)

Specifies the device number of the primary volume for the volume pair to be established. The device number is the four-digit hexadecimal address of the device to which the I/O operation is directed.

VOLCNT(number)

Specifies the number of consecutive primary volumes to be paired with the same number of secondary volumes within one logical subsystem.

PSERIAL(serial)

Specifies the primary storage control serial number, which can include up to 12 digits, depending on the type of storage control.

PCCA(cca)

Specifies the primary a two-digit channel connection address.

PSSID(ssid)

Specifies the primary site storage subsystem ID and it is a four-digit hexadecimal number.

PLSS(lss)

(Optional) Specifies the primary site logical subsystem number and it is a two-digit hexadecimal number.

SSERIAL(serial)

Specifies the secondary storage control serial number, which can include up to 12 digits, depending on the type of storage control.

SCCA(cca)

Specifies the secondary a two-digit channel connection address.

SSSID(ssid)

Specifies the secondary site storage subsystem ID and it is a four-digit hexadecimal number.

SLSS(Iss)

(Optional) Specifies the secondary site logical subsystem number and it is a two-digit hexadecimal number.

SUBCHSET(0/1)

(Optional) Specifies the subchannel set where the command is issued. The subchannel set where the I/O is issued is the subchannel set as defined in the Hardware Configuration Dialog (HCD). The values are:

0

(Default) Subchannel set 0.

1

Subchannel set 1.

MODE(copy/nocopy/incres/resync)

(Optional) Specifies one of the following PPRC modes:

COPY

Specifies that PPRC copy all tracks on the primary volume to the secondary volume.

NOCOPY

(Default) Specifies only those tracks on the primary volume that are updated after this command is issued are copied to the secondary volume. Specify this mode only if the volumes are exact copies of each other.

The NOCOPY option assumes that the specified volumes are fully synchronized at the time the copy is started. Ensure that no I/O operations occur to either volume before starting the copy operation.

Important! To protect the data integrity of the two volumes, use this option.

INCRES

Limits the tracks to be copied by doing an incremental resynchronization using the change recording bitmaps.

RESYNC

Specifies that PPRC reestablish a suspended copy. Use this option only after verifying that either the primary and secondary volumes are both in a suspended state, or the secondary is in a simplex state and the primary is suspended. PPRC only copies from the primary volume to the secondary volume the cylinders that were updated during the period of suspension.

CRIT(yes/no)

(Optional) Specifies whether PPRC allows subsequent write requests to the primary volume. The values are:

NO

(Default) Specifies if an I/O completion error occurs that PPRC allows subsequent write requests to the PPRC primary volume. The primary storage control performs change recording.

YES

Specifies if an I/O error occurs that PPRC either allows or does not allow subsequent writes, depending on how the storage subsystem is configured. The PPRC pair then remains in a suspended state until you correct the problem.

ONLINSEC(yes/no)

(Optional) Specifies whether the secondary volume is online to a host system. The values are:

NO

(Default) Specifies if the secondary volume appears to be online to any host system, the `SETUP VOLUME` command fails with an error.

YES

Specifies that the `SETUP VOLUME` command can be processed even if the secondary volume appears to be online to a host system.

Created stem variables:

The output stem variable record contains the following delimited fields:

- Return code
- Reason code
- Error message

PPRCCMD DELETE VOLUME—Delete selected PPRC volume pair

The PPRCCMD DELETE VOLUME command deletes a volume pair.

This command has the following format:

```
PPRCCMD DELETE VOLUME DEVNUM(devid)
PSERIAL(serial) PCCA(cca) PSSID(ssid) [PLSS(lss)]
SSERIAL(serial) SCCA(cca) SSSID(ssid) [SLSS(lss)]
[SUBCHSET(0/1)]
```

DEVNUM(devid)

Specifies the device number of the primary volume for the volume pair to be deleted. The device number is the four-digit hexadecimal address of the device to which the I/O operation is directed.

PSERIAL(serial)

Specifies the primary storage control serial number, which can include up to 12 digits, depending on the type of storage control.

PCCA(cca)

Specifies the primary a two-digit channel connection address.

PSSID(ssid)

Specifies the primary site storage subsystem ID and it is a four-digit hexadecimal number.

PLSS(lss)

(Optional) Specifies the primary site logical subsystem number and it is a two-digit hexadecimal number.

SSERIAL(serial)

Specifies the secondary storage control serial number, which can include up to 12 digits, depending on the type of storage control.

SCCA(cca)

Specifies the secondary a two-digit channel connection address.

SSSID(ssid)

Specifies the secondary site storage subsystem ID and it is a two-digit hexadecimal number.

SLSS(lss)

(Optional) Specifies the secondary site logical subsystem number and it is a four-digit hexadecimal number.

SUBCHSET(0/1)

(Optional) Specifies the subchannel set where the command is issued. The subchannel set where the I/O is issued is the subchannel set as defined in the Hardware Configuration Dialog (HCD). The values are:

0

(Default) Subchannel set 0.

1

Subchannel set 1.

Created stem variables:

The output stem variable record contains the following delimited fields:

- Return code
- Reason code
- Error message

PPRCCMD FREEZE—Stop Updates to the Secondary Subsystem

You can use PPRCCMD FREEZE when your automation application detects a failure while mirroring data from the primary to the secondary. The FREEZE command lets you preserve data consistency.

Important! You must be intimately familiar with PPRC before coding applications that issue the FREEZE command.

The PPRCCMD FREEZE command stops all updates to secondary subsystems for pairs that have primary and secondary volumes on the corresponding logical subsystems that this command specified. The system suspends these pairs and removes all paths between the specified primary and secondary subsystems.

All host application I/O to the primary holds off with a long busy condition for 2 minutes, or until a PPRCCMD RUN command is issued. To use the FREEZE command, establish the path with CGROUP(YES) as the PPRCCMD SETUP PATH command specifies.

If you fail to specify CGROUP(YES) in the PPRCCMD SETUP PATH command, the host I/O is not held off with a long busy condition. To resume operations, you reestablish the paths between the primary and secondary subsystems. Then, you reestablish the pairs using the MODE(RESYNC) parameter, through the PPRCCMD SETUP VOLUME command.

This command has the following format:

```
PPRCCMD FREEZE DEVNUM(devid)
PSERIAL(serial) PSSID(ssid) [PLSS(lss)]
SSERIAL(serial) SSSID(ssid) [SLSS(lss)]
[SUBCHSET(0|1)]
```

DEVNUM(devid)

Specifies the device number of any disk volume that is attached to the primary site storage control, without directing any operation to the volume. The device number is the four-digit hexadecimal address of any disk volume behind the primary site storage control.

PSERIAL(serial)

Specifies the primary storage control serial number, which can include up to 12 digits, depending on the type of storage control.

PSSID(ssid)

Specifies the primary site storage subsystem ID as a four-digit hexadecimal number.

PLSS(lss)

(Optional) Specifies the primary site logical subsystem number as a two-digit hexadecimal number.

SSERIAL(serial)

Specifies the secondary storage control serial number, which can include up to 12 digits, depending on the type of storage control.

SSSID(ssid)

Specifies the secondary site storage subsystem ID as a two-digit hexadecimal number.

SLSS(lss)

(Optional) Specifies the secondary site logical subsystem number as a four-digit hexadecimal number.

SUBCHSET(0/1)

(Optional) Specifies the subchannel set that issued the command. The subchannel set that issued the I/O is the subchannel set as defined in the Hardware Configuration Dialog (HCD).

This set has the following values:

0

(Default) Subchannel set to 0.

1

Subchannel set to 1.

Created stem variables:

The output stem variable record contains the following delimited fields:

- Return code
- Reason code
- Error message

PPRCCMD RUN—Resumes I/O on Primary Volumes

The PPRCCMD RUN command specifies that host application I/O can resume on primary volumes for pairs that have primary and secondary volumes on the corresponding logical subsystems that this command specified, which have been suspended due to an earlier PPRCCMD FREEZE command.

This command has the following format:

```
PRCCMD RUN DEVNUM(devid)
PSERIAL(serial) PSSID(ssid) [PLSS(lss)]
SSERIAL(serial) SSSID(ssid) [SLSS(lss)]
[SUBCHSET(0|1)]
```

DEVNUM(devid)

Specifies the device number of any disk volume that is attached to the primary site storage control without directing any operation to the volume. The device number is the four-digit hexadecimal address of any disk volume behind the primary site storage control.

PSERIAL(serial)

Specifies the primary storage control serial number, which can include up to 12 digits, depending on the type of storage control.

PSSID(ssid)

Specifies the primary site storage subsystem ID as a four-digit hexadecimal number.

PLSS(lss)

(Optional) Specifies the primary site logical subsystem number as a two-digit hexadecimal number.

SSERIAL(serial)

Specifies the secondary storage control serial number, which can include up to 12 digits, depending on the type of storage control.

SSSID(ssid)

Specifies the secondary site storage subsystem ID as a two-digit hexadecimal number.

SLSS(lss)

(Optional) Specifies the secondary site logical subsystem number as a four-digit hexadecimal number.

SUBCHSET(0/1)

(Optional) Specifies the subchannel set where the command is issued. The subchannel set where the I/O is issued is the subchannel set as defined in the Hardware Configuration Dialog (HCD).

This set has the following values:

0

(Default) Subchannel set to 0.

1

Subchannel set to 1.

Created stem variables:

The output stem variable record contains the following delimited fields:

- Return code
- Reason code
- Error message.

ADDRESS SOF Examples

The examples presented here demonstrate various uses of the ADDRESS SOF command. Actual commands will start with “ADDRESS SOF”. To simplify the appearance in these examples, “ADDRESS SOF” is omitted.

Example 1: Find the Servers

This command finds all SOF servers with the CCI application name starting with 'ABCDE'.

```
"FND SOF SERVER(*,ABCDE*)"
```

Note: This is a command to the CCI API, and you can use wildcards to specify the SERVER.

Example 2: Obtain List of Managed Systems

This command obtains a list of all SOF managed systems that are connected to the specified SOF server and have the same CCI application name.

```
"QUERY SYSTEMS SYSNAME(*) SERVER(A11SENF,OPSofServer)"
```

Example 3: Send an Operator Command to the SOF Server

This command sends an operator command to the SOF server specified by the SERVER keyword.

```
"COMMAND 'Display Defaults' SERVER(A31SENF,OPSofServer)"
```

Example 4: Obtain Port Information

This command obtains port information on switch ports 0-F, associated with the switch identified by the SWITCH keyword.

```
"QUERY PORTS PORTADDR(0,f)", "SWITCH(006064001MCD010000831206250006)",  
"SERVER(A11SENF,OPSofServer)"
```

Example 5: Obtain Configuration Information

This command obtains configuration file information for the file associated with the switch identified by the SWITCH keyword.

```
"QUERY FILES SERVER(A11SENF,OPSofServer)",  
"SWITCH(009032005IBMD20000000412030000)"
```


Example 6: Use of the FILTERBY Keyword

This example issues the FILTERBY keyword on the QUERY command against a control unit. It will retrieve information only for attached devices made by manufacturer XYZ with tags starting with 4A.

```
ADDRESS SOF 'QUERY DEVICES';  
'CONTROLUNIT(002105302MFG060000000806960326)',  
'FILTERBY(*,XYZ,*,4A*)'
```

Both the control unit and every attached device have a unique device ID (DevID) consisting of the following components:

- Type - 6 characters
- Model - 3 characters
- Manufacturer - 3 characters
- Plant - 2 characters
- Serial # - 12 characters
- Tag - 4 characters

Note: In this example, we entered the entire control unit DevID for proper CU identification, separated the DevID for device selection into individual components to comply with FILTERBY syntax, specified the full manufacturer 3-character ID, and partially wildcarded the Tag.

PPRCCMD Request Examples

Example 1: Display all defined paths

This command finds all PPRC paths which exist within the current environment.

```
"PPRCCMD DISPLAY PATH SERVER(A11SENF,OPSofServer)"
```

Example 2: Create new path between primary and secondary LSSes

This command establishes new PPRC path which consists of two fibre channel links.

```
"PPRCCMD SETUP PATH DEVNUM(8401) PWWNN(5005076308FFC641) FSSD(8400) FLSS(02)",  
"SWMNN(5005076303FFC38B) SSSD(8500) SLSS(03)",  
"CGROUP(NO) SUBCHSET(0)",  
"LINK(X00330233' X01030303)",  
"SERVER(A11SENF,OPSofServer)"
```

Example 3: Create new volume pairs

This command creates ten consecutive volume pairs starting with 8410-8510 pair.

```
"PPRCCMD SETUP VOLUME DEVNUM(8410) VOLCNT(10)",  
"PSERIAL(0000000MC711) PCCA(10) FSSD(8400) FLSS(02)",  
"SSERIAL(000000062241) SCCA(10) SSSD(8500) SLSS(03)",  
"SERVER(A11SENF,OPSofServer)"
```

ADDRESS SQL Commands

For complete information about ADDRESS SQL instructions, including syntax diagrams, see the chapter "[Relational Data Framework Reference](#) (see page 415)."

For complete RDF usage, see the chapter "Using the Relational Data Framework" in the *User Guide*.

ADDRESS SYSVIEWE—Interface to CA SYSVIEW

The ADDRESS SYSVIEWE host environment provides a direct interface to the CA SYSVIEW product without requiring the use of CA GSS. The ADDRESS SYSVIEWE host environment sends commands directly to CA SYSVIEW and returns output from those commands directly to the OPS/REXX external data queue.

Note: For usability and new functionality information, see the example in member SYSVIEWE of the SAMPLES data set.

For example, the following code:

```
ADDRESS SYSVIEWE
"COMMAND(LINKLIST)"
"COMMAND(END)"
do while QUEUED() > 0
  pull line
  say line
end
```

Produces output similar to the following:

```
MGSVX900I CA SYSVIEW 12.5 COPYRIGHT 2009 CA, INC.
T|SYSVIEW|12.5|LA88|LINKLIST|10/09/09|14:50:31|
I|JOBNAME USER104 ASD 012C JOBID TSU03947
I|SETNAME LNKLIST00 STATUS CURRENT FL CHK ALLOCATIONS ACTIVE
I|LLA SEARCH AVAILABLE EXTENTS 109
I|LIBRARIES 89 ALLOC 0 OPEN 0
H|CMD|DATASET-NAME X|TN|VOLSER|APF|MESSAGE
D| |SYS1.LINKLIB | 2|MRB14|APF|
D| |SYS1.MIGLIB | 1|MRB14|APF|
D| |SYS1.CMDLIB | 1|MRB14|APF|
```

Usage Notes:

- The first character in each returned line indicates the type of data being returned.
- The END command must be the last command you send to the API. As a result of the END command, the RC variable is set to 20 to indicate that the API has terminated.
- The ADDRESS SYSVIEWE host environment is not available in AOF rules.

ADDRESS SYSVIEWE Return Codes

The ADDRESS SYSVIEWE host environment produces these return codes:

100

ADDRESS SYSVIEWE cannot be used in a rule.

103

LOAD failed for the SYSVIEWE API module.

All of the return codes below 100 are documented in the CA SYSVIEW guides. For more information about these return codes and other aspects of the CA SYSVIEW API, see the CA SYSVIEW guides.

Note: Return code 36 indicates a problem with the REXX external data queue. In most cases this means that the external data queue overflowed.

ADDRESS TSO—Route Commands to TSO

The ADDRESS TSO instruction routes commands to TSO. OPS/REXX captures output from TSO commands using the standard TSO/E mechanism for trapping messages issued with the PUTLINE service routine. Nevertheless, OPS/REXX captures neither messages issued by TSO using TPUT nor full-screen output, however requested. Host commands sent to a server through ADDRESS TSO can contain no more than 256 bytes. When invoked from rules (with the exception of REQ rules), host commands are always sent to a server for execution.

This command has the following format:

```
ADDRESS TSO "TSO command"
```

ADDRESS TSO Return Codes

The meaning of the ADDRESS TSO return code value varies depending on the command issued.

ADDRESS USS Commands

ADDRESS USS is an OPS/REXX host command environment that allows you to issue UNIX System Services commands and, optionally, retrieve command responses. The commands are sent to the USS class of OSF servers for execution in a UNIX System Services environment, and the responses are returned to the issuing OPS/REXX program in the form of REXX stem variables.

With the exception of USSCMD, the ADDRESS USS commands result in calls to the CCS for z/OS Application Programming Interface (API) from the USS server. These API calls augment the standard UNIX commands provided with CCS for z/OS that are executable using the USSCMD command.

The API commands bring additional functionality to the standard CCS for z/OS commands because they provide the ability to specify values for the optional fields of the message records (for example, message severity that can be used by Event Manger, or CA OPS/MVS rules for automation applications). Some API functions, such as DOM, have no equivalent UNIX command in CCS for z/OS.

For example, when executed as a USSCMD, the CAWTOR waits indefinitely for a response. The WTOR API call honors the wait time specified by the caller and removes the WTOR if no response is provided in the waiting period. This prevents USS server cancellation if the transaction elapsed time limit for the server is exceeded.

ADDRESS USS USSCMD—Issue UNIX Commands

The USSCMD command issues any UNIX command that can be executed from a shell environment. This includes the CAWTOR, CAWTO, OPRCMD, CAREPLY, and OPRPING commands that are provided with CCS for z/OS.

This command has the following format:

```
ADDRESS USS "USSCMD keywords"  
{COMMAND('cmd text')}  
[LOG(Y|N)]  
[STEM(stem)]  
[WAIT(seconds)]  
[DELAY(seconds)]
```

COMMAND

This required keyword defines the text of the USS command. It can be up to 2048 characters in mixed case. Note that proper care must be given for text that contains any metacharacters, or characters with special meaning to the UNIX System Services (USS) shell. Some common metacharacters are ^ \$. * + ? [] | () \. To use one of these characters literally (without its special meaning), enter a backslash in front of the character. For example, to use the \$ character without its special meaning, specify \\$ in the text. For a complete list of metacharacters, see the IBM documentation on the USS shell.

LOG, STEM, WAIT, and DELAY

(Optional) You may specify the LOG, STEM, WAIT, and DELAY keywords, which are common to all ADDRESS USS commands.

Important! The following commands are specific to the API. The USS server validates all keyword operands specific to the API commands. The syntax for the ADDRESS USS API verbs follows.

More information:

[Keywords Common to All ADDRESS USS Commands](#) (see page 397)

ADDRESS USS CMD—Issue USS Commands

The CMD command issues a USS command or any internal command that is executed by the Event Management component of CCS for z/OS. The response is displayed on the CA Event Manager console. No response lines are returned.

Limits: The maximum length of the complete USS command including keywords and operands is 2048 bytes.

This command has the following format:

```
ADDRESS USS "CMD keywords"

/* Optional API Keywords */
[COMMAND('cmd text')]
[CATEGORY('category name')]
[SOURCE('source name')]
[COLOR(red|green|yellow|orange|blue|pink|purple)]
[ATTRIBUTE(BLINK|REVERSE)]
[USERID('user id')]
[USERDATA('user data')]
[PROCESS('process data')]
[WORKSTATION('workstation name')]
[TAG ('platform')]
[DEVICE('device name')]
[NODE(nodename)]

/* Optional USS Keywords */
[LOG(Y|N)]
[STEM(stem)]
[WAIT(seconds)]
[DELAY(seconds)]
```

The optional API keyword COMMAND of the CMD command has this value:

COMMAND

(Optional) Defines the text of the API command.

For the values of the other API keywords, see API Keywords in this chapter.

The USS keywords of the CMD command have these values:

LOG, STEM, WAIT, and DELAY

(Optional) You may specify the LOG, STEM, WAIT, and DELAY keywords, which are common to all ADDRESS USS commands.

More information:

[Keywords Common to All ADDRESS USS Commands](#) (see page 397)

ADDRESS USS DOM—Remove Message from Console Display

The DOM command removes a message from the held message area of the CA Event Manager console display. This function is the equivalent of the Acknowledge Message function that is usually performed manually at the CA Event Manager console or by the DELKEEP command on the Windows version of CA NSM.

This command has the following format:

```
ADDRESS USS "DOM keywords"
```

```
/* Optional API Keywords */  
[TEXT(match text)]  
[REPLYID(reply number)]  
[MSGNUM(message number)]
```

```
/* Optional USS Keywords */  
[LOG(Y|N)]  
[STEM(stem)]  
[WAIT(seconds)]  
[DELAY(seconds)]
```

The USS keywords of the DOM command have these values:

LOG, STEM, WAIT, and DELAY

(Optional) You may specify the LOG, STEM, WAIT, and DELAY keywords, which are common to all ADDRESS USS commands.

More information:

[Keywords Common to All ADDRESS USS Commands](#) (see page 397)

[API Keywords](#) (see page 393)

ADDRESS USS PING—Ping CA Event Manager

The PING command tests whether CA Event Manager is active and reachable on a specified node name.

Note: The USSCODE variable indicates whether the PING was successful. For more information on the USSCODE variable, see USSCODE and USSREASON Variables in this chapter.

This command has the following format:

```
ADDRESS USS "PING keywords"
```

```
/* Optional API Keywords */  
[NODE(nodename)]  
[LITE]
```

```
/* Optional USS Keywords 8?  
[LOG(Y|N)]  
[STEM(stem)]  
[WAIT(seconds)]  
[DELAY(seconds)]
```

The USS keywords of the PING command have these values:

LOG, STEM, WAIT, and DELAY

(Optional) You may specify the LOG, STEM, WAIT, and DELAY keywords, which are common to all ADDRESS USS commands.

More information:

[Keywords Common to All ADDRESS USS Commands](#) (see page 397)

[API Keywords](#) (see page 393)

ADDRESS USS REPLY—Reply to WTOR

The REPLY command replies to an outstanding WTOR in the Event Management component of CCS for z/OS.

This command has the following format:

```
ADDRESS USS "REPLY keywords"
```

```
/* Optional API Keywords */
```

```
[TEXT(reply text)]
```

```
[REPLYID(reply number)]
```

```
[NODE(nodename)]
```

```
/* Optional USS Keywords */
```

```
[LOG(Y|N)]
```

```
[STEM(stem)]
```

```
[WAIT(seconds)]
```

```
[DELAY(seconds)]
```

The USS keywords of the REPLY command have these values:

LOG, STEM, WAIT, and DELAY

(Optional) You may specify the LOG, STEM, WAIT, and DELAY keywords, which are common to all ADDRESS USS commands.

Note: Due to the asynchronous nature of the REPLY command, the specified REPLYID is not validated prior to the return of the underlying CA Event Manager command to the ADDRESS USS environment. Therefore, an invalid REPLYID is not reported through the returned RC value. If the REPLY ID is invalid, then the following USS message is generated on the system where the REPLY command was targeted to run. If it was a z/OS system, then it is available to USS rules and OPSLOG.

```
%CAOP_E_501 Replyid not found.
```

You can write a USS rule on the above message using CAOP_E_501 as the message ID.

More information:

[Keywords Common to All ADDRESS USS Commands](#) (see page 397)

[API Keywords](#) (see page 393)

ADDRESS USS WTO—Send Message to Event Manager

The WTO command issues a message to the Event Management component of CCS for z/OS. This is the API version of the CAWTO command.

This command has the following format:

ADDRESS USS 'WTO *keywords*'

/ Optional API Keywords */*

[TEXT(*message text*)]
 [NODE(*nodename*)]
 [MSGNUM(*message number*)]
 [CATEGORY(*category name*)]
 [SOURCE(*source name*)]
 [SEVERITY([W|E|S|F])]
 [COLOR(red|green|yellow| orange|blue|pink|purple)]
 [ATTRIBUTE(BLINK|REVERSE)]
 [KEEP]
 [HIDE]
 [INVISIBLE]
 [LOGONLY]
 [USERID(*user id*)]
 [USERDATA(*user data*)]
 [PROCESS(*process data*)]
 [WORKSTATION(*workstation name*)]
 [TAG(*platform*)]
 [DEVICE(*device name*)]

/ Optional USS Keywords */*

[LOG(Y|N)]
 [STEM(*stem*)]
 [WAIT(*seconds*)]
 [DELAY(*seconds*)]

The USS keywords of the WTO command have these values:

LOG, STEM, WAIT, and DELAY

(Optional) You may specify the LOG, STEM, WAIT, and DELAY keywords, which are common to all ADDRESS USS commands.

More information:

[Keywords Common to All ADDRESS USS Commands](#) (see page 397)

[API Keywords](#) (see page 393)

ADDRESS USS WTOR—Send and Receive Messages

The WTOR command issues a message to, *and* waits for a reply from, the Event Management component of CCS for z/OS. If the reply is completed before the wait time expires, the reply text is returned. The CAWTOR USS command waits until a response is received with no time out provision. If no response is supplied, the USS server abnormally ends because the time limit for the transaction was exceeded.

Note: Five seconds are added to the value specified for the WAIT keyword to allow for the removal of a WTOR for which no reply is received.

This command has the following format:

```
ADDRESS USS "WTOR keywords"

/* Optional API Keywords */
[TEXT(message text)]
[ATTRIBUTE(BLINK|REVERSE)]
[CATEGORY(category name)]
[COLOR(red|green|yellow|orange|blue|pink|purple)]
[DEVICE(device name)]
[HILITE]
[INVISIBLE]
[KEEP]
[LOGONLY]
[NODE(nodename)]
[MSGNUM(message number)]
[PROCESS(process data)]
[SEVERITY(I|W|E|S|F)]
[SOURCE(source name)]
[TAG(platform)]
[USERDATA(user data)]
[USERID(user id)]
[WORKSTATION(workstation name)]
[DELAY(seconds)]

/* Optional USS Keywords */
[LOG(Y|N)]
[STEM(stem)]
[WAIT(seconds)]
```

The USS keywords of the WTOR command have these values:

LOG, STEM, WAIT, and DELAY

(Optional) You may specify the DELAY, LOG, STEM, and WAIT keywords, which are common to all ADDRESS USS commands.

Note: The HILITE, INVISIBLE, and KEEP keywords do not have any effect on a command when it is directed to a CA Event Console on z/OS, whether it be the local system where the ADDRESS USS WTOR command was issued or a remote system where the WTOR command was directed through the NODE keyword.

More information:

[API Keywords](#) (see page 393)

[Keywords Common to All ADDRESS USS Commands](#) (see page 397)

API Keywords

This section discusses the keywords for the Application Programming Interface (API).

ATTRIBUTE

Specifies the video attributes of the message in the CA Event Manager console display.

- BLINK-Causes the message to blink on and off
- REVERSE-Causes the message to appear against a highlighted background

Default: None

CATEGORY

Categorizes the message event using a string of 1 to 255 characters.

Default: UNIAPI

COLOR

Specifies the color of the message in the CA Event Manager console display. Valid values are red, green, yellow, orange, blue, pink, and purple.

cmd text

Specifies a 1- to 2048-character UNIX command in mixed case that is valid for a UNIX shell environment.

command text

Specifies a 1- to 255-character string of command text for the CA Event Manager in mixed case.

DEVICE

Specifies a string of 1 to 255 characters that associates an SNMP trap message with a particular device.

Default: None

INVISIBLE

Causes the message not to appear on the CA Event Manager console display. The message is still retained in the CA Event Manager log file.

Default: The message displays

HILITE

Causes the message to be highlighted on the CA Event Manager console display.

Default: Normal intensity

KEEP

Indicates that the message should be placed in the held message area of the CA Event Manager console display until the message is acknowledged.

Default: The message displays in the normal message area.

LITE

Indicates that PING should only test the connectivity to the target node, without sending a test message to the CA Event Manager on the target system.

LOGONLY

Skips the CA Event Manager action processing for the message. This means that the message cannot be automated by CA Event Manager rules; however, it can still be automated by CA OPS/MVS USS rules.

Default: Event Manager action processing proceeds

match text

Contains the 1- to 255-character string that is the exact message text of the message that is to be DOMd when the message number is not supplied. Generic matching of the remainder of the text is implied when an asterisk * appears as the last character of the match text.

MSGNUM

Assigns an arbitrary positive fullword integer to the message for identification purposes (such as 1302 in the OPS1302E message). When used with the DOM verb, the number for the message is the CA Event Manager log record number, which is acknowledged and removed from the held display area. The log record number can only be obtained from the detail message display of the CA Event Manager console.

Default: None

TEXT

Specifies a 1- to 255-character string of message text in mixed case.

NODE

Specifies a string of 1 to 64 characters that designates the target node for the API function. The TCP/IP host name of the target system is required for all API functions that are not directed to the local system.

Default: The local node name

TAG

Specifies a string of 1 to 255 characters that records the type of platform that generated the message. When this operand is omitted, default values are supplied for each platform type that runs Unicenter.

Default:

- OS390 (for CCS for z/OS)
- WNT (for the Windows NT framework)

PROCESS

Specifies a string of 1 to 255 characters that records the USS process ID and program data of the issuer of the message. The required format is: *nnnnn,data*

Notes:

- *nnnnn* is a numeric value (usually the USS process number). If this value is omitted, you must still supply the comma.
- *data* is an arbitrary string of characters, such as a program name.

Default: process ID, program name

reply number

Specifies the reply number that appears between the parentheses at the front of a WTOR message event.

For example, the number 12 is required for the REPLY verb to reply to the correct WTOR. When used with the DOM verb and when the REPLY ID is specified, the reply number is prefixed to the matching message text. This causes the correct WTOR to be acknowledged and removed from the held display area of the console of CA Event Manager. The REPLY ID can be omitted if the matching text already contains the REPLY ID.

Default: None

reply text

Specifies a 1- to 255-character string that replies to an outstanding CA Event Manager WTOR.

SEVERITY

Provides an indicator of the type and severity of the message.

Values are:

I = Informational

E = Error

F = Failure

S = Success

W = Warning

Default: None

SOURCE

Identifies the application that generated the event using a string of 1 to 255 characters.

Default: The CA OPS/MVS subsystem name (OPSS)

USERDATA

Specifies a string of 1 to 255 characters that may be used as desired. This field may contain additional data that further explains the meaning of the message text.

USERID

Identifies the user who generated the event using a string of 1 to 32 characters. For CCS for z/OS, this is the security package user ID. For Windows, this is the Domain\User who created the event.

Default: Security package user ID

WORKSTATION

Specifies the name of the workstation using a string of 1 to 255 characters. This field is primarily used by the CA Workload Management application.

Default: None

Keywords Common to All ADDRESS USS Commands

This section explains the keywords for ADDRESS USS.

Important! Command output from any Address USS command will only be returned when the STEM keyword is specified. The associated WAIT keyword is only necessary when a wait time greater or less than the default OCWAIT parameter time is required. When Address USS is executed in an AOF rule type that does not permit waiting for output (all rules types except TOD, REQ, and SCR), the STEM, WAIT, and DELAY keywords are ignored and no output will be returned. In this case an RC=0 simply means that the Address USS command was successfully added to the USS server command queue.

The following keywords *cannot* be abbreviated.

DELAY

Delays the processing of a command by the time specified. The command is not executed until this value expires.

LOG

When the STEM keyword is not specified (that is, output is not returned), the LOG keyword controls whether any output produced by the function is sent to the SYSLOG and OPSLOG. LOG(N) causes the output to be discarded without any logging. LOG(Y) sends the output to SYSLOG and OPSLOG. LOG(Y) is the default. When the STEM keyword is used, the LOG keyword is ignored and no logging occurs.

STEM

Note: This keyword must be specified when a command is issued and a response is expected.

Defines the REXX variable stem that is used to create the variables containing the command output. STEM.0 indicates the number of output variables created. If this keyword is omitted, ADDRESS USS defaults to NOOUTPUT.

An OPS/REXX variable-not a GLOBAL variable-must be specified on the STEM keyword. If a GLOBAL variable is specified, this is checked and an error is produced.

Note: When ADDRESS USS is issued from a rule without the STEM keyword, and the request is successfully queued to the USS server for execution, a return code of 0 is always returned. A CCS for z/OS command or API request may still fail if, for example, CCS for z/OS is not active. Since output from the request was not requested, no indication of this failure is returned.

Example:

```
ADDRESS USS "PING NODE(SYSTEM99)"
```

In this example, the command is passed to the USS server, and a return code of 0 is returned. When the server executes the command, a return code of 1200 is returned by the PING API call, indicating that CA NSM is not active on the node. The 1200 return code is never returned to the ADDRESS USS caller. The STEM and WAIT variables must be coded to see this return code.

WAIT

Specifies a time between 1 second and 604800 seconds to allow ADDRESS USS to wait for command output to be returned. This keyword is only valid when STEM is specified and output is expected.

Note: The OSF USS servers are intended to execute relatively short USS commands. While long WAIT values may be used, we strongly recommend that the WAIT value not be set higher than 600 seconds (10 minutes). Long-running transactions should be run in the background by placing the ampersand (&) symbol at the end of the USS command. However, doing so results in the command output not being returned to the OPS/REXX EXEC that issued the command. Using the OSF USS servers to execute long running transactions may result in other automation that is time-critical to not execute in a timely fashion, since all of the USS servers may be busy executing long-running work. It may also result in the OSF USS execute queue filling up, which in turn may result in ADDRESS USS failures and failing automation.

ADDRESS USS Return Codes

The ADDRESS USS command produces these return codes:

0

The request was completed successfully.

8

A command execution error has occurred.

16

A security failure has occurred.

20

The CA OPS/MVS subsystem is not active.

24

There has been a STEM name error.

32

There has been a system level error.

36

The USS server was not initialized.

40

The USS server is not available.

44

A command length error has occurred.

48

A syntax error has occurred.

52

A message queue error has occurred.

56

No output was received.

60

A receive error has been detected.

64

Obtained storage release failed.

68

A timer failure for delay has occurred.

72

A failure has occurred in sending the command to the USS server.

76

The server message queue address was not found.

USSCODE and USSREASON Variables

USSCODE and USSREASON are OPS/REXX variables that are created when a USS command is executed and an answer is returned. When ADDRESS USS is expected to receive output, and a return code of 8 is returned, see the OPS/REXX variables USSCODE and USSREASON.

For the specific values and meanings of the USSCODE and USSREASON variables, see the following table:

USSCODE	USSREASON	Meaning
0	N/A	Request completed successfully
16	101	USS server command/API not found
16	102	USS server command/API not implemented
40	N/A	USS server command syntax error
1001	N/A	DOM message not found
1001	103	WTOR API timed out while waiting for response
1002	N/A	CCS for z/OS inactive
1200	N/A	CA Event Manager not active on node
1202	N/A	Requested CA Event Manager entity not found
1203	N/A	Invalid API parameters
1204	N/A	CA Event Manager node unreachable
1205	N/A	API function not yet implemented
1299	N/A	Other API error

Note: For the USSCMD verb, a constant of 1000 is added to any non-zero return code from the command. Reason codes are returned as is. Meanings for USS command return and reason codes can be found in the IBM documentation.

ADDRESS WTO—Issue WTO Messages

The ADDRESS WTO host environment complements the OPSWTO command processor, giving you more flexibility in issuing WTO messages. Using the ADDRESS WTO host environment, you can:

- Issue a WTO message synchronously in AOF rules, causing the WTO to be sent when the rule executes. When you send a WTO message by using the OPSWTO command processor in a rule, the OPSWTO command goes to an OSF server. This can cause a slight delay between the rule executing and the WTO being issued.
- Issue multiline WTO messages. The ADDRESS WTO host environment uses REXX stem variables to store individual lines of the message.

Use one of the following formats for an ADDRESS WTO command:

To issue a single-line WTO message:

```
ADDRESS WTO "TEXT(messagetext) keywords"
[AREAD(areaid)]
[CNAME(consolenames)]
[DELAY(delaytime)]
[DESC(desccode)]
[HLITE|LOWLITE]
[MCSFLAGS(flagvalues)]
[MSGID(messageid)]
[NOLOG]
[OPTION(value)]
[REPLY]
[ROUTE(routecode)]
[SUBSYS(ssid)]
[SYSTEM(ALL|EXT|sysnames)]
[SYSWAIT(seconds)]
[TOKEN(dom token)]
[WAIT(waittime)]
[WTOID(wtoid)]
```

To issue a multiline WTO message:

```
ADDRESS WTO "TEXTVAR(stem-name) keywords"
[AREAD(areaid)]
[CNAME(consolenames)]
[DELAY(delaytime)]
[DESC(desccode)]
[HLITE|LOWLITE]
[LINETYPE(linetype)]
[MCSFLAGS(flagvalues)]
[MSGID(messageid)]
[NOLOG]
[OPTION(value)]
[ROUTE(routecode)]
[SUBSYS(ssid)]
[SYSTEM(ALL|EXT|sysnames)]
[SYSWAIT(seconds)]
[TOKEN(adm token)]
[WTOID(wtoid)]
```

Note: Specifying a *wtoid* value for the WTOID keyword is optional.

ADDRESS WTO Keywords

Following lists and describes the ADDRESS WTO optional keywords:

ARE Aid

Defines the console area ID to display the message. You must use this keyword to specify the console to receive the message. You also must specify the DESC keyword with the values 8 and 9.

You cannot use the ARE Aid keyword with the REPLY keyword.

CNAME

Defines the names of the consoles that will receive the WTO or WTOR message. Specify up to 16 alphanumeric console names containing up to eight characters each.

DELAY

Defines how long OPSWTO suspends processing before executing the current command. You can abbreviate DELAY as DL.

Specify any delay time between 1 and 300 seconds. DELAY has no default value.

In any type of AOF rule except for time-of-day (TOD) and request (REQ) rules, specifying the DELAY keyword, the WAIT keyword, or the REPLY keyword automatically converts an ADDRESS WTO instruction to an OPSWTO command processor and sends it to a server for asynchronous processing.

However, when a TOD or REQ rule contains an ADDRESS WTO command that specifies the DELAY, WAIT, or REPLY keywords, CA OPS/MVS permits the ADDRESS WTO command to execute synchronously. The TOD or REQ rule waits until the ADDRESS WTO command completes before it resumes processing.

Note: The use of the REPLY, WAIT, or DELAY keyword in a TOD rule may delay the execution of other TOD rules.

DESC

Defines message descriptor codes for the current message. Descriptor codes are numbers between 1 and 16; codes 1 through 6 and descriptor code 11 are mutually exclusive.

You can assign codes 7 through 10 in combination with any other code, but each code must be unique. If you specify more than one code, separate them with commas as shown in the following example:

DESC(1,7)

The following is a list of additional descriptor codes and their corresponding values:

1-SYSFAIL
2-IMEDACTN
3-EVENACTN
4-SYSSTAT
5-IMEDCMD
6-JOBSTAT
7-APPLPRGM
8-OOLMSG
9-OPERREQ
10-DYNSTAT
11-CRITEVET
12-HILITE

Notes:

- If numeric descriptor codes are used, no leading zeros are permitted.
- There is no support for descriptor codes when issuing a WTO with REPLY. Use of the DESC keyword and the REPLY keyword together results in an error condition and return code 12.

HILITE and LOWLITE

Determines how the WTO or WTOR message displays on the console that receives it. Specify HILITE to send a high-intensity, non-scrollable WTO or WTOR, or specify LOWLITE to make the message scrollable and displayed in low intensity.

LINETYPE

Designates control and label lines in the WTO message.

Possible values for this keyword are:

- C-Message line 1 is a control line.
- CL-Message line 1 is a control line and line 2 is a label line.
- CLL-Message line 1 is a control line and lines 2 and 3 are label lines.
- L-Message line 1 is a label line.
- LL-Message lines 1 and 2 are label lines.

MCSFLAGS

Indicates MCS flag values, which are passed to the operating system. Each MCS flag is a unique character string. Following are sample MCS flags:

MCSFLAGS(REGO)
MCSFLAGS(BRDCST)

You can specify any of these flag values:

- RESP-The WTO is an immediate command response.
- REPLY-The WTO is a reply to a WTOR.
- BRDCST-Broadcast the message to all active consoles.
- HRDCPY-Queue the message for hard copy only.

MSGID

Defines the message ID that prefixes the WTO text. If you omit the MSGID keyword, the OPSWTO command processor uses a default message ID, OPS1371I. Other message IDs can contain up to 10 characters. Message IDs are not padded with blanks.

NOLOG

Suppresses the generation of message OPS1370H, which documents the origin and parameters of the WTO message. NOLOG may only be specified in an AOF rules environment.

OPTION

Performs the same function that the **OPTION** argument of the **OPSEND()** OPS/REXX built-in function does. Its purpose is to precisely route messages to a specific area in a local or cross-system environment.

If you issue the **ADDRESS WTO** command from a message rule (for example, to re-route the message), the route and descriptor codes that are used will be the same as those for the original message that caused the rule to execute. You can override this by specifying the **ROUTE** keyword, **DESC** keyword, or both in the text of the **ADDRESS WTO** command.

WTO messages that arrive at a remote copy of CA OPS/MVS through an MSF link are always logged in OPSLOG; however, the **OPTION** keyword permits you to specify additional processing that should occur for those messages.

Values for **OPTION** are:

- **A**—Specifies that the AOF of the copy of CA OPS/MVS on the receiving system processes the message. When the message is received it can cause rules to execute on the receiving system. However, if those rules use the variable **MSG.SYSID** to determine the system ID of the system on which the message was originally written to the operator, the rules will not execute on the receiving system.

If you specify **A** in any rule other than a **REQ** or **TOD** rule, it is converted to **W**.

You cannot specify **OPTION(A)** if you have specified a value for either the **REPLY** keyword or the **TEXTVAR** keyword.

- **B**—Specifies that the message goes only to the OPSLOG Browse data area. **B** is the default for the **OPTION** keyword.

You cannot specify **OPTION(B)** if you have specified a value for either the **REPLY** keyword or the **TEXTVAR** keyword.

- **C**—Specifies that the message is WTOd on the receiving system, but CA OPS/MVS does not subject it to AOF processing. The values of the **ROUTE**, **DESC**, and **CNNAME** keywords route the message on the receiving system when you specify **C** as the value of **OPTION**.

You cannot specify **OPTION(C)** if you have specified a value for either the **REPLY** keyword or the **TEXTVAR** keyword.

- **L**— Specifies that the message is WTOd on the receiving system as a local message, but CA OPS/MVS does not subject it to AOF processing. The values of the **ROUTE**, **DESC**, and **CNNAME** keywords route the message on the receiving system when you specify **L** as the value of **OPTION**.

You cannot specify **OPTION(L)** if you have specified a value for either the **REPLY** keyword or the **TEXTVAR** keyword.

The SYSNAME column in OPSLOG on the receiving system will indicate the system from which the message was sent, but the SYSLOG/ OPERLOG will show the message as being locally issued on the receiving system. This differs from option C which issues the message as a foreign or reissued WTO on the receiving system.

- W—Tells OPS/REXX to reissue a WTO for the message on the receiving system, and tells the AOF to process that message as though the receiving system had generated it. The value of the MSG.SYSID variable for a rule processing the re-WTOd message is that of the system where the rule is running, because that is where the WTO is issued.

Note: Values A and B are not valid for multi-line WTOs. If any of these values are specified for the OPTION keyword in an MLWTO, the OPS1361S message is issued and a return code of 128 is generated.

REPLY

In any type of AOF rule (except for TOD and REQ rules), specifying the REPLY, WAIT, or DELAY keywords in an ADDRESS WTO instruction automatically converts the ADDRESS WTO instruction to an OPSWTO command processor and sends it to a server for asynchronous processing.

However, when a TOD or REQ rule contains an ADDRESS WTO command that specifies the REPLY, WAIT, or DELAY keywords, CA OPS/MVS permits the ADDRESS WTO command to execute synchronously. The TOD or REQ rule waits until the ADDRESS WTO command completes before it resumes processing.

OPSWTO waits for the operator to respond (for the time specified with the WAIT keyword) and echoes any reply made with the TSO PUTLINE service. This lets a CLIST pick up the reply using TSO SYSOUTTRAP mechanism. For SYSOUTTRAP mechanism details, see the chapter “POI Command Processors.”

Notes:

- Using the REPLY, WAIT, or DELAY keyword in a TOD rule may delay the execution of other TOD rules.
- The REPLY and TEXTVAR keywords are mutually exclusive.
- There is no support for descriptor codes when issuing a WTO with REPLY. Use of the DESC keyword and the REPLY keyword together results in an error condition and return code 12.
- The maximum reply length is 119 characters.

ROUTE

Defines route codes for the message. You can specify as many route codes as you need to, separated by commas.

A route code is any number between 1 and 128 or one of the following character strings:

- MSTRACTN-Equivalent to route code 1
- MSTRINFO-Equivalent to route code 2
- TAPEPOOL-Equivalent to route code 3
- DASDPOOL-Equivalent to route code 4
- TAPELIB-Equivalent to route code 5
- DISKLIB-Equivalent to route code 6
- UR-Equivalent to route code 7
- TP-Equivalent to route code 8
- SECURITY-Equivalent to route code 9
- SYSERR-Equivalent to route code 10
- PROGINFO-Equivalent to route code 11
- EMULATOR-Equivalent to route code 12

Important! Routing codes 30-40 may be specified, but they are considered reserved routing codes by IBM and are ignored.

SUBSYS

Defines the subsystem ID of the system that will receive the WTO message.

SYSTEM

Sends a WTO message to a remote system. Specify one of these values for SYSTEM:

- ALL-All copies of the CA OPS/MVS product that are defined to the MSF issue the WTO message.
- EXT-All remote copies of CA OPS/MVS that are defined to the MSF issue the WTO message.
- sysnames-Specify a list of 1 to 8 MSF-defined system names (remote or local). Each of the listed systems issues the WTO message.

Output is returned only when you specify a single system name as the value of sysnames. Output may be generated if you specify the WTOID or REPLY keyword.

For the SYSTEM keyword to work, MSF must connect the copies of CA OPS/MVS running on the local and remote systems.

Note: Also, see the description of the SYSWAIT keyword.

SYSWAIT

Defines the number of seconds (from 1 to 300) of wait time to accommodate communication delays across systems. CA OPS/MVS adds the value of the SYSWAIT keyword to the MSFSYSWAIT value.

Note: Also see the description of the SYSTEM keyword.

TEXT

Defines the text of a single-line WTO message. This text must meet these criteria:

- Contains up to 124 characters for a WTO or 124 characters for a WTOR message minus the length of the MSGID keyword.
- Uses a pair of double quotes to indicate a single quote in the message text

Following are some sample text strings:

```
TEXT('this is a test value')
TEXT('put quotes in "double" quotes')
TEXT(' provide a leading blank')
```

You can insert the contents of a REXX variable into the message text. For example, the following example inserts the contents of variable FIELD1 into the message text as a text string:

```
TEXT('||FIELD1||')
```

TEXTVAR

Provides the stem for a set of REXX variables, each variable containing one line of a multiline WTO message. You define these variables and assign lines of message text to them in separate statements in the OPS/REXX program that invokes the ADDRESS WTO environment. You are limited to a maximum of 255 lines of message text.

For example, if you specify TEXTVAR(LINE_DATA.), the names of the variables storing lines of your WTO will be LINE_DATA.1, LINE_DATA.2, and so on.

The stem name you specify must meet standard REXX conventions for stem names. If you are not familiar with these conventions, consult *The REXX LANGUAGE: A Practical Approach to Programming* by M.F. Cowlshaw. You can order a copy of the Cowlshaw book from Prentice-Hall.

Instead of a stem, the TEXTVAR keyword can also specify a valid REXX variable name prefix. For example, if you specify TEXTVAR(PREFIX_DATA), the names of variables storing your WTO are PREFIX_DATA1, PREFIX_DATA2, and so on.

The number of lines of output produced depends on the number of consecutive variables that meet the criteria.

Thus, in the following example:

```
LineData.1 = "Line1"
```

```
LineData.2 = "Line2"
```

```
LineData.4 = "Line4"
```

A command specifying ADDRESS WTO TEXTVAR("LineData.") produces only two lines of data output.

Note: The TEXTVAR and REPLY keywords are mutually exclusive.

TOKEN

Defines a one- to four-character name or a one- to four-byte hexadecimal value such as ABCD or X'0A204200'. The token value can be used in delete-operator-message (DOM) processing to delete all messages that have the same token value.

WAIT

Determines how long OPS/REXX waits for a reply to the WTOR message. You can specify any wait time between 1 and 3600 seconds. As soon as the wait time expires, the WTOR message is deleted.

In any type of AOF rule except for time-of-day (TOD) and request (REQ) rules, specifying the WAIT keyword (or the REPLY or DELAY keywords) in an ADDRESS WTO instruction automatically converts the ADDRESS WTO instruction to an OPSWTO command processor and sends it to a server for asynchronous processing. However, when a TOD or REQ rule contains an ADDRESS WTO command that specifies the REPLY, WAIT, or DELAY keywords, CA OPS/MVS permits the ADDRESS WTO command to execute synchronously. The TOD or REQ rule waits until the ADDRESS WTO command completes before it resumes processing.

Note: Using the REPLY, WAIT, or DELAY keyword in a TOD rule may delay the execution of other TOD rules.

WTOID

If you specify the WTOID keyword with no argument, OPS/REXX returns the number for the WTO message into the OPS/REXX external data queue.

The optional *wtoid* argument specifies the name of the variable that will store the WTO ID number. The value returned will be in unprintable binary format.

You can use the REXX C2X function to convert the value to displayable hexadecimal format. For example:

```
C2X(WTOID_VAR_NAME)
```

To convert the value to displayable numeric format, use the REXX C2D function. For example:

```
C2D(WTOID_VAR_NAME)
```

ADDRESS WTO and CA Automation Point

The ADDRESS WTO host environment allows a single-line message to be written to CA Automation Point from an OPS/REXX program. In all cases, only character data may be sent between CA OPS/MVS and CA Automation Point.

Valid characters include all alphanumeric characters, plus *blank*.

? < > : ; , () ~ ^ ` % ! - _ / \ & * \$ # @ ' " = | ' " { } .

Both ADDRESS WTO and OPSWTO will accept all the keywords, regardless of their relevancy to CA Automation Point. CA Automation Point ignores any keywords (for example, ROUTE, DESC) that are not applicable to CA Automation Point processing.

ADDRESS WTO Return Codes

The ADDRESS WTO host environment produces these return codes:

0

The WTO/WTOR completed normally.

4

The WTO/WTOR TEXT keyword is missing.

8

The authorization check failed.

12

The DESC keyword cannot be used with the REPLY keyword.

14

The TEXTVAR keyword cannot be used with the REPLY keyword.

16

The HILITE and LOWLITE keywords are mutually exclusive.

18

The subsystem is not active and CA OPS/MVS is not running.

20

The length of the WTO/WTOR is greater than the allowed maximum.

24

The combination of the message text and MSGID specified exceeded the maximum length allowed.

28

A console ID is required with REG0/QREG0.

32

The WTOR timed out. No reply was received.

36

A WAIT was ended by an attention interrupt.

40

You must specify the CNID keyword in conjunction with the AREAID keyword.

44

You cannot use the AREAID keyword with the REPLY keyword.

48

An invalid value was specified for the ROUTE, DESC, or MCSFLAGS keyword.

52

A CLIST variable access error occurred.

56

A word tokenization error occurred.

80

TSO/E is not installed.

84

A command buffer parse error occurred.

88

The command processor is not authorized.

92

The authorization user exit abended.

96

A serious system macro error occurred.

100

A serious product control block error occurred.

108

You specified an invalid value for the CNNAME keyword.

112

The values specified for the CNNAME keyword is in conflict.

116

The MLWTO TEXTVAR variable was not found.

120

The value of the MLWTO TEXTVAR variable is too long.

122

Cross-system WTO request failed, MSF is not active.

123

Cross-system WTO request failed, none of the SYSIDs in the list are active.

124

A cross-system request cannot be completed. Check for error messages in the external data queue related to this error.

125

The message text for an AP WTO request contains an invalid character.

126

There are too many active remote systems.

127

The output message queue could not be allocated for a cross-system WTO request due to insufficient virtual storage.

128

The OPTION keyword is in error. Either an invalid value was supplied or OPTION A or B was used with the REPLY or TEXTVAR keyword.

132

The SYSTEM keyword value of ALL is mutually exclusive with EXT.

134

The WTOID keyword is not allowed when specifying multiple consoles.

136

The NOLOG keyword may only be specified in AOF rules.

186

There was an abend %1 in the WTO function routine.

Note: Return codes greater than 1000 indicate a problem in the IBM WTO service. Since CA OPS/MVS adds a value of 1000 to IBM WTO return codes, you must subtract 1000 from the ADDRESS WTO return code and then review the IBM documentation for its meaning. IBM documentation normally lists the WTO macro return and reason codes in hexadecimal form.

Usage Notes for ADDRESS WTO

Keep these points in mind when using the ADDRESS WTO host environment:

- Be careful of issuing WTO messages to the local system without using a server, because doing so can create a recursion loop, tying up system resources longer than necessary.
- A non-zero return code is accompanied by a CA OPS/MVS error message in the external data queue.
- Multiline WTO messages (MLWTOs) have these length restrictions:

Line Type	Maximum Length
Control	34
Data	70
Label	70

Notes:

- If you do not specify a value for the LINETYPE keyword in an ADDRESS WTO command, all lines will be treated as data lines.
- You must take into account the length of the message ID when considering the length of the first data line of a multiline WTO. The maximum length of the first data line is 70 characters. This includes the number of characters in the message ID and a blank separator between the message ID and the message text. For example, if the message ID is the 8 character string MESSAGID, then the actual text on the first data line of the MLWTO cannot exceed 61 characters (70 minus 9 (the number of characters in MESSAGID (8) plus the blank separator (1))).

Chapter 3: Relational Data Framework Reference

This chapter provides syntax information on statements, programs, and functions used with the Relational Data Framework component of CA OPS/MVS.

This section contains the following topics:

[SQL Statements](#) (see page 415)

[OPS/REXX Programs](#) (see page 457)

[Relational Table Editor Batch API Functions](#) (see page 460)

SQL Statements

This section provides the syntax for using the CA OPS/MVS SQL commands.

Expressions Used in SQL Statements

The SQL statements described in this chapter are used to perform various functions related to SQL tables and the data in them. These statements consist of a *verb* that enacts the main function (for example, SELECT) along with required and optional arguments. The arguments that you provide in these statements can include clauses, predicates, operators, functions, keywords, and data type definitions.

The following table describes the various types of expressions that may be used in these statements. Additional usage information about these expressions is provided in the syntax pages that follow. Provided with each expression shown in the table is an example of its usage.

Expression	Type	Usage	Example
AND	Operator	Adds conditions to a statement.	WHERE COL1 = 'A' AND COL2 = 'B'
AS	Keyword	Creates aliases for table names.	TABLE1 AS T
AVG	Function	Determines the average value of a column.	AVG(COL1)

Expression	Type	Usage	Example
CHAR	Data Type	Defines a fixed length character-type data column.	ADD COLUMN COL1 CHAR(8)
COUNT	Function	Counts the number of rows in a table.	COUNT (*) FROM TABLE1
DATE	Data Type	Defines a date-type data column.	ADD COLUMN COL2 DATE
DECIMAL	Data Type	Defines a decimal-type data column.	ADD COLUMN COL3 DECIMAL
ESCAPE	Keyword	Causes the next character to be taken literally.	LIKE '10\$%' ESCAPE '\$'
HEX	Data Type	Defines a hexadecimal-type data column.	ADD COLUMN COL4 HEX(2)
IN	Predicate	Matches groups of data.	WHERE COL1 IN ('A','B')
INTEGER	Data Type	Defines an integer-type data column.	ADD COLUMN COL5 INTEGER
LIKE	Predicate	Matches strings to column values.	WHERE COL1 LIKE 'AB%'
MAX	Function	Determines the maximum value in a column.	MAX (COL1)
MIN	Function	Determines the minimum value in a column.	MIN (COL1)
NOT	Operator	Specifies a negative match.	WHERE NOT COL1 = 'A'
OR	Operator	Allows for one of two statements to be true.	WHERE COL1 = 'A' OR COL2 = 'B'

Expression	Type	Usage	Example
SMALLINT	Data Type	Defines a half-word integer-type data column.	ADD COLUMN COL6 SMALLINT
SUBSTR	Function	Allows string selection for comparison.	SUBSTR ('abc' FROM 1 FOR 3)
SUM	Function	Determines the sum of all values in a column.	SUM (COL1)
TIME	Data Type	Defines a time-type data column.	ADD COLUMN COL7 TIME
TIMESTAMP	Data Type	Defines a date/time-type data column.	ADD COLUMN COL8 TIMESTAMP
VARCHAR	Data Type	Defines a variable-length character-type data column.	ADD COLUMN COL1 VARCHAR(300)
WHERE	Clause	Establishes search criteria for a statement.	WHERE COL1 = 'A'
=,<>,<=,>=,<,>	Operator	Used to compare values.	WHERE COL1 >= 2

ALTER TABLE Statement—Add a Table Column

An ALTER TABLE statement, issued with an ADD COLUMN clause, is used to add a column to an existing relational table. A table can contain up to 100 columns.

Note: You can use host variables for *tablename* and *colname*.

- Use this syntax in an AOF rule or an OPS/REXX program:

```
ADDRESS SQL "ALTER TABLE keywords" {tablename}  
ADD COLUMN  
  {colname}  
  {datatype}  
  [DEFAULT string]NULL]  
  [UPPER CASE]  
  [NOT NULL]  
  [SYSTEM(ALL|EXT|sysnames)]  
  [SYSWAIT(seconds)]  
  [OUTPUT|NOOUTPUT]  
  [SYSPLEX]  
  [SUBSYS(subsystem)]
```

- Use this syntax to invoke the statement from a TSO terminal, a TSO/E REXX program, or a TSO CLIST:

```
OPSQL ALTER TABLE {tablename}  
ADD COLUMN  
  {colname}  
  {datatype}  
  [DEFAULT 'string'NULL]  
  [UPPER CASE]  
  [NOT NULL]
```

tablename

Defines the 1- to 18-character name of the relational table to which you are adding a column.

colname

Defines the 1- to 18-character name of the column that you are adding.

datatype

Defines the type of data the column will store.

Data types include:

- CHAR(*length*)-Character data, with *length* as the fixed number of characters you want this column to store.

Note: We recommend using data type CHAR for most purposes, or VARCHAR if the length of the stored data differs from row to row and the full potential length of the column is seldom used. CHAR and VARCHAR are processed more efficiently than other data types.

- DATE-A date indicator of the form *yyyy-mm-dd*; for example, 1994-07-25.
- DECIMAL-Decimal data, with the maximum number of digits being 15.
- HEX(*length*)-Hexadecimal data, with *length* as the maximum number of hexadecimal bytes this column will store.
- INTEGER-Full-word integer data, with a maximum value of 2147483647.
- SMALLINT-Half-word integer data, with a maximum value of 32767.
- TIME(*length*)-A time indicator of the form *hh:mm:ss.nnn*; for example, 13:21:53.876. If a value for *length* is not specified, then the default value of 8 is used.
- TIMESTAMP(*length*)-A date/time indicator, the format being a combination of the DATE and TIME formats. If a value for *length* is not specified, then the default value of 20 is used.
- VARCHAR(*length*)-Character data, with *length* as the maximum number of characters this column will store.

Notes:

- Data of type CHAR is stored in fixed lengths by RDF and padded, as necessary, with blank characters. Data of type VARCHAR is stored in variable lengths by RDF and not padded. Use of the VARCHAR data type saves space in the CA OPS/MVS global variable pool when it is used to store data that is often significantly less than the maximum defined length of a column. VARCHAR data types cannot be defined as primary keys because of their variable length values.
- The DATE, TIME, and TIMESTAMP data types are stored as unsigned packed decimal numbers. When you are inserting, updating, deleting, or searching for values of these types, you must specify the data type along with the value. For example:

WHERE UPDATE = DATE '1994-04-27'

DEFAULT string

(Optional) Indicates the string is a default value to be set for this column if an INSERT statement does not provide a data value. The default string can be either a character string or a numeric string. If you are using the OPSQL command processor to invoke the ALTER TABLE statement, you must enclose the value of string in single quotes.

NULL

(Optional) Indicates that the column can contain no data. This is the default.

UPPER CASE

(Optional) The UPPER CASE keyword in a column definition specifies that the column can contain only uppercase characters. It also instructs SQL to convert any lowercase value inserted into that column to uppercase.

NOT NULL

(Optional) Indicates that the new column cannot contain a null data value.

SYSTEM

(Optional) Performs cross-system SQL operations. Specify one of the following values:

ALL

Routes the SQL command to all active MSF-defined systems, including the local system.

EXT

Routes the SQL command to all remote, active MSF-defined systems.

sysnames

Routes the SQL command to the specified systems. You may specify from one to eight system names as the value of sysnames.

For more information, see the chapter “Using the Relational Data Framework” in the *User Guide*.

SYSWAIT

(Optional) Defines the number of seconds the SQL processor waits for output from a remote system. You may specify a value between 1 and 300 seconds.

Do not specify a value for SYSWAIT if you specify the SYSTEM(ALL), SYSTEM(EXT), or NOOUTPUT keywords.

OUTPUT or NOOUTPUT

(Optional) Indicates whether the command returns output to the external data queue. Specify **OUTPUT** to have output returned; otherwise, specify **NOOUTPUT**.

NOOUTPUT is implied when:

- The **ADDRESS SQL** command is issued in an AOF rule.
- You specify more than one system name as the value of the **SYSTEM(sysnames)** keyword.
- You specify **SYSTEM(ALL)** or **SYSTEM(EXT)**.

SYSPLEX

(Optional) Reduces the scope of the **SYSTEM(ALL|EXT)** operand to MSF connected systems that belong to the same z/OS sysplex as the command issuer. The keyword has no effect on a list of explicit system names.

SUBSYS

(Optional) Performs SQL operations on other CA OPS/MVS subsystems that are in communication with the current one. Specify the following value:

subsystem

Routes the SQL command to the specified subsystem. You can use only one four-character name of the subsystem as follows: OPSx.

Example: Add a Table Column

Add an eight-character column called **OPERNAME**

ADDRESS SQL

"ALTER TABLE SYSTEMS ADD COLUMN OPERNAME CHAR(8) NOT NULL"

CLOSE Statement—End Cursor Operation

Issue a CLOSE statement to end a cursor operation.

Keep this information in mind when using the CLOSE statement:

- The SQL command that executes the CLOSE statement must be issued from an AOF rule, an OPS/REXX program, a TSO/E REXX program, or a TSO CLIST. You cannot invoke SQL commands that perform cursor operations from a TSO terminal.
- It is important to include the CLOSE statement when a cursor operation has ended to release resources.

Use this syntax in an AOF rule or an OPS/REXX program:

```
ADDRESS SQL "CLOSE keywords" {cursorname}  
[SYSTEM{ALL|EXT|sysnames}]  
[SYSWAIT(seconds)]  
[OUTPUT|NOOUTPUT]  
[SYSPLEX]  
[SUBSYS(subsystem)]
```

Use this syntax to invoke the statement from a TSO/E REXX program or a TSO CLIST:

```
OPSQL CLOSE {cursorname}
```

You may specify these operands for the CLOSE statement:

cursorname

Defines the name of the cursor operation you are ending.

SYSTEM

Performs cross-system SQL operations. Specify one of these values:

ALL

Routes the SQL command to all active MSF-defined systems, including the local system.

EXT

Routes the SQL command to all remote, active MSF-defined systems.

sysnames

Routes the SQL command to the specified systems. You may specify from one to eight system names as the value of sysnames.

For more information, see the chapter “Using the Relational Data Framework” in the *User Guide*.

SYSWAIT

Defines the number of seconds the SQL processor waits for output from a remote system. You may specify a value between 1 and 300 seconds.

Do not specify a value for SYSWAIT if you specify the SYSTEM(ALL), SYSTEM(EXT), or NOOUTPUT keywords.

OUTPUT or NOOUTPUT

Indicates whether the command returns output to the external data queue. Specify OUTPUT to have output returned; otherwise, specify NOOUTPUT.

NOOUTPUT is implied when:

- The ADDRESS SQL command is issued in an AOF rule.
- You specify more than one system name as the value of the SYSTEM(*sysnames*) keyword.
- You specify SYSTEM(ALL) or SYSTEM(EXT).

SYSPLEX

(Optional) Reduces the scope of the SYSTEM(ALL|EXT) operand to MSF connected systems that belong to the same z/OS sysplex as the command issuer. The keyword has no effect on a list of explicit system names.

SUBSYS

(Optional) Performs SQL operations on other CA OPS/MVS subsystems that are in communication with the current one. Specify the following value:

subsystem

Routes the SQL command to the specified subsystem. You can use only one four-character name of the subsystem as follows: OPSx.

Example: CLOSE CURSTAT Command

To end the cursor operation named CURSTAT, you can issue this command from an OPS/REXX program:

```
ADDRESS SQL  
"CLOSE CURSTAT"
```

CREATE TABLE Statement—Define Relational Table

Issue a CREATE TABLE statement to define a new relational table to CA OPS/MVS.

The CA OPS/MVS product supports up to 1000 relational tables. You can define as many as 100 columns per table, with as many rows as you need. Each row can contain up to 16,000 characters.

To create a table, invoke an SQL statement using the keywords and arguments shown on the following pages. For each column, include a separate column definition clause in your CREATE TABLE statement. You must separate the column definition clauses with commas.

For a list of reserved keywords in SQL statements, see the chapter “Using the Relational Data Framework” in the *User Guide*. When specifying columns for relational tables, do not use these keywords as column names. The list may grow in future releases of the SQL standard component.

When using the CREATE TABLE statement:

- You can use host variables for *tablename* and *colname*.
- The following minimum and maximum lengths apply to columns of the specified data type:

Data Type	Minimum - Maximum
CHAR	1 - 16000
DECIMAL	1 - 15
HEX	1 - 256
TIME	8 - 15
TIMESTAMP	19 - 26
VARCHAR	1 - 16000

Use this syntax in an AOF rule or an OPS/REXX program:

```
ADDRESS SQL "CREATE [GLOBAL TEMPORARY] TABLE keywords"
           {tablename}
           {datatype}
```

```
/* For every column that you define, you may specify one or more of */
/* these keywords. If you use them, specify them after the datatype */
/* keyword, in the closing parentheses. Specify them in the order */
/* shown:                                     */
```

```
[UPPER CASE]
[NOT NULL]
[PRIMARY KEY]
[DEFAULT 'string'NULL]
```

/* You may specify these keywords only *once* for each CREATE TABLE statement: */

```
[SYSTEM{ALL|EXT|sysnames}]
[SYSWAIT(seconds)]
[OUTPUT|NOOUTPUT]
[SYSFLEX]
[SUBSYS(subsystem)]
```

Use this syntax to invoke the statement from a TSO terminal, a TSO/E REXX program, or a TSO CLIST:

```
OPSQL CREATE [GLOBAL TEMPORARY] TABLE
           {tablename(colname datatype)}
```

```
/* For every column that you define, you may specify one or more of */
/* these keywords. If you use them, specify them after the datatype */
/* keyword, in the closing parentheses. Specify them in the order */
/* shown:                                     */
```

```
[UPPER CASE]
[NOT NULL]
[PRIMARY KEY]
[DEFAULT 'string'NULL]
```

You may specify these operands for the CREATE TABLE statement:

tablename

Defines the 1- to 18-character name of the table you are creating.

colname

Defines the 1- to 18-character name of a column that you are defining for this table. For special considerations when naming columns, see Reserved Keywords in SQL Statements in the chapter "Using the Relational Data Framework" in the *User Guide*.

datatype

Defines the type of data the column will store. Data types include:

CHAR(*length*)

Character data, with *length* as the fixed number of characters you want this column to store.

Note: We recommend that the data type CHAR be used.

DATE

A date indicator of the form *yyyy-mm-dd*; for example, 1994-07-25.

DECIMAL

Decimal data, with the maximum number of digits being 15.

HEX(*length*)

Hexadecimal data, with *length* as the maximum number of hexadecimal bytes this column will store.

INTEGER

Full-word integer data, with a maximum value of 2147483647.

SMALLINT

Half-word integer data, with a maximum value of 32767.

TIME(*length*)

A time indicator of the form *hh:mm:ss.nnn*; for example, 13:21:53.876. If a value for *length* is not specified, then the default value of 8 is used.

TIMESTAMP(*length*)

A date/time indicator, the format being a combination of the DATE and TIME formats. If a value for *length* is not specified, then the default value of 20 is used.

VARCHAR(*length*)

Character data, with *length* as the maximum number of characters this column will store.

Notes:

- Data of type CHAR is stored in fixed lengths by RDF and padded, as necessary, with blank characters. Data of type VARCHAR is stored in variable lengths by RDF and not padded. Use of the VARCHAR data type saves space in the CA OPS/MVS global variable pool when it is used to store data that is often significantly less than the maximum defined length of a column. VARCHAR data types cannot be defined as primary keys because of their variable length values.

- The DATE, TIME, and TIMESTAMP data types are stored as unsigned packed decimal numbers. When you are inserting, updating, deleting, or searching for values of these types, you must specify the data type along with the value; for example:

```
WHERE UPDATE = DATE '2004-04-27'
```

GLOBAL TEMPORARY

(Optional) Creates a table definition in the SYSCKH1 DIV data set that will enable you to store relational table data. Include the GLOBAL TEMPORARY operand on a CREATE TABLE statement. Although the table definition is permanent, the table rows are temporary and will be deleted when CA OPS/MVS stops.

Use the following syntax to create a table definition:

```
ADDRESS SQL
"CREATE GLOBAL TEMPORARY TABLE (rest of SQL statement...)"
```

UPPER CASE

(Optional) Indicates that the column can contain only uppercase characters. It also instructs SQL to convert any lowercase value inserted in that column to uppercase. Specify UPPER CASE if you want INSERT and UPDATE statements to always translate data to uppercase before storing that data in the table.

In the CA OPS/MVS version of SQL, all string data comparisons are case-sensitive.

NOT NULL

(Optional) Indicates that the column can never contain a null data value.

PRIMARY KEY

(Optional) Designates a column as the primary key for this table. The primary key parallels the concept of keyed files and improves Relational Data Framework performance.

Note: VARCHAR data type columns cannot be specified as primary keys.

You can define up to 10 columns as the primary key; however, they must be defined sequentially contiguous to each other and their combined length cannot exceed 71 characters. If you do not identify a primary key, the Relational Data Framework stores rows in the order in which you added them. We strongly recommend that you always specify at least one column as the primary key for every table. There is a significant amount of overhead in processing non-keyed tables with the table editor.

When designating more than one column as a primary key, you can enter PRIMARY KEY beside each column definition. Alternately, you can enter the column definitions, then enter PRIMARY KEY(firstcol-lastcol) at the end of the column definitions.

Note: The firstcol and lastcol variables represent the first and last sequentially contiguous columns of the primary key.

When the CA OPS/MVS table editor displays a table definition, you cannot enter new data into a PRIMARY KEY field. To change the contents of this field, you can copy the line that contains it, make changes to the PRIMARY KEY field on the new line, and then delete the original line.

Note: We strongly recommend that you use CHAR data type columns for the primary key.

DEFAULT string

(Optional) The string is a default value to be set for this column if an INSERT statement does not provide a data value. The default string can be either a character string or a numeric string. If you are using the OPSQL command processor to invoke the CREATE TABLE statement, you must enclose the value of string in single quotes.

NULL

(Default) Indicates that the column can contain no data.

SYSTEM

(Optional) Performs cross-system SQL operations. Specify one of these values:

ALL

Routes the SQL command to all active MSF-defined systems, including the local system.

EXT

Routes the SQL command to all remote, active MSF-defined systems.

sysnames

Routes the SQL command to the specified systems. You may specify from one to eight system names as the value of sysnames.

For more information, see the chapter “Using the Relational Data Framework” in the *User Guide*.

SYSWAIT

(Optional) Defines the number of seconds the SQL processor waits for output from a remote system. You may specify a value between 1 and 300 seconds.

Do not specify a value for SYSWAIT if you specify the SYSTEM(ALL), SYSTEM(EXT), or NOOUTPUT keywords.

OUTPUT or NOOUTPUT

(Optional) Indicates whether the command returns output to the external data queue. Specify OUTPUT to have output returned; otherwise, specify NOOUTPUT.

NOOUTPUT is implied when:

- The ADDRESS SQL command is issued in an AOF rule.
- You specify more than one system name as the value of the SYSTEM(sysnames) keyword.
- You specify SYSTEM(ALL) or SYSTEM(EXT).

SYSPLEX

(Optional) Reduces the scope of the SYSTEM(ALL|EXT) operand to MSF connected systems that belong to the same z/OS sysplex as the command issuer. The keyword has no effect on a list of explicit system names.

SUBSYS

(Optional) Performs SQL operations on other CA OPS/MVS subsystems that are in communication with the current one. Specify the following value:

subsystem

Routes the SQL command to the specified subsystem. You can use only one four-character name of the subsystem as follows: OPSx.

Examples: CREATE TABLE

The following examples illustrate the use of CREATE TABLE.

Example 1-Designate a Primary Key

This CREATE TABLE statement sets up the SYSTEMS table. The RECOV_PROC column stores the desired name of a REXX program to invoke when the current and desired states of a system do not match.

ADDRESS SQL

```
"CREATE TABLE SYSTEMS (NAME CHAR(8) PRIMARY KEY,,"
    "CURRENT_STATE CHAR(4),,"
    "DESIRED_STATE CHAR(4),,"
    "RECOV_PROC CHAR(8))"
```

The NAME column stores the system name. Because system names should be unique and provide an easy way to refer to rows in the table, the NAME column is defined as the primary key.

Example 2-Invoke the CREATE TABLE Statement from a TSO CLIST

This TSO CLIST creates a relational table named DAILY_SCHEDULE.

```
PROC 0
CONTROL MSG CONLIST SYMLIST
OPSQL CREATE TABLE DAILY_SCHEDULE      +
(NAME CHAR(8) NOT NULL PRIMARY KEY,    +
EVENT CHAR(4) NOT NULL,                +
TYPE CHAR(1) UPPER CASE,               +
STATUS CHAR(1) UPPER CASE,             +
SCHED_DATE DATE,                       +
SCHED_TIME TIME,                       +
REPEAT_TIME TIME,                      +
RUN_DATE DATE,                        +
RUN_TIME TIME,                        +
END_DATE DATE,                        +
END_TIME TIME,                        +
MAX_CC CHAR(4),                       +
PREREQ1 CHAR(20),                     +
PRETYPE1 CHAR(1),                     +
PREREQ2 CHAR(20),                     +
PRETYPE2 CHAR(1),                     +
PREREQ3 CHAR(20),                     +
PRETYPE3 CHAR(1)) SUB(OPSF)
WRITE &SQLCODE
WRITE &SYSOUTLINE
WRITE &LASTOC
```

Example 3-Case-sensitive Comparison A

In the CA OPS/MVS version of SQL, all string data comparisons are case-sensitive. Examples 3 and 4 deal with case-sensitivity issues.

For example, this sample CREATE TABLE statement sets up the SYSTEMS table:

```
ADDRESS SQL
"CREATE TABLE SYSTEMS (NAME CHAR(8) PRIMARY KEY,,"
"CURRENT_STATE CHAR(4),,"
"DESIRED_STATE CHAR(4),,"
"RECOV_PROC CHAR(8))"
```

Now suppose that the NAME column in the SYSTEMS table contains the lowercase value 'cics'. The following SELECT statement asks SQL to return all columns containing the characters 'cics':

```
SELECT * FROM SYSTEMS WHERE NAME = 'cics'
```

This statement returns the 'cics' value from the NAME column, because it matches the lowercase 'cics' specified on the SELECT statement.

Example 4-Case-sensitive Comparison B

Now suppose that you are still using the CREATE TABLE statement shown in Example 3 and that the NAME column in the SYSTEMS table still contains the lowercase value 'cics'. However, your SELECT statement is as follows:

```
SELECT * FROM SYSTEMS WHERE NAME = 'CICS'
```

This SELECT statement does not return the value 'cics' from the NAME column, because the uppercase search criteria 'CICS' does not match the lowercase column value 'cics'.

DECLARE CURSOR Statement—Define a Cursor

Issue a DECLARE CURSOR statement to define a cursor. You will include a SELECT statement to establish the location in a table where the cursor operation is to occur.

Note: The SQL command that executes the DECLARE CURSOR statement must be issued from an AOF rule, an OPS/REXX program, a TSO/E REXX program, or a TSO CLIST. You cannot invoke SQL commands that perform cursor operations from a TSO terminal.

Use this syntax in an AOF rule or an OPS/REXX program:

```
ADDRESS SQL "DECLARE {cursorname} CURSOR FOR {selectstatement} keywords"
```

```
/* Optional Keywords: */
[SYSTEM(ALL|EXT|sysnames)]
[SYSWAIT(seconds)]
[OUTPUT|NOOUTPUT]
{SYSPLEX}
[SUBSYS(subsystem)]
```

Use this syntax to invoke the statement from a TSO/E REXX program or a TSO CLIST:

```
OPSQL DECLARE {cursorname} CURSOR FOR {selectstatement}
```

You can specify these operands for the DECLARE CURSOR statement:

cursorname

Defines the 1- to 18-character name of the cursor that you are defining.

Insert this cursorname keyword between the words DECLARE and CURSOR.

selectstatement

Defines the statement that contains the selection criteria for the cursor operation. Insert this selectstatement keyword after the word FOR.

For more information, see SELECT Statement in this chapter.

SYSTEM

(Optional) Performs cross-system SQL operations. Specify one of these values:

ALL

Routes the SQL command to all active MSF-defined systems, including the local system.

EXT

Routes the SQL command to all remote, active MSF-defined systems.

sysnames

Routes the SQL command to the specified systems. You may specify from one to eight system names as the value of *sysnames*.

For more information, see Notes on Performing Cross-system SQL Operations in the chapter “Using the Relational Data Framework” in the User Guide.

SYSWAIT

(Optional) Defines the number of seconds the SQL processor waits for output from a remote system. You may specify a value between 1 and 300 seconds.

Do not specify a value for SYSWAIT if you specify the SYSTEM(ALL), SYSTEM(EXT), or NOOUTPUT keywords.

OUTPUT or NOOUTPUT

(Optional) Indicates whether the command returns output to the external data queue. Specify OUTPUT to have output returned; otherwise, specify NOOUTPUT. NOOUTPUT is implied when:

- The ADDRESS SQL command is issued in an AOF rule.
- You specify more than one system name as the value of the SYSTEM(*sysnames*) keyword.
- You specify SYSTEM(ALL) or SYSTEM(EXT).

SYSPLEX

(Optional) Reduces the scope of the SYSTEM(ALL|EXT) operand to MSF connected systems that belong to the same z/OS sysplex as the command issuer. The keyword has no effect on a list of explicit system names.

SUBSYS

(Optional) Performs SQL operations on other CA OPS/MVS subsystems that are in communication with the current one. Specify the following value:

subsystem

Routes the SQL command to the specified subsystem. You can use only one four-character name of the subsystem as follows: OPSx.

Example: SQL command that defines a cursor operation

This example SQL command defines a cursor operation named STATDOWN that selects the application ID, update time, and status from a table named APPLICATIONS when the status is DOWN.

```
ADDRESS SQL
"DECLARE STATDOWN CURSOR FOR",
"SELECT APPLID, UPDATE, STATUS FROM APPLICATIONS",
"WHERE STATUS = 'DOWN'"

```

DELETE FROM Statement—Delete a Row

Issue a DELETE FROM statement in any operation to delete a specified row. In a searched operation, DELETE FROM deletes the row or rows that meet the specified criteria on a SELECT clause. In a cursor operation, DELETE FROM deletes the row currently being processed.

Important! If you use the DELETE FROM statement for a table and do not include a WHERE clause, all rows from the table are deleted. Use the DELETE FROM statement with extreme caution.

Keep this information in mind when using the DELETE FROM statement:

- You can use a host variable for *tablename*.
- If you are invoking a DELETE FROM statement for a cursor operation, the statement must be issued from an AOF rule, an OPS/REXX program, a TSO/E REXX program, or a TSO CLIST. You cannot invoke SQL commands that perform cursor operations from a TSO terminal.

Use this syntax in an AOF rule or an OPS/REXX program:

```
ADDRESS SQL "DELETE FROM keywords"
{tablename}
[WHERE searchcriteria]
[SYSTEM(ALL|EXT|sysnames)]
[SYSWAIT(seconds)]
[OUTPUT|NOOUTPUT]
[SYSEXEC]

```

Use this syntax to invoke the statement from a TSO terminal, a TSO/E REXX program, or a TSO CLIST:

```
OPSQL DELETE FROM {tablename}
[WHERE searchcriteria]

```

You may specify these operands for the DELETE FROM statement:

tablename

Defines the name of the table from which you want to delete rows.

searchcriteria

(Optional) For a searched operation, the criteria on a WHERE clause can be any valid search criteria.

(Optional) For a cursor operation, the criteria on a WHERE clause must include CURRENT OF cursorname, which causes the function to be performed on the current row being processed in the cursorname operation. See the example that follows.

SYSTEM

(Optional) Performs cross-system SQL operations. Specify one of these values:

ALL

Routes the SQL command to all active MSF-defined systems, including the local system.

EXT

Routes the SQL command to all remote, active MSF-defined systems.

sysnames

Routes the SQL command to the specified systems. You may specify from one to eight system names as the value of sysnames.

For more information, see the chapter “Using the Relational Data Framework” in the *User Guide*.

SYSWAIT

(Optional) Defines the number of seconds the SQL processor waits for output from a remote system. You may specify a value between 1 and 300 seconds.

Do not specify a value for SYSWAIT if you specify the SYSTEM(ALL), SYSTEM(EXT), or NOOUTPUT keywords.

OUTPUT or NOOUTPUT

(Optional) Indicates whether the command returns output to the external data queue. Specify OUTPUT to have output returned; otherwise, specify NOOUTPUT.

NOOUTPUT is implied when:

- The ADDRESS SQL command is issued in an AOF rule.
- You specify more than one system name as the value of the SYSTEM(sysnames) keyword.
- You specify SYSTEM(ALL) or SYSTEM(EXT).

SYSPLEX

(Optional) Reduces the scope of the SYSTEM(ALL|EXT) operand to MSF connected systems that belong to the same z/OS sysplex as the command issuer. The keyword has no effect on a list of explicit system names.

SUBSYS

(Optional) Performs SQL operations on other CA OPS/MVS subsystems that are in communication with the current one. Specify the following value:

subsystem

Routes the SQL command to the specified subsystem. You can use only one four-character name of the subsystem as follows: OPSx.

Examples: DELETE FROM

These examples illustrate the use of the DELETE FROM statement:

- Example 1-Using DELETE FROM in a Searched Operation

Suppose that you want to delete from the SYSTEMS table all information associated with the CICS1 system. You can do this by invoking the following statement from OPS/REXX:

```
ADDRESS SQL
"DELETE FROM SYSTEMS WHERE NAME = 'CICS1'"
```

- Example 2-Using DELETE FROM in a Cursor Operation

To delete the current row being processed in a cursor operation named STATDOWN, issue this command from an AOF rule:

```
ADDRESS SQL
"DELETE FROM APPLICATIONS WHERE CURRENT OF STATDOWN"
```

- Example 3-Using DELETE FROM to Empty a Table

When you want to delete a relational table, you must first delete all of the rows in that table. You do this by invoking a DELETE FROM statement that omits the WHERE clause, as shown here:

```
ADDRESS SQL
"DELETE FROM SYSTEMS"
```

Then, to remove the relational table from the DIV data set, invoke a DROP TABLE statement, which is described in the next section.

DROP TABLE Statement—Remove Table from DIV Data Set

Issue a DROP TABLE statement to remove a relational table from the DIV data set. Before you can delete a table, you must first delete all of the rows in the table using the DELETE FROM statement (described in the previous section). This prevents you from accidentally deleting a table that contains information that a rule or an automation procedure needs to use.

Note: You can use a host variable for *tablename*.

Use this syntax in an AOF rule or an OPS/REXX program:

```
ADDRESS SQL "DROP TABLE keywords"  
{tablename}  
[SYSTEM{ALL|EXT|sysnames}]  
[SYSWAIT(seconds)]  
[OUTPUT|NOOUTPUT]  
[SYSPLEX]  
[SUBSYS(subsystem)]
```

Use this syntax to invoke the statement from a TSO terminal, a TSO/E REXX program, or a TSO CLIST:

```
OPSQL DROP TABLE  
{tablename}
```

You may specify these operands for the DROP TABLE statement:

tablename

Defines the 1- to 18-character name of the table to be removed.

SYSTEM

(Optional) Performs cross-system SQL operations. Specify one of these values:

ALL

Routes the SQL command to all active MSF-defined systems, including the local system.

EXT

Routes the SQL command to all remote, active MSF-defined systems.

sysnames

Routes the SQL command to the specified systems. You may specify from one to eight system names as the value of *sysnames*.

For more information, see the chapter “Using the Relational Data Framework” in the *User Guide*.

SYSWAIT

(Optional) Defines the number of seconds the SQL processor waits for output from a remote system. You may specify a value between 1 and 300 seconds.

Do not specify a value for SYSWAIT if you specify the SYSTEM(ALL), SYSTEM(EXT), or NOOUTPUT keywords.

OUTPUT or NOOUTPUT

(Optional) Indicates whether the command returns output to the external data queue. Specify OUTPUT to have output returned; otherwise, specify NOOUTPUT.

NOOUTPUT is implied when:

- The ADDRESS SQL command is issued in an AOF rule.
- You specify more than one system name as the value of the SYSTEM(*sysnames*) keyword.
- You specify SYSTEM(ALL) or SYSTEM(EXT).

SYSPLEX

(Optional) Reduces the scope of the SYSTEM(ALL|EXT) operand to MSF connected systems that belong to the same z/OS sysplex as the command issuer. The keyword has no effect on a list of explicit system names.

SUBSYS

(Optional) Performs SQL operations on other CA OPS/MVS subsystems that are in communication with the current one. Specify the following value:

subsystem

Routes the SQL command to the specified subsystem. You can use only one four-character name of the subsystem as follows: OPSx.

Example: Delete the SYSTEMS Table

The following two statements delete the SYSTEMS table. The first statement deletes all contents of all rows in the table. The DROP TABLE statement then deletes the empty table.

```
ADDRESS SQL
"DELETE FROM SYSTEMS"
```

```
ADDRESS SQL
"DROP TABLE SYSTEMS"
```

FETCH Statement—Retrieve Row Values

Issue a FETCH statement to retrieve the values in the current row of a cursor operation. Usually this statement is executed in a loop controlled by the SQLCODE variable so that CA OPS/MVS processes each row in a cursor operation. For a sample program that uses SQLCODE, see OPS/REXX Program That Demonstrates Cursor Operations in the chapter “Using the Relational Data Framework” in the *User Guide*.

Note: The SQL command that executes the FETCH statement must be issued from an AOF rule, an OPS/REXX program, a TSO/E REXX program, or a TSO CLIST. You cannot invoke SQL commands that do cursor operations from a TSO terminal.

Use this syntax in an AOF rule or an OPS/REXX program:

```
ADDRESS SQL "FETCH keywords"
/* Specify these required keywords in the order shown: */
{cursorname}
{INTO(hostvarlist)}
/* Optional keywords */
[SYSTEM(ALL|EXTsysnames)]
[SYSWAIT(seconds)]
[OUTPUT|NOOUTPUT]
[SYSPLEX]
[SUBSYS(subsystem)]
```

Use this syntax to invoke the statement from a TSO/E REXX program or a TSO CLIST:

```
OPSQL FETCH
/* Specify the following required keywords in the order shown: */
{cursorname}
{INTO(hostvarlist)}
```

You may specify these operands for the FETCH statement:

cursorname

Defines the name of the cursor operation from which you are retrieving values.

hostvarlist

Defines a set of host variable names in which to store the selected column values. Specify a name for each column defined in the SELECT clause of the DECLARE CURSOR statement, and in the same order.

SYSTEM

(Optional) Use the SYSTEM keyword to perform cross-system SQL operations. Specify one of these values:

ALL

Routes the SQL command to all active MSF-defined systems, including the local system.

EXT

Routes the SQL command to all remote, active MSF-defined systems.

sysnames

Routes the SQL command to the specified systems. You may specify from one to eight system names as the value of sysnames.

For more information, see the chapter “Using the Relational Data Framework” in the *User Guide*.

SYSWAIT

(Optional) Defines the number of seconds the SQL processor waits for output from a remote system. You may specify a value between 1 and 300 seconds.

Do not specify a value for SYSWAIT if you specify the SYSTEM(ALL), SYSTEM(EXT), or NOOUTPUT keywords.

OUTPUT or NOOUTPUT

(Optional) Indicates whether the command returns output to the external data queue. Specify OUTPUT to have output returned; otherwise, specify NOOUTPUT.

NOOUTPUT is implied when:

- The ADDRESS SQL command is issued in an AOF rule.
- You specify more than one system name as the value of the SYSTEM(*sysnames*) keyword.
- You specify SYSTEM(ALL) or SYSTEM(EXT).

SYSPLEX

(Optional) Reduces the scope of the SYSTEM(ALL|EXT) operand to MSF connected systems that belong to the same z/OS sysplex as the command issuer. The keyword has no effect on a list of explicit system names.

SUBSYS

(Optional) Performs SQL operations on other CA OPS/MVS subsystems that are in communication with the current one. Specify the following value:

subsystem

Routes the SQL command to the specified subsystem. You can use only one four-character name of the subsystem as follows: OPSx.

Example: SQL Command

To retrieve the values from the example cursor called STATDOWN, issue this SQL command:

```
ADDRESS SQL  
"FETCH STATDOWN INTO (:APPLID,:UPDATE,:STATUS)"
```

INSERT Statement—Insert Table Rows

Issue the INSERT statement to insert new rows into a table. You can define as many rows per table as you need.

Keep this information in mind when using the INSERT statement:

- You can use host variables for *columnlist* or *tablename*.
- The order of the columns in *columnlist* must match the order of the values in *valuelist*. You do not need to match the actual order of the columns in the table with *columnlist*. As long as the order of *columnlist* and *valuelist* match each other, the values will be inserted into the columns properly.
- Unless you choose to specify the QUERY STATEMENT operand, you should provide values for all columns, which are defined in the table as NOT NULL. Otherwise, CA OPS/MVS gives the columns for which you specified no values the default value given on the CREATE TABLE statement defining this table. If you specified no default, the CA OPS/MVS product makes the columns null.

Use this syntax in an AOF rule or an OPS/REXX program:

```
ADDRESS SQL "INSERT keywords"
{INTO}
[[columnlist]]
/* The following keywords are mutually exclusive: */
[VALUES(valuelist)]
[QUERY STATEMENT]
/* Place these keywords at the end of the command if you use them: */
[SYSTEM(ALL|EXT|sysnames)]
[SYSWAIT(seconds)]
[OUTPUT|NOOUTPUT]
[SYSPLEX]
[SUBSYS(subsystem)]
```

Use this syntax to invoke the statement from a TSO terminal, a TSO/E REXX program, or a TSO CLIST:

```
OPSQL INSERT
/* Specify these required keywords in the order shown: */
{INTO}
{tablename}
/* Optional keywords */
[[columnlist]]
/* The following keywords are mutually exclusive: */
[VALUES(valuelist)]
[QUERY STATEMENT]
```

You may specify these operands for the INSERT statement:

tablename

Defines the table to which you are adding rows.

columnlist

(Optional) Defines the column names that store the values specified with the *valuelist* operand. If you specify no *columnlist*, SQL stores the values into the columns in the order in which they were defined on the CREATE TABLE statement for this table.

valuelist

(Optional) Defines the column values for the new row. A value can be any of the following:

- A date
- A time
- A time stamp
- A character string
- A numeric string
- A host variable
- NULL (which sets a null value)

QUERY STATEMENT

(Optional) Retrieves the values to be inserted into the table. This can be any valid search criteria. See the chapter “Using the Relational Data Framework” in the *User Guide*.

SYSTEM

(Optional) Performs cross-system SQL operations. Specify one of these values:

ALL

Routes the SQL command to all active MSF-defined systems, including the local system.

EXT

Routes the SQL command to all remote, active MSF-defined systems.

sysnames

Routes the SQL command to the specified systems. You may specify from one to eight system names as the value of *sysnames*.

For more information, see the chapter “Using the Relational Data Framework” in the *User Guide*.

SYSWAIT

(Optional) Defines the number of seconds the SQL processor waits for output from a remote system. You may specify a value between 1 and 300 seconds.

Do not specify a value for SYSWAIT if you specify the SYSTEM(ALL), SYSTEM(EXT), or NOOUTPUT keywords.

OUTPUT or NOOUTPUT

(Optional) Indicates whether the command returns output to the external data queue. Specify OUTPUT to have output returned; otherwise, specify NOOUTPUT.

NOOUTPUT is implied when:

- The ADDRESS SQL command is issued in an AOF rule.
- You specify more than one system name as the value of the SYSTEM(*sysnames*) keyword.
- You specify SYSTEM(ALL) or SYSTEM(EXT).

SYSPLEX

(Optional) Reduces the scope of the SYSTEM(ALL|EXT) operand to MSF connected systems that belong to the same z/OS sysplex as the command issuer. The keyword has no effect on a list of explicit system names.

SUBSYS

(Optional) Performs SQL operations on other CA OPS/MVS subsystems that are in communication with the current one. Specify the following value:

subsystem

Routes the SQL command to the specified subsystem. You can use only one four-character name of the subsystem as follows: OPSx.

Examples: Using the INSERT statement

These examples illustrate the use of the INSERT statement:

- **Example 1-Inserting Values into Selected Columns**
When your table contains many columns but you need to set values for only a few columns initially, you can insert values into selected columns. When specifying column values with the VALUES clause, specify them in the same order you used to specify columns in the column list. The following sample statement, invoked by OPS/REXX, inserts a row into the SYSTEMS table. This row will have the values CICS1 and DOWN in the NAME and CURRENT_STATE columns respectively:

```
ADDRESS SQL
"INSERT INTO SYSTEMS (NAME, CURRENT_STATE) VALUES ('CICS1' 'DOWN')"
```

- Example 2-Specifying Values Through Host Variables

You also can specify values through host variables. For example, to insert a job name value into a row of the SYSTEMS table, you could use the variable JOBNAME in an AOF rule as follows:

```
JOBNAME = MSG.JOBNAME
ADDRESS SQL
  "INSERT INTO SYSTEMS (NAME) VALUES (:JOBNAME)"
```

- Example 3-Using a Query Statement

Suppose that you have two tables. The APPLICATIONS table contains four columns: APPL_ID, USER_ID, UPDATE, and STATUS. The structure of the NEWAPPS table is identical to that of the APPLICATIONS table. You might want to add a row to the APPLICATIONS table for each of the new applications that have been altered by a certain three users. In this case, you could substitute the VALUES clause with a query statement and invoke your INSERT statement from a TSO CLIST as follows:

```
OPSQL INSERT INTO APPLICATIONS SELECT * FROM NEWAPPS +
  WHERE USER_ID IN ('TSOUSR1','TSOUSR2','TSOUSR3')
```

OPEN Statement—Initiate Cursor Operation

Issue an OPEN statement to initiate a cursor operation. When the OPEN statement is issued, the SELECT criteria for the cursor operation (specified with the DECLARE CURSOR statement) is immediately executed.

Note: The SQL command that executes the OPEN statement must be issued from an AOF rule, an OPS/REXX program, a TSO/E REXX program, or a TSO CLIST. You cannot invoke SQL commands that perform cursor operations from a TSO terminal.

Use this syntax in an AOF rule or an OPS/REXX program:

```
ADDRESS SQL "OPEN keywords"
{cursorname}
[SYSTEM(ALL|EXT|sysnames)]
[SYSWAIT(seconds)]
[OUTPUT|NOOUTPUT]
[SYSFLEX]
[SUBSYS(subsystem)]
```

Use this syntax to invoke the statement from a TSO/E REXX program or a TSO CLIST:

```
OPSQL OPEN {cursorname}
```


You may specify these operands for the OPEN statement:

cursorname

Defines the name of the cursor you are initiating (previously defined using the DECLARE CURSOR statement).

SYSTEM

(Optional) Performs cross-system SQL operations. Specify *one* of these values:

ALL

Routes the SQL command to all active MSF-defined systems, including the local system.

EXT

Routes the SQL command to all remote, active MSF-defined systems.

sysnames

Routes the SQL command to the specified systems. You may specify from one to eight system names.

For more information, see the chapter “Using the Relational Data Framework” in the *User Guide*.

SYSWAIT

(Optional). Defines the number of seconds the SQL processor waits for output from a remote system. You may specify a value between 1 and 300 seconds.

Do not specify a value for SYSWAIT if you specify the SYSTEM(ALL), SYSTEM(EXT), or NOOUTPUT keywords.

OUTPUT or NOOUTPUT

(Optional) Indicates whether the command returns output to the external data queue. Specify OUTPUT to have output returned; otherwise, specify NOOUTPUT.

NOOUTPUT is implied when:

- The ADDRESS SQL command is issued in an AOF rule.
- You specify more than one system name as the value of the SYSTEM(*sysnames*) keyword.
- You specify SYSTEM(ALL) or SYSTEM(EXT).

SYSPLEX

(Optional) Reduces the scope of the SYSTEM(ALL|EXT) operand to MSF connected systems that belong to the same z/OS sysplex as the command issuer. The keyword has no effect on a list of explicit system names.

SUBSYS

(Optional) Performs SQL operations on other CA OPS/MVS subsystems that are in communication with the current one. Specify the following value:

subsystem

Routes the SQL command to the specified subsystem. You can use only one four-character name of the subsystem as follows: OPSx.

Example: Initiate a cursor operation named STATDOWN

To initiate the cursor operation named STATDOWN, issue this command from an OPS/REXX program:

```
ADDRESS SQL  
"OPEN STATDOWN"
```

SELECT Statement—Extract Relational Table Data

Issue a SELECT statement to extract data from a relational table that meets criteria you specify. You can use the SELECT statement by itself or in another statement (then called a subquery).

Note: You can use host variables for *columnlist* or *tablename*.

Use this syntax in an AOF rule or an OPS/REXX program:

```
ADDRESS SQL "SELECT {columnlist#} keywords"
/* The following keyword must follow the optional INTO hostvarlist keywords, */
/* if used. You may specify more than one tablename; if you do, be sure to */
/* separate each one with a comma: */
{FROM tablename}

/* The following, if used, must follow the required columnlist or * keyword: */
[INTO hostvarlist]

/* The following, if used, must follow the required FROM tablename keywords: */
[WHERE searchcriteria]
[ORDER BY colname ASC|DESC]
[SYSTEM(ALL|EXT|sysnames)]
[SYSWAIT(seconds)]
[OUTPUT|NOOUTPUT]
[COLLIST]
[SYSFLEX]
[SUBSYS(subsystem)]
```

Use this syntax to invoke the statement from a TSO terminal, a TSO/E REXX program, or a TSO CLIST:

```
OPSQL SELECT {columnlist#}
/* This keyword must follow the optional INTO hostvarlist keywords (if used). */
/* You may specify more than one tablename; if you do, be sure to separate */
/* each one with a comma: */
{FROM tablename}
/* These keywords, if used, must follow the required columnlist or * keyword: */
[INTO hostvarlist]
/* These keywords, if used, must follow the required FROM tablename keywords: */
[WHERE searchcriteria]
[ORDER BY colname ASC|DESC]
[COLLIST]
[LINESIZE(20-250)]
```

You may specify these operands for the `SELECT` statement:

columnlist

Defines the column names that contain the values you want. Each column name should be separated by a comma. If you specify an asterisk (*), the CA OPS/MVS product selects all columns in the table.

hostvarlist

(Optional) Defines the host variables (TSO, REXX, or global variables) in which to store the selected column values. The CA OPS/MVS product returns data from selected columns from left to right into the corresponding host variables.

You would not use this option in a cursor declaration, as this function would be accomplished through the `FETCH` statement.

tablename

Defines the name of the table from which you are selecting data. If you specify multiple tables for a join operation, separate each distinct table name with a comma. If a blank rather than a comma follows a table name, the table name that follows is considered to be an alias or correlation value of the preceding table. For more information about join operations, see the chapter “Using the Relational Data Framework” in the *User Guide*.

searchcriteria

(Optional) Defines the data CA OPS/MVS looks for as it searches for the data you want to fetch from the table. The CA OPS/MVS product compares the contents of each row in the table to the *searchcriteria* you specify. For a description of search criteria, the chapter “Using the Relational Data Framework” in the *User Guide*. Then, CA OPS/MVS processes those rows whose contents meet the search criteria. For example, if you have a table that stores information about the systems at your site, specify `WHERE NAME='CICS'` on a `SELECT` statement if you want to fetch data associated with the CICS system.

colname

(Optional). Defines the name of the column SQL uses as a guideline for sorting table data that this `SELECT` statement retrieves. If you also specify the `ASC` keyword, SQL sorts the retrieved values in ascending order, starting with the value in the *colname* column that comes first in the alphabet or is numerically lowest. If you specify the `DESC` keyword, SQL sorts the retrieved values in descending order, starting with the value in the *colname* column that comes last in the alphabet or is numerically highest.

SYSTEM

(Optional) Performs cross-system SQL operations. Specify one of these values:

ALL

Routes the SQL command to all active MSF-defined systems, including the local system.

EXT

Routes the SQL command to all remote, active MSF-defined systems.

sysnames

Routes the SQL command to the specified systems. You may specify from one to eight system names as the value of *sysnames*.

For more information, see the chapter “Using the Relational Data Framework” in the *User Guide*.

SYSWAIT

(Optional) Defines the number of seconds the SQL processor waits for output from a remote system. You may specify a value between 1 and 300 seconds.

Do not specify a value for SYSWAIT if you specify the SYSTEM(ALL), SYSTEM(EXT), or NOOUTPUT keywords.

OUTPUT or NOOUTPUT

(Optional) Indicates whether the command returns output to the external data queue. Specify OUTPUT to have output returned; otherwise, specify NOOUTPUT.

NOOUTPUT is implied when:

- The ADDRESS SQL command is issued in an AOF rule.
- You specify more than one system name as the value of the SYSTEM(sysnames) keyword.
- You specify SYSTEM(ALL) or SYSTEM(EXT).

COLLIST

(Optional) This option can be used if SELECT * is specified. It returns column information in three additional variables:

- COLUMN#LIST

Displays a list of column names, in column definition sequence, separated by a single space.

- COLUMN#TYPE

Displays a list of single-character type identifiers, separated by a single space. The positioning of this wordlist corresponds exactly to that of the COLUMN#LIST list, so after a name index is determined, it can be used to position to the type identifier. The identifier codes are described below.

■ COLUMN#KEYS

Displays a list of column names which are defined as primary keys. This list may not correspond to either of the above, and should be referenced separately.

Column Type identifiers

The definitions correspond to the datatype specification at column creation.

- 1 = CHAR
- 2 = VARCHAR
- 3 = HEX
- 4 = INTEGER
- 5 = SMALLINT
- 6 = DECIMAL
- 7 = DATE
- 8 = TIME
- 9 = TIMESTAMP

LINESIZE (20-250)

(Optional) Sets the maximum size of the terminal output lines returned by OPSQL when terminal output is implied by the environment in which the command is executed.

This operand is useful for setting a line size to match a sysout print line when the command is run in a batch TSO TMP with SYSTSPRT assigned to SYSOUT

Valid for: OPSQL POI command

Default for TSO: 79 characters

Default for Netview: 68 characters

SYSPLEX

(Optional) Reduces the scope of the SYSTEM(ALL|EXT) operand to MSF connected systems that belong to the same z/OS sysplex as the command issuer. The keyword has no effect on a list of explicit system names.

SUBSYS

(Optional) Performs SQL operations on other CA OPS/MVS subsystems that are in communication with the current one. Specify the following value:

subsystem

Routes the SQL command to the specified subsystem. You can use only one four-character name of the subsystem as follows: OPSx.

Output from the SELECT Statement

Most SQL statements modify relational tables, and their only output is the SQLCODE variable. However, the SELECT statement returns data stored in selected rows. How this data is treated depends on the source from which you invoked SQL.

■ Output From a TSO Terminal

If you invoke a SELECT statement from a TSO terminal, SQL displays the returned column values on the terminal, one row at a time. A display line can contain up to 250 characters. To fit as much information as possible on one line, SQL either truncates some lengthy character fields to 20 bytes or, if necessary, omits several column values.

If you need to see the complete text of a display line, you can do so through the CA OPS/MVS relational table editor.

To look at all of the rows in a table, you could invoke a SELECT statement from foreground TSO as follows:

```
OPSQL SELECT * FROM tablename
```

■ Output From a TSO/E REXX Program or a TSO CLIST

When you issue the OPSQL command processor from in a TSO/E REXX program or a TSO CLIST, the terminal receives no output. Instead, the selected values are returned as REXX or CLIST variables.

■ Output From an AOF Rule or an OPS/REXX Program

If an AOF rule or any other type of OPS/REXX program invokes a SELECT statement, SQL fetches the values from each selected row and places them into stem variables:

- If the SELECT statement does not contain an INTO clause, the column name becomes the REXX variable stem.
- If the SELECT or FETCH statement includes the INTO clause, the REXX variable stems are those specified on it.

Unless you specify otherwise, SQL returns the column values in host variables. When you want to return a value into a variable or a column that has a different name, include the INTO clause in your SELECT statement. The INTO clause supplies the names of the variables where you want to store the returned values.

For example, suppose that you want to extract from the SYSTEMS table values from the NAME and CURRENT_STATE columns, and you want to store those values in variables called SYSNAME and STATE. You would invoke a SELECT statement like the following:

```
ADDRESS SQL
"SELECT NAME, CURRENT_STATE INTO :SYSNAME :STATE",
"FROM SYSTEMS WHERE NAME='CICS1'"
```

This statement generates these OPS/REXX output variables:

- SYSNAME.0, set to 1 (the number of rows matching the search criteria)
- SYSNAME.1, set to CICS (the value in the NAME column for the selected row)
- STATE.0, set to 1 (the number of rows matching the search criteria)
- STATE.1, set to DOWN (the value in the CURRENT_STATE column for the selected row)
- SQLCODE, set to 0 (indicating that the SELECT statement executed successfully)

Use the FETCH cursor operation statement to select data one row at a time. For more information, refer to the description of the FETCH statement earlier in this chapter.

Examples: SELECT Statement

The following examples illustrate the use of the SELECT statement.

■ Example 1-Selecting Data From All the Columns in a Row

Suppose that you want to fetch the contents of only one row. If you wanted only the data for system CICS1 from the SYSTEMS table, you could use this SELECT statement in an OPS/REXX program:

```
ADDRESS SQL
"SELECT * FROM SYSTEMS WHERE NAME = 'CICS1'"
```

The asterisk indicates that you want SQL to return data from *all* columns in the row where the NAME column has the value CICS1.

■ Example 2-Comparing Two Columns

When using the WHERE clause in an SQL statement, you typically compare a column against a string or host variable. However, you also can compare one column against another. For example, the SYSTEMS table contains system status information, so you can get a list of all the systems where the current state does not equal the desired state. To get this list, invoke the following SELECT statement from OPS/REXX:

```
ADDRESS SQL
"SELECT * FROM SYSTEMS WHERE CURRENT_STATE <> DESIRED_STATE"
```


■ Example 3-Specifying Multiple Search Criteria

When you need to specify multiple search criteria linked by a mixture of AND and OR operators, you may need to enclose some of the criteria in parentheses to ensure the right result. For example, to select row CICS1 or CICS2, but only when the current state of the CICS1 or CICS2 system is DOWN, you could invoke the following statement from OPS/REXX:

```
ADDRESS SQL
"SELECT * FROM SYSTEMS WHERE CURRENT_STATE = 'DOWN',
  "AND (NAME = 'CICS1' OR NAME = 'CICS2')"
```

If you leave out the parentheses, row CICS1 would be selected when its current state is down, but row CICS2 would be selected regardless of its state.

■ Example 4-Retrieving Data From Selected Columns

In the preceding examples, the SELECT statements fetched data from all columns of the selected rows because an asterisk was specified as the column list operand. However, you can instruct CA OPS/MVS to fetch data only from certain columns. The following SELECT statement, which is invoked from an OPS/REXX program, asks CA OPS/MVS to return only the data stored in the CURRENT_STATE column.

```
ADDRESS SQL
"SELECT CURRENT_STATE FROM SYSTEMS WHERE NAME = 'CICS'"
```

■ Example 5-Inserting Retrieved Data Into a REXX Variable

When instructing CA OPS/MVS to fetch data from a certain column in a certain row, you can insert the retrieved data into a REXX variable. To retrieve the value FIXIMS and place it in the REXX variable RECOV, you could use this REXX statement:

```
ADDRESS SQL
"SELECT RECOV_PROC INTO RECOV FROM SYSTEMS",
  "WHERE NAME = 'IMS'"
```

The INTO clause identifies the variable that will store the retrieved value, FIXIMS.

■ Example 6-Selecting Multiple Columns

When you want to select multiple columns from a table, you can invoke a SELECT statement like this from a rule:

```
ADDRESS SQL
"SELECT CURRENT_STATE, DESIRED_STATE FROM SYSTEMS"
```

■ Example 7-Using SELECT in a TSO/E REXX Program

You can use the SELECT statement in a TSO/E REXX program, as shown here:

```
OPSQL "SELECT NAME INTO :XYZ FROM COLUMN SUB(OPSF)"
SAY XYZ
DO I = 1 TO XYZ.0
  SAY XYZ.I
END
```

- Example 8-Using SELECT for Cross-System SQL Operations

You can use the `SYSTEM` keyword to select all items from `STCTABLE` on the ZOS1 system. The `SYSWAIT(7)` keyword indicates that you want the SQL processor to wait for 7 seconds for output from ZOS1:

```
ADDRESS SQL
"SELECT * FROM STCTABLE SYSTEM(ZOS1) SYSWAIT(7)"
```

UPDATE Statement—Insert Values

Issue an `UPDATE` statement to insert values into selected columns in a table. The `WHERE` clause identifies the rows in the table that are to be updated. You then specify simple expressions in the `SET` clause to indicate how you want to update the identified column.

Keep this information in mind when using the `UPDATE` statement:

- If you omit the `WHERE` clause, SQL updates the identified columns in all rows.
- You can use host variables for *colname* and *tablename*.
- If you are invoking an `UPDATE` statement for a cursor operation, the statement must be issued from an AOF rule, an OPS/REXX program, a TSO/E REXX program, or a TSO CLIST. You cannot invoke SQL commands that perform cursor operations from a TSO terminal.

Note: If you are updating multiple columns, specify a `SET` clause with multiple *colname* operands and separate the operands with commas.

Use this syntax in an AOF rule or an OPS/REXX program:

```
ADDRESS SQL "UPDATE keywords"
{tablename}
{SET colname=string|hostvar|NULL}
[WHERE searchcriteria]
[SYSTEM(ALL|EXT|sysnames)]
[SYSWAIT(seconds)]
[OUTPUT|NOOUTPUT]
[SYSLEX]
[SUBSYS(subsystem)]
```

Use this syntax to invoke the statement from a TSO terminal, a TSO/E REXX program, or a TSO CLIST:

```
OPSQL UPDATE
{tablename}
{SET colname=string|hostvar|NULL}
[WHERE searchcriteria]
```

You may specify these operands for the UPDATE statement:

tablename

Defines the name of the table you are updating.

colname

Defines the name of the column into which you are inserting data. You can specify the colname value as a string, a host variable name, or as NULL. To update multiple columns, include multiple colname clauses in your UPDATE statement, and separate them with commas.

searchcriteria

(Optional) For a searched operation, the criteria on a WHERE clause can be any valid search criteria. For information about specifying search criteria, see Searched Operations in the chapter “Using the Relational Data Framework” in the User Guide.

(Optional) For a cursor operation, the criteria on a WHERE clause must include CURRENT OF cursorname, which causes the function to be performed on the current row being processed in the cursorname operation.

Note: See the example that follows.

SYSTEM

(Optional) Performs cross-system SQL operations. Specify one of these values:

ALL

Routes the SQL command to all active MSF-defined systems, including the local system.

EXT

Routes the SQL command to all remote, active MSF-defined systems.

sysnames

Routes the SQL command to the specified systems. You may specify from one to eight system names as the value of sysnames.

For more information, see Notes on Performing Cross-system SQL Operations in the chapter “Using the Relational Data Framework” in the User Guide.

SYSWAIT

(Optional). Defines the number of seconds the SQL processor waits for output from a remote system. You may specify a value between 1 and 300 seconds.

Do not specify a value for SYSWAIT if you specify the SYSTEM(ALL), SYSTEM(EXT), or NOOUTPUT keywords.

OUTPUT or NOOUTPUT

(Optional) Indicates whether the command returns output to the external data queue. Specify OUTPUT to have output returned; otherwise, specify NOOUTPUT.

NOOUTPUT is implied when:

- The ADDRESS SQL command is issued in an AOF rule
- You specify more than one system name as the value of the SYSTEM(*sysnames*) keyword
- You specify SYSTEM(ALL) or SYSTEM(EXT)

SYSPLEX

(Optional) Reduces the scope of the SYSTEM(ALL|EXT) operand to MSF connected systems that belong to the same z/OS sysplex as the command issuer. The keyword has no effect on a list of explicit system names.

SUBSYS

(Optional) Performs SQL operations on other CA OPS/MVS subsystems that are in communication with the current one. Specify the following value:

subsystem

Routes the SQL command to the specified subsystem. You can use only one four-character name of the subsystem as follows: OPSx.

Examples: UPDATE

These examples illustrate the use of the UPDATE statement:

- Example 1-Using UPDATE in a Searched Operation

Suppose that you want to alter the SYSTEMS table by changing the current state value for the CICS1 system to UP. To do this, you would use this SQL statement in an AOF rule:

```
ADDRESS SQL
"UPDATE SYSTEMS SET CURRENT_STATE = 'UP' WHERE NAME = 'CICS1'"
```

- Example 2-A Searched Operations That Omits the WHERE Clause

You have one or more rules that execute when CA OPS/MVS starts up, and you want to set the current state of all systems to DOWN. You plan to decide later whether a specific system should be up or down. You could invoke the SQL statement from a rule as follows:

```
ADDRESS SQL
"UPDATE SYSTEMS SET CURRENT_STATE = 'DOWN'"
```

The absence of the WHERE clause from this statement signals that you want to update the CURRENT_STATE column in all rows.

- Example 3-Using UPDATE in a Cursor Operation

To set the value of the STATUS column to UP for the current row in a cursor operation called STATDOWN, you could include this SQL statement in a TSO CLIST:

```
OPSQL UPDATE APPLICATIONS SET STATUS = 'UP'
      WHERE CURRENT OF STATDOWN
```

OPS/REXX Programs

This section provides the syntax for using the READTBL and WRITETBL OPS/REXX programs.

READTBL Program—Read Backup Data Set

The READTBL OPS/REXX program reads the backup data set created by the WRITETBL program and rebuilds the table on the target CA OPS/MVS system using the data from the data set. If you want, you can use the TABLE keyword to give the reconstructed table a new name.

To invoke the READTBL program, issue the following command:

```
OI READTBL
[CMDRESP(TERMINAL|XDQ)]
[DSNAME(datasetname)]
[FND(tablename)]
[LIST]
[SUBSYS(subsystem)]
[TABLE(tablename)]
```

You may specify these keywords when invoking the READTBL program:

CMDRESP

(Optional) Possible values:

TERMINAL

Causes messages to be issued to the terminal or default destination.

XDQ

Causes messages to be placed in the REXX external data queue.

Default: TERMINAL

DSNAME

(Optional) Defines the name of the backup data set. To specify the full data set name, enclose the name with a pair of single quotes, as shown in the following example. If you omit the quotes, the user ID becomes the high-level node.

```
DSNAME('OPS.BACKUP.STCTBL')
```

The backup data set must have the attributes DSORG=PS, RECFM=FB, LRECL=80. In addition, WRITETBL must have been used to offload the table into this data set.

If you specify no data set name, the CA OPS/MVS product uses the default name *userid.subsys.WRITETBL*.

FIND

(Optional) Backup data set is searched for a matching table name. If found, the table is recreated.

LIST

(Optional) Read a backup data set and issue a message for each table name found.

SUBSYS

(Optional). Defines the subsystem name of CA OPS/MVS that contains the desired table.

TABLE

(Optional) Defines the name of the recreated table.

Default: The original name of the table.

WRITETBL Program—Create Backup RDF Table

The WRITETBL OPS/REXX program creates a sequential data set that contains a backup copy of an RDF table. WRITETBL retrieves the definition of the table and row data from SQL. The READTBL OPS/REXX program can then reconstruct the table from this backup data set using SQL on any system where the CA OPS/MVS product runs.

To invoke the WRITETBL program, issue the following command:

```
CI WRITETBL
{TABLE(tablename)}
[BLKSIZE(9040,blocksize)]
[OMDRESP(TERMINAL|XDQ)]
[DSF(SHR|MOD)]
[DROPCOL('columnnames')]
[DSNAME(datasetname)]
[SPACE('pri,sec')]
[SUBSYS(subsystem)]
[UNIT(unitname)]
[VOLSER(volume)]
```

You may specify these keywords when invoking the WRITETBL program:

TABLE

Defines the name of the table to be written to a backup data set. Invoking the WRITETBL program against a table does not change that table.

BLKSIZE

(Optional) Sets the z/OS block size for the backup data set. The value you specify must be a multiple of 80. The default value is 9040.

CMDRESP

(Optional) Possible values:

TERMINAL

Causes messages to be issued to the terminal or default destination.

XDQ

Causes messages to be placed in the REXX external data queue.

Default: TERMINAL

DISP

(Optional) Defines the initial status of the backup data set if the data set already exists. If more than one RDF table is being backed up to the same data set, DISP(MOD) must be specified for all but the first table.

Default: SHR

DROPCOL

(Optional) Defines the names of any columns to be dropped from the backup copy of the table (but not from the original table). If you want to drop more than one column, enclose the list of column names in quotation marks and separate them with at least one blank.

Note: The backup data set will not contain the column definition or the row data for dropped columns.

DSNAME

(Optional) Defines the name of the backup data set. To specify the full data set name, enclose the name with a pair of single quotes as shown in the following example. If you omit the quotes, the user ID becomes the high-level node.

DSNAME('OPS.BACKUP.STCTBL')

The backup data set must have the attributes DSORG=PS, RECFM=FB, LRECL=80.

If you specify no data set name, CA OPS/MVS uses the default name *userid.subsys.WRITETBL*.

SPACE

(Optional) Sets the primary and secondary z/OS space allocation for the backup data set, in blocks.

Default: '50,50'

SUBSYS

(Optional) Defines the subsystem name of CA OPS/MVS that contains the table to be backed up.

UNIT

(Optional) Provides the z/OS unit name for allocating the backup data set.

Default: SYSDA

VOLSER

(Optional) Defines a volume serial number for the backup data set. There is no default volume.

Relational Table Editor Batch API Functions

This section shows the syntax for the API functions provided by the ASOTEAPI OPS/REXX program.

Non-zero Return Values

A non-zero return value from ASOTEAPI should be considered a failure. Error messages may be issued by the REXX SAY statement or by the table editor program. In some cases, an ISPF message number and its variable substitution values for the message may be displayed. To find the message text, look for the ISPF message prefix in the appropriate OPSMLIB member.

COPY Function—Copy Data Rows

The COPY function copies the structure and data rows of a current table to another table. If the target table does not exist, it is created. If the target table does exist, it is deleted and recreated with the new table structure and data.

The COPY function has the following syntax:

```
❯ ASOTEAPI COPY TABLE(from_table_name) TARGET(to_table_name) SSMPROT(YES|NO)
```

The SSMPROT keyword determines whether this potentially destructive operation may be performed on an active System State Manager table. A value of YES prohibits this operation. The default is NO, except for the DELETE, DROPCOLS, and RENAME operations.

Note: To copy the *columns* of a table to another table, use the TRANSFER function.

DELETE Function—Remove Table

The DELETE function removes all the rows from a table and drops the table from the Relational Data Framework so that it no longer exists. The following is the syntax for the DELETE function:

```
❯ ASOTEAPI DELETE TABLE(table_name) SSMPROT(YES|NO)
```

The SSMPROT keyword determines whether the potentially destructive operation will be performed on an active System State Manager resource table or the SSM directory table. A value of NO unconditionally allows the operation. A value of YES prohibits the requested operation against any active SSM table. The default is NO except for the DELETE, DROPCOLS, and RENAME operations.

DROPCOLS Function—Remove Table Columns

The DROPCOLS function removes unwanted columns from a table. The table is then placed into storage, deleted, and recreated without the dropped columns. The list of column names to remove is limited to 200 bytes in length, and each column name is separated by a blank.

The DROPCOLS function has the following syntax:

```
❯ ASOTEAPI DROPCOLS TABLE(table_name) COLUMNS(colname colname...) SSMPROT(YES|NO)
```

The SSMPROT keyword determines whether the potentially destructive operation will be performed on an active System State Manager resource table or the SSM directory table. A value of NO unconditionally allows the operation. A value of YES prohibits the requested operation against any active SSM table. The default is NO except for the DELETE, DROPCOLS, and RENAME operations.

FREE Function—Remove the Global Variable Enqueue

The FREE function removes the global variable enqueue from a table that is being used by another user or orphaned by an unexpected termination that left the enqueue in the held state.

The FREE function has the following syntax:

```
❏ ASOTEAPI FREE TABLE(table_name) {USER(system>userid)}
```

Note: If you use the optional USER keyword, you can specify that the FREE function can only be performed if a system and user name that you specify match the current value of the enqueue holder. If you do not specify the USER keyword, the FREE function is performed unconditionally. This may cause unpredictable results if other users are manipulating the table.

RENAME Function—Rename Tables

The RENAME function renames a table by copying the table to a new table and deleting the old table.

The RENAME function has the following syntax:

```
❏ ASOTEAPI RENAME TABLE(from_table_name) TARGET(to_table_name) SSMPROT(YES|NO)
```

The SSMPROT keyword determines whether the potentially destructive operation will be performed on an active System State Manager resource table or the SSM directory table. A value of NO unconditionally allows the operation. A value of YES prohibits the requested operation against any active SSM table. The default is NO except for the DELETE, DROPCOLS, and RENAME operations.

RENCOLS Function—Rename Column Names

The RENCOLS function renames selected column names of a current table to new column names in a new table. Unchanged columns are copied as is. The columns to be renamed are specified in pairs of old column name and new column name. The total length of the column name list cannot exceed 200 characters.

The RENCOLS function has the following syntax:

```
OI ASOTEAPI RENCOLS TABLE(from_table_name) TARGET(to_table_name)  
      COLUMNS(fromcol tocol fromcol tocol...) SSMPROT(YES|NO)
```

The SSMPROT keyword determines whether the potentially destructive operation will be performed on an active System State Manager resource table or the SSM directory table. A value of NO unconditionally allows the operation. A value of YES prohibits the requested operation against any active SSM table. The default is NO except for the DELETE, DROPCOLS, and RENAME operations.

TRANSFER Function—Copy Matching Column Names

The TRANSFER function copies data from matching column names in a table to the corresponding columns in another table. The columns in the tables are considered to be corresponding if the name and data type match and the columns have the same NOT NULL attribute. Any differences in the length of the columns are padded or truncated, as appropriate.

When the new data rows from the source table are added to the target table, columns in the target table that are *not* matched in the source table are assigned their default values. Unmatched columns in the source table are discarded. If the key field column names do not match or the key values are replicated, there may be duplicate rows. In an ISPF environment, a panel is displayed to allow manual resolution of the duplicate rows. The TRANSFER function fails in a non-ISPF environment.

The TRANSFER function has the following syntax:

```
OI ASOTEAPI TRANSFER TABLE(from_table_name) TARGET(to_table_name) SSMPROT(YES|NO)
```

The SSMPROT keyword determines whether the potentially destructive operation will be performed on an active System State Manager resource table or the SSM directory table. A value of NO unconditionally allows the operation. A value of YES prohibits the requested operation against any active SSM table. The default is NO except for the DELETE, DROPCOLS, and RENAME operations.

UCCCOPY Function—Add Upper Case to Copied Rows

The UCCCOPY function copies the structure and rows of a table to a new table and adds the UPPER CASE attribute to all character-type columns in the new table.

The UCCCOPY function has the following syntax:

```
❏ ASOTEAPI UCCCOPY TABLE(from_table_name) TARGET(to_table_name) SSMPROT(YES|NO)
```

The SSMPROT keyword determines whether the potentially destructive operation will be performed on an active System State Manager resource table or the SSM directory table. A value of NO unconditionally allows the operation. A value of YES prohibits the requested operation against any active SSM table. The default is NO except for the DELETE, DROPCOLS, and RENAME operations.

UCPKCOPY Function—Add Upper Case to Primary Keys

The UCPKCOPY function copies the structure and rows of a table to a new table and adds the UPPER CASE attribute to all character-type columns that are *primary keys* in the new table.

The UCPKCOPY function has the following syntax:

```
❏ ASOTEAPI UCPKCOPY TABLE(from_table_name) TARGET(to_table_name) SSMPROT(YES|NO)
```

The SSMPROT keyword determines whether the potentially destructive operation will be performed on an active System State Manager resource table or the SSM directory table. A value of NO unconditionally allows the operation. A value of YES prohibits the requested operation against any active SSM table. The default is NO except for the DELETE, DROPCOLS, and RENAME operations.

Chapter 4: OPS/REXX Built-in Functions

The OPS/REXX built-in functions are explained this chapter. A chapter table of contents follows.

This section contains the following topics:

[Overview](#) (see page 467)
[DATE Function](#) (see page 468)
[FIND Function](#) (see page 469)
[INDEX Function](#) (see page 471)
[OPSARM Function](#) (see page 471)
[OPSARMST Function](#) (see page 478)
[OPSAUTO Function](#) (see page 485)
[OPSBITS Function](#) (see page 487)
[OPSBN Function](#) (see page 490)
[OPSCA7 Function](#) (see page 490)
[OPSCAWTO Function](#) (see page 492)
[OPSCLEDQ Function](#) (see page 495)
[OPSCOLOR Function](#) (see page 496)
[OPSCPF Function](#) (see page 497)
[OPSCPU Function](#) (see page 499)
[OPSELV Function](#) (see page 501)
[OPSDEV Function](#) (see page 502)
[OPSDOM Function](#) (see page 509)
[OPSDUMP Function](#) (see page 510)
[OPSECURE Function](#) (see page 511)
[OPSENG Function](#) (see page 524)
[OPSGETVL Function](#) (see page 530)
[OPSHFI Function](#) (see page 532)
[OPSINFO Function](#) (see page 533)
[OPSIPL Function](#) (see page 552)
[OPSJES2 Function](#) (see page 559)
[OPSJESX Function](#) (see page 574)
[OPSLIKE Function](#) (see page 598)
[OPSLOG Function](#) (see page 599)
[OPSMTRAP Function](#) (see page 610)
[OPSPDS Function](#) (see page 612)
[OPSPRM Function](#) (see page 616)
[OPSTATUS Function](#) (see page 618)
[OPSTORE Function](#) (see page 631)
[OPSUBMIT Function](#) (see page 632)
[OPSUSS Function](#) (see page 633)
[OPSVALUE Function](#) (see page 641)
[OPSVASRV\(\) Function](#) (see page 654)
[OPSVSAM Function](#) (see page 670)
[OPSWAIT Function](#) (see page 675)
[OPSWLM Function](#) (see page 677)
[OPSWORD Function](#) (see page 680)
[OPSYSPLX Function](#) (see page 681)
[OPSYSSYM Function](#) (see page 684)

[TIME Function—Returns Current or Elapsed Time](#) (see page 686)

[TRACE Function](#) (see page 687)

Overview

The REXX comprehensive set of built-in functions is one of its significant attractions. OPS/REXX supports all standard REXX functions as defined by the second edition of *The REXX Language: A Practical Approach to Programming* by M.F. Cowlshaw, plus functions specifically added for CA OPS/MVS.

Built-in functions operate under OPS/REXX exactly as they do under standard REXX, except for the differences described in this chapter. For information about functions not discussed here, see the second edition of the Cowlshaw book, which you can order from Prentice-Hall.

For more information about the OPS/REXX language in general, and how it differs from and extends standard REXX, see the chapter “Using OPS/REXX” in the *User Guide*.

Error Handling For REXX Functions

Passing an invalid argument to a function usually produces an OPS/REXX execution-time error.

Functions Not Supported in OPS/REXX

OPS/REXX does not support the following standard REXX built-in functions:

- CHARIN
- CHAROUT
- CHARS
- LINEIN
- LINEOUT
- LINES
- S (Scan) option of the TRACE function

Usage Recommendation

Some OPS/REXX built-in functions whose names begin with OPS can only be used in OPS/REXX. If they are used in a CLIST or TSO/E REXX EXEC, they terminate with an S0C3 abend as shown in the following example, which executes a REXX EXEC called TESTPGM that attempts to use the OPSINFO OPS/REXX function:

```
%TESTPGM
System abend code 0C3, reason code 00000003.
Abend in external function OPSINFO.
  46 +++ Temp = OPSINFO("ASID")
Error running TESTPGM, line 46: Incorrect call to routine
READY
```

If you examined the dump for this type of abend, you would find that it always occurs at x'6A' into the CA OPS/MVS function module.

DATE Function

REXX code that uses the DATE function invokes the TSO/E REXX version of the function when running under TSO/E.

Note: Use the DATE function in OPS/REXX or AOF rules.

The OPS/REXX non-SAA extension to the DATE function has the following format:

```
var = DATE([x][,source-date][,y])
```

x

(Optional) Includes, as in standard REXX, the format of the date that is returned by the function and may be any of the following standard REXX date formats: B, C, D, E, J, M, N, O, S, U, W

Default: N

source-date

(Optional) Includes, as in standard REXX, the date to be returned by the function.

Default: The current date at execution time.

y

(Optional) Includes, as in standard REXX, the format of the source-date. However, unlike standard REXX, the only accepted values for y are B and S.

Default: yyyyddd, where ddd is the Julian day of the year yyyy.

This format is useful for calculating past and future dates. For example, the date 1 Feb 2009 in standard REXX format N is expressed as 2009032 in this default format that is unique to OPS/REXX.

Valid DATE Formats

Following are examples of specifying dates, assuming today is March 20, 2009:

Sample Statement	Value Returned
DATE()	20 Mar 2009' (Defaults to today's date)
DATE('D')	'09'
DATE('N','2003080')	'21 Mar 2009'
DATE('U','20030322','S')	'03/22/09'

Usage Notes for the DATE Function

Keep these notes in mind when using the DATE function:

- The standard REXX date format B is the number of *whole* days since and including January 1, 0001. This number does not include the current day, since it is not yet complete. For example, the following code would set *var* to today's date in format B:

```
var = DATE('B') + 1
```

And the following code would set *var* to the date two weeks from today in format B, assuming year end is not crossed:

```
var = DATE('B') + 15
```

- B format conversion decrements the source-date, even if it is supplied in B format. For example, the following code returns 731246 from the OPS/REXX DATE function:

```
var = DATE('B',731247,'B')
```

The same conversion in standard REXX does not decrement the date.

- When converting from B to any other format, standard REXX adds one day to the resulting date, whereas OPS/REXX does not.

FIND Function

REXX code that uses the FIND function invokes the TSO/E REXX version of the function when running under TSO/E.

Note: You can use the FIND function in OPS/REXX or AOF rules.

This function has the following format:

```
var = FIND(string,phrase,startingposition)
```

This function searches *string* for the first occurrence of a blank-delimited *phrase* (which may contain one word) and returns the *word number* of the first word of the phrase in the string. During the search, multiple blanks between words in either strings or phrases are treated as single blanks. If the *phrase* is not found, the FIND function returns 0.

FIND(y,x) is equivalent to the function call WORDPOS(x,y) in TSO/E REXX. In general, use the WORDPOS function instead of FIND. For more information about WORDPOS, see your TSO/E REXX documentation.

Note: The OPS/REXX FIND function supports a *startingposition* argument in its column. The standard REXX FIND function does not support *startingposition*.

Examples: FIND

In these examples, the numbers on the right are the word numbers returned:

Sample Statement	Value Returned
var = FIND('now is the time', 'the')	3
var = FIND('now is the time', 'The')	0
var = FIND('now is the time', 'is the')	2
var = FIND('now is the time', 'is the')	2
var = FIND('now is the time', 'is time')	0
var = FIND('To be or not to be', 'be')	2
var = FIND('To be or not to be', 'be', 3)	6

INDEX Function

The INDEX function returns the position of the first character of *string2* in *string1*. If *string1* does not contain *string2*, OPS/REXX returns 0. The search for *string2* begins at the first character of *string1* unless you specify a *startingposition* for the search.

Note: Use the INDEX function in OPS/REXX or AOF rules. REXX code that uses the INDEX function invokes the TSO/E REXX version of the function when running under TSO/E.

This function has the following format:

```
var = INDEX(string1,string2[,startingposition])
```

INDEX(string1,string2)

Performs the same operation as the POS(*string2*,*string1*) function of TSO/E. For new code, use POS, not INDEX. Both functions support an optional *startingposition* parameter, but the first two arguments are swapped.

OPSARM Function

The OPSARM function allows requests for z/OS automatic restart management (ARM) services to be issued from AOF rules on behalf of the current address space. In addition, OPSARM allows any batch job or started task to request ARM for itself based on its own message traffic instead of internally coded calls to ARM.

Important! If you use OPSARM to issue ARM service requests using message rules, the message *must* originate from the address space that is the target of the ARM request. The only exception to this are requests for ARM abstract elements. You can issue these from any address space and they are identified using the TOKEN parameter. You should not use messages that originate from system address spaces such as MASTER, CONSOLE, or JESx to issue ARM requests.

Note: The meaning and use of the ARM requests are explained in further detail in the ARM section of IBM documentation.

This function has the following format:

```
var = OPSARM (function, elemname,{elemtype},  
             {termtype},{timeout},{starttext},{eventext},{token})
```

Example: OPSARM

In the following example, batch job PAYROLL1 is placed under ARM restart control, for restart only, on the current system. No predecessor elements are defined in the policy.

```
)MSG $HASP373
)NT
armjob =PAYROLL1
)PROC
  if msg.jobname = armjob Then Do
    arc = Opsam('REGISTER',armjob,, 'ELEMTERM')
    if arc < 8 Then Do
      arc = Opsam('READY')
      Say armjob 'HAS REGISTERED WITH ARM'
    End
  Else
    Say "ARM REGISTRATION OF" armjob "FAILED RC="arc, "REASON="opsrecd
  End
  Return 'NORMAL'
)END
```

Values Returned

The OPSARM function returns the return code from the ARM service request in decimal as its returned value. The reason code value from the ARM request is returned in the decimal in a REXX variable named OPSRECD. For the meaning of the ARM return and reason codes, see the IXCARM section of the IBM documentation.

The following values are returned:

RC=0 RE=0

Request completed without any warnings.

RC=4 RE=260

Register request for restart with original start text or JCL.

RC=4 RE=264

Register request for restart with different start text or JCL.

RC=4 RE=516

WAITPRED request timed out. Predecessor did not issue READY within the time limit.

RC=4 RE=772

READY request completed, but some predecessor element has not issued a READY. WAITPRED was not issued when predecessor elements were defined.

RC>4 RE>0

Request failed. See IXCYARM table of return and reason codes. Macro IXCYARM also contains return and reason code meanings.

Additional Variable Output

When a successful REGISTER request is completed ($RC < 8$), additional variable output is created.

The following are the created output variables:

OPSRGTYPE

- 1-Initial REGISTER request performed.
- 2-Restart REGISTER request performed.

OPSRGFLAG

1 byte hex flag indicating special restart conditions:

'80'x-ARM restarts are disabled in this sysplex. Automatic restart on failure is not performed.

'40'x-ASSOCIATE request completed in previous start. Reissue ASSOCIATE request.

'20'x-READY request completed in previous start. Element reached ready state prior to failure causing restart.

Use `BITAND(OPSRGFLAG,'n0'x) = 'n0'x` to test flag for specific values.

OPSRGHOMID

MVS clone ID of original registration.

OPSRGCURID

MVS clone ID of current registration. Compare to home ID to detect cross-system restart.

OPSRMTOKEN

Restart the manager token. The REGISTER request creates and returns this value when registering an abstract element (`TERMTYPE = 'CURSYS'`). Save this value for use in subsequent abstract element requests. The contents of the restart token must not be modified by the application registering the element.

For further information on the register data values displayed above, see the IBM macro `IXCYARAA`.

Function Argument

The function argument describes the ARM service to be performed. The typical sequence of ARM functions is as follows:

1. Issues a REGISTER request when the job or task starts. You must provide a sysplex-unique element name.
2. If the register request indicates that an ARM restart is being performed, issues a WAITPRED request to ensure that it restarted properly, in accordance with the ARM policy in the couple data set.
3. Issues a READY request when the job or task is fully operational.
4. Issues a Deregister request when the job or task terminates as usual and no automatic restart is desired.

The following lists the ARM functions and provides an explanation for each:

REGISTER

This request causes the current job or task to be placed under ARM control for automatic restart if the job or task ends without issuing a Deregister request. You must supply the element name argument. All other arguments are optional.

```
arc = OPSARM('REGISTER','IDMSOVSYS1',, 'ELEMTERM')
```

WAITPRED

During an automatic restart, this function should be issued if this element is dependent on a predecessor element in the ARM policy definition. The job or task waits for the predecessor elements to issue their READY requests or for the policy wait time limit to be exceeded. No other arguments beyond the function may be specified. The exception to this would be the addition of the token for an abstract element.

```
arc = OPSARM('WAITPRED',{opsrmtoken})
```

READY

Indicates that job or task is fully initialized and ready to process work. This is primarily applicable to transaction processing tasks such as CICS. The exception to this would be the addition of the token for an abstract element.

```
arc = OPSARM('READY',{opsrmtoken})
```

Deregister

Terminates ARM control of the job or task. No automatic restart is performed once the job or task deregisters. No other arguments beyond the function may be specified. The exception to this would be the addition of the token for an abstract element.

```
arc = OPSARM('Deregister',{opsrmtoken})
```

ASSOCIATE

This function is provided for a backup job or task such as an XRF task to override the automatic restart of a currently registered element. The task that initially registered the element name is not restarted until the task that issues the associate request deregisters itself. No other arguments beyond the element name may be specified. The exception to this would be the addition of the token for an abstract element.

```
arc = OPSARM('ASSOCIATE',{opsrmtoken})
```

ELEMNAME Argument

The element name argument is a 1- to 16-character name that is unique across the sysplex and is used to uniquely identify a particular task or job. It must be supplied for REGISTER and ASSOCIATE functions.

ELEMTYPE Argument

The element type argument specifies that a REGISTER request may optionally supply a 1- to 8-character element type name to relate the element name to a restart order level in the ARM policy.

EVENTEXIT Argument

The EVENTEXIT argument specifies the name of an event exit load module that is called when a restart event occurs. The module must be in LPA or a linklist library. AOF ARM rules, which run as a system level event exit, can generally preclude the need for this exit.

STARTTEXT Argument

The STARTTEXT argument specifies that, for started task elements, an alternate start command may be supplied for ARM restart events. The text of the command may contain upper and lower case text up to 126 characters in length. AOF ARM rules can also be used to supply start text or alternate JCL. The default is the original start command text.

TERMTYPE Argument

The TERMTYPE argument specifies that the REGISTER request may optionally specify the termination type for which the restart is performed.

ELEMTERM

Restricts restarts to failures on the current system. In the event of a system failure

ALLTERM (Default)

Allows restarts for both the current system and other systems in the sysplex.

SYSTEM

Automatically restarts the element if the system on which the element is registered terminates or is removed from the sysplex, provided that TERMTYPE is set to SYSTEM. However, the element will not restart if it unexpectedly terminates. This applies to standard jobs or started tasks.

CURSYS

Distinguishes an abstract element from a conventional job or started task restart element. These elements are not tied to specific address spaces. The register request and subsequent requests can be issued from any address space and are identified by the TOKEN argument.

TIMEOUT Argument

The TIMEOUT argument specifies the amount of time that ARM waits for a restarted task or job to issue a REGISTER request. NORMAL (the default) defaults to five minutes, while LONG defaults to six hours. ARM policy can set specific time values.

TOKEN Argument

The TOKEN argument performs all ARM functions for an abstract element after the REGISTER call, such as READY and DEREGISTER. This value is of no practical automation use for the job or started task elements since all ARM requests must be issued in the original registered address space.

The REGISTER request creates and returns the TOKEN value in the REXX output variable OPSRMToken. This request requires that you specify the TERMTYPE parameter CURSYS. The OPSRMToken variable and the TOKEN argument values are 32-character hexadecimal numbers.

Example: ARM abstract element REGISTER and READY call

```
arc = opsarm('REGISTER',armjob,,'CURSYS')
if arc < 8 then do
  arc = opsarm('READY',opsmtoken)
  ...
end
```

OPSARMST Function

The OPSARMST function can be used in OPS/REXX, AOF rules, TSO/E REXX, or CLIST to return ARM related information in the form of output stem variables for a unique ARM element name, a particular jobname, an ARM restart group name, a single system, or a full sysplex. The primary use of this function is to determine whether a single job or element name is registered with ARM and its current status and attributes. For tasks or jobs that were registered with ARM using the OPSARM function, the OPSARMST function can be used in any subsequent rule to determine if the element should be deregistered at normal end of job or set to the ready state by some task-issued message.

This command has the following format:

```
var = OPSARMST(selector_name, selector_value,
               {LOCALONLY}, {DETAILS}, keyword arguments)
```

The function arguments describe the type of information retrieval that OPSARMST does.

Possible arguments are:

selector_name

Determines the criteria for retrieving ARM data records from the system. The possible values are:

- ELEMENT-retrieves data for a specific sysplex unique ARM element name designated by the caller at ARM registration.
- JOBNAME-retrieves data for a specific jobname that registered with ARM. More than one ARM element record may be returned since job names are not necessarily unique.
- RESTARTGRP-retrieves data for all elements that are defined in the current ARM policy as belonging to the restart group name specified.
- SYSNAME-retrieves data for all elements that are registered with ARM on a particular z/OS system name.
- ALLDATA-retrieves data for all elements that registered with ARM in the current sysplex configuration.

selector_value

Returns the ARM element name, jobname, restart group name, or system name value for the matching selector name value specified. Wildcard values are not supported. This argument should be null for ALLDATA.

LOCALONLY

(Optional) Limits the scope of ARM data retrieval to the current system only. This argument should only be specified for the JOBNAME and RESTARTGRP selector name arguments.

DETAILS

(Optional) Causes the complete set of output variables to be created for each selected element record. If omitted, then only the global and basic variables are created.

Keywords

(Optional) Contains keyword parameters separated by blanks that control the operation of the function. Possible values are:

- PREFIX(*output variable prefix*)-a REXX stem name or character string that is appended to the beginning of the output variable names. The default is ARMST.
- SUBSYS(*subsys name*)-The subsystem name of the copy of CA OPS/MVS that processes the request of this function for ARM data. This operand should only be needed when OPSARMST is called from a TSO/E REXX program and the default OPSS subsystem name is not active.

Example: OPSARMST

The following example assumes that a CA NetSpy started task was registered with ARM under element name NETSPYPROD using OPSARM when the IEF403I message was issued at task start time:

```
OPSARM('REGISTER','NETSPYPROD','ELEMTERM')
```

CA NetSpy is considered ready for work when the following message is issued:

```
NSY0145 - NETSPY  ACCEPTING  LOGONS - V3R3L0
```

OPSARMST can be used to ensure that the NSY0145 message came from the ARM registered element NETSPYPROD by checking the ASID of the message issuer with the ARM registered element name ASID. If the ASIDs match, then ARM should be notified that NETSPYPROD is ready for work using the OPSARM function.

```
)MSG NSY0145
)PROC
amcnt = OPSARMST('ELEMENT','NETSPYPROD')
if amcnt = 1 & C2X(OPSINFO('ASID')) = armst_asid.1 then
  amrc = OPSARM('READY')
)END
```

Values Returned

The OPSARMST function returns a count of ARM elements that match the selection criteria specified in the function arguments. If an error was encountered executing the function, -1 is returned. If ARM is not active on the system, -2 is returned. When an error is the result of a failure in the IBM IXCQUERY macro call, which provides the ARM data for this function, a diagnostic message containing the return and reason codes from IXCQUERY is also produced. These return and reason codes are documented in the IBM documentation.

Variable Output

Output variables produced by OPSARMST vary in format depending on the variable prefix value specified. ARMST is the default prefix. When the prefix ends with a period, it is used as is to form a true REXX stem variable that is easy to drop. When the prefix does not end with a period, an underscore is appended to the prefix to form multiple REXX stem variables.

Prefix	Resulting Variable Name
ARMST.	ARMST.ARMSTATUS
ARMST	ARMST_ARMSTATUS

Output variables also fall into three classes.

- Global ARM status variables
- Basic ARM element variables
- Detail element variables

Additional information on the meaning of much of the ARM data returned in these variables is contained in the IBM documentation.

Global ARM Variables

The global ARM status variables reflect the overall state of ARM in the sysplex. The following variable names start with the prefix value as explained above:

ARMST_ARMSTATUS

Contains the functional status of ARM regarding its ability to perform restarts of failed elements. Values are ENABLED or DISABLED.

ARMST_ARMCONNECT

Contains the connection status of the ARM couple data set across the sysplex. Values are FULL or WARN when data may not be current due to one or more system failures.

ARMST_ARMELEMENTS

Contains the current count of element names in use by ARM.

ARMST_MAXELEMENTS

Contains the maximum number of element names allowed by the current ARM policy.

Basic Element Variables

The basic ARM element variables contain the most useful information about a particular element.

ARMST_ASID.*n*

Contains the address space number in which the element is currently running or most recently ran in hexadecimal character format.

ARMST_CURRSYS.*n*

Contains the current system on which the element is running.

ARMST_ELEMENT.*n*

Contains the 1- to 16-character name of the ARM element selected.

ARMST_ELEMBIND.*n*

Contains the relationship between the element and the system. The possible values are:

- CURJOB-Applies to standard job and started task restarts. If the batch job or started task represented by the element fails, it needs to be restarted.
- CURSYS-Applies to abstract resources. The application registering with automatic restart management needs to be restarted only when the system fails.

ARMST_ELEMSTATE.*n*

Contains the current ARM state of the element. The possible values are:

- STARTING-Registered but not ready
- AVAILABLE-Registered and ready
- AVAILABLE-TO-Registered and considered ready due to a wait for ready timeout
- FAILED-Failed and not restarted
- RESTARTING-Restarting but not registered
- RECOVERING-Restarted and registered but not ready
- N/A-Status not available

ARMST_INITSYS.*n*

Contains the system on which the element initially registered.

ARMST_JOBTYPE.*n*

Contains the job type of the element is either STC for started tasks or JOB for batch jobs.

ARMST_JOBNAME.*n*

Contains the name of the job or started task. If the element is in a failed or restarting state, N/A is the value.

ARMST_RMTOKEN.*n*

Contains the Restart Manager Token, the 32-character hexadecimal number identifying the abstract element.

ARMST_TOTRESTARTS.*n*

Contains the total number of restarts for this element since initial registration.

Detail Element Variables

The detail element variables give the remaining element information that is generally of less interest.

ARMST.ASSOCBACKING.*n*

When ASSOCELEMENT is not blank, this variable indicates whether the current element is the backup FOR the associated element name or is being backed up BY the associated element. This variable is blank if the current element is not participating in any association. FOR or BY are the only possible non-blank values.

ARMST. ASSOCELEMENT.*n*

The name of the element that the current element is either backed by or is the backup for. To determine which case applies, see the ASSOCBACKING variable. This variable is blank if no ARM associate request has been issued by or for the current element.

ARMST.ASSOCYSNAME.*n*

The name of the system on which the associated element name is running.

ARMST.ELEMTYPE.*n*

The element type name assigned at registration. It is blank if no type was assigned. Element type names determine the restart order that ARM uses for multiple element restarts.

ARMST.EVENTEXIT.*n*

The name of the event exit load module specified on the registration request of the element or blank if no event exit was specified.

ARMST.FREECSA.*n*

The number of kilobytes of CSA that must be available on the target system for this element to be restarted cross-system.

ARMST.FREEECSA.*n*

The number of kilobytes of ECSA that must be available on the target system for this element to be restarted cross-system.

ARMST.FRESTARTDATE.*n*

Date in YYYYMMDD format of the first restart of this element. N/A if the element has never been restarted.

ARMST.FRESTARTTIME.*n*

TIME in HH:MM:SS format of the first restart of the element. N/A if the element has never been restarted.

ARMST.INITCLONE.*n*

The clone ID of the system on which the element initially registered.

ARMST.INTRESTARTS.n

Number of times the element has restarted in the restart interval specified in the current ARM policy.

ARMST.JESGROUP.n

The name of the JES XCF group to which the element belongs.

ARMST.LEVEL.n

Level number of this element determined by the element type from the registration request and the LEVEL parameter in the current ARM policy. Level number determines the sequence in which elements are restarted.

ARMST.LRESTARTDATE.n

Date in YYYYMMDD format of the most recent restart of this element. N/A if the element has never been restarted.

ARMST.LRESTARTTIME.n

Time in HH:MM:SS format of the most recent restart of this element. N/A if the element has never been restarted.

ARMST.MAXRESTARTS.n

Maximum number of restart attempts that ARM allows in the restart interval specified in the current ARM policy. The element is deregistered if this limit is exceeded.

ARMST.OVERRIDE.n

Indicates whether override JCL or command text to be used for restarts exists for this element. The value CMD indicates command text exists, while the value JCL indicates that an alternate JCL stream will be used to restart the job. If neither case exists, then the value is blank. The actual command text or JCL location is available at ARM restart time in AOF ARM rules.

ARMST.PACING.n

The number of seconds that ARM delays the restart of this element when multiple elements in the same restart group are being restarted. This helps reduce system overload when many elements are being restarted.

ARMST.READYTIMEOUT.n

The maximum number of seconds that ARM waits for this element to issue a ready request after a restart. If this time is exceeded, the element is treated as ready but the element status is displayed as AVAILABLE-TO.

ARMST.REGISTERDATE.n

Date in YYYYMMDD format that the element last registered with ARM for the first time or after being deregistered.

ARMST.REGISTERTIME.*n*

Time in HH:MM:SS format that the element last registered with ARM for the first time or after being deregistered.

ARMST.RESTARTGROUP.*n*

The name of the restart group that the element belongs to as determined by the current ARM policy. This value is DEFAULT if the default ARM policy is active or restarts are currently disabled.

ARMST.RESTARTINT.*n*

Number of seconds in the restart interval over which restart attempts are counted and compared to the maximum value in the current ARM policy.

ARMST.RESTARTTIMEOUT.*n*

The maximum number of seconds that ARM waits for this restarted element to reregister with ARM. The element is deregistered if this time is exceeded.

ARMST.TERMTYPE.*n*

An indicator for whether cross-system restarts are allowed.

- ELEMTERM means no cross-system restarts.
- CURSYS indicates an abstract element.
- ALLTERM indicates that the element may be restarted on another eligible system that meets the ARM policy requirements.
- SYSTEM indicates that the element is restarted only when the system on which the element is registered is removed from the sysplex or unexpectedly terminates, but not when the element unexpectedly terminates.

OPSAUTO Function

The OPSAUTO function enables you to query and change the autoenabled status of rule members before CA OPS/MVS is active. Doing so through ADDRESS AOF host commands requires CA OPS/MVS to be active.

Note: You can use OPSAUTO in OPS/REXX and in the PROC section of an AOF TOD rule, but not in any other rule context.

Example: Set autoenabled status

This REXX statement sets the autoenabled status of *mem1* and resets the status of *mem2*. All arguments are converted to uppercase.

```
address TSO
"ALLOC FILE(DD1) DA(OPSTEST.RULES) SHR REUSE"
Old_State_Mem1 = OPSAUTO("S","DD1","mem1")
Old_State_Mem2 = OPSAUTO("R","DD1","mem2")
"FREE FILE(DD1)"
```

OPSAUTO returns a single character indicating the autoenabled status of a member of a rule data set before this function executed. If the function returns the character Y, the member is autoenabled; a returned N indicates that the member is not autoenabled. If the returned value contains more than one character, that value is an error message.

The OPSAUTO function has the following format:

```
var = OPSAUTO(func,ddname,member)
```

func

The *func* code may be one of the following:

I

Inquires about the autoenabled status of the specified member.

R

Resets the autoenabled flag for the specified member.

S

Sets the autoenabled flag for the specified member.

ddname

The *ddname* is the name of a previously allocated ddname; it must represent a partitioned data set. Allocate the rule data set using the ADDRESS OPSDYNAM or TSO ALLOCATE command or an equivalent method.

member

The *member* is the name of any existing member in the data set associated with the *ddname*. This member must have valid ISPF statistics.

Usage Notes for OPSAUTO

Keep these notes in mind when using OPSAUTO:

- The security packages of your system *must* provide security for the rule set.
- The PROC section of AOF time-of-day rules is the only rule section that can use OPSAUTO.
- You can use the OPSAUTO function on any partitioned data set, but it has meaning only for CA OPS/MVS rule data sets.

OPSBITS Function

The OPSBITS function returns a character string whose internal binary representation is all binary zeros except that the bits its arguments specify are on. Bit numbers begin with 1, starting at the leftmost bit, so OPSBITS(1) returns '80'X. The character string returned is as long as needed to hold the rightmost on bit specified. For instance, OPSBITS(16) returns '0001'X, while OPSBITS(17) returns '000080'X.

Note: You can use the OPSBITS function in OPS/REXX or AOF rules.

This function has the following format:

```
var = OPSBITS(bit1[,...bitn])
```

OPSBITS Arguments

OPSBITS can have multiple arguments specified in no particular order.

Example: OPSBITS arguments

OPSBITS(1,2,3) returns 'E0'X, as do OPSBITS(3,2,1) or OPSBITS(2,1,3).

Interpreting Character Strings

Besides interpreting integer values, OPSBITS also interprets character strings into integer values. The pages that follow list these strings and show the integer values into which OPSBITS interprets them.

MCSFLAGS Strings

String	Integer Value
CODES	1

String	Integer Value
RESP	3
MSGTYP	4
REPLY	5
BRDCST	6
HRDCPY	7
NOTIME	9
MLWTO	10
NOLOG	11
EXWPL	12
CMD	13
NOCPY	14

Route Code Strings

String	Integer Value
MSTRACTN	1
MSTRINFO	2
TAPEPOOL	3
DASDPOOL	4
TAPELIB	5
DISKLIB	6
UR	7
TP	8
SECURITY	9
SYSERROR	10
PGMRINFO	11
EMULATOR	12

Descriptor Code Strings

String	Integer Value
SYSFAIL	1
IMEDACTN	2
EVENACTN	3
SYSSTAT	4
IMEDCMD	5
JOBSTAT	6
APPLPRGM	7
OOLMSG	8
OPERREQ	9
DYNSTAT	10
CRITEVET	11
HILITE	12

OPSBITS and Variable Names

In most cases, use quotation marks to enclose variable names specified with OPSBITS, as shown in the following example:

```
msg.desc = opsbits('hilite')
```

You *must* use quotation marks if you use *hilite* as a variable in your OPS/REXX program. The following statements are equivalent to the one immediately above:

```
anyvar = 'hilite'  
msg.desc = opsbits(anyvar)
```

If you are familiar with assembler language, note that OPSBITS(1) is not the same value as the commonly used EQU symbol BIT1 because bit numbers in the assembler language start at 0 rather than 1.

OPSBN Function

The OPSBN function supports the DESC and ROUTCDE operands for the ADD, CHANGE, and DELETE keywords that appear in translated Automate rules.

Note: You can use the OPSBN function in OPS/REXX or AOF rules.

This function has the following format:

```
var = OPSBN(string)
```

string

Contains a binary string suitable for use as a message routing or descriptor code.

Depending on the value of *string*, OPSBN either returns a whole number or a string of whole numbers separated by commas. The input *string* is converted so that bit 0 yields 1, bit 1 yields 2, and so on.

For example, if this is the value of *string*:

```
OPSBN('10010011'B)  □ 1,4,7,8
```

Note: The OPSBN function is not dependent upon CA OPS/MVS being active.

Example: OPSBN function

This example illustrates the use of the OPSBN function to propagate the routing codes of the original message to a replacement message:

```
)MSG SOMEMSG
)PROC
/* Issue a replacement message with the same route code */
address WTO "MSGID(NEWMSGID) TEXT('Just testing')",
         "ROUTE('OPSBN(MSG.ROUTE)')"
```

OPSCA7 Function

The OPSCA7 function lets you issue CA 7 commands from within your OPS/REXX programs. The OPSCA7 function cannot be used in rules.

Note: The OPSCA7 function can be used only in OPS/REXX.

This function has the following format:

```
RC = OPSCA7(["/LOGON userid;command{;cmds};/LOGOFF"])
```

/LOGON userid;

(Optional) This argument allows CA 7 security to authorize a command based on the specified *userid*.

For more information about this argument, see the *CA 7 Security Guide*.

Command{;cmds}

This argument specifies the commands to be issued to CA 7.

OPSCA7 calls the U7SVC program. By default the U7SVC program communicates with instance CA71, but you can specify the parameter CA7=CA7*n* to communicate with another instance.

The same CA7= parameter can be specified in the OPSCA7 function.

```
RC = OPSCA7("CA7=CA7n;/LOGON=userid;your CA7 command;/LOGOFF")
```

Note: For the most current information, see the CA7 documentation.

Example: OPSCA7 function

```
RC = OPSCA7(["/LOGON UID001;DEMAND,JOB=ACTJ001;/LOGOFF"])
```

Values OPSCA7 Returns

OPSCA7 returns the return code from the function call, which is zero if the command was successfully passed to the U7SVC routine. However, a return code of zero does not mean that the command executed successfully. The EDQ may contain information about command syntax errors.

Other possible values include:

806

The program U7SVC was not found in the STEPLIB or LINKLIST data sets.

For more information about U7SVC, see the CA 7 Installation Guide.

28

OPSCA7 was used in a rule. OPSCA7 can only be called from an OPS/REXX program.

40

No valid arguments were passed.

69

Too few arguments were passed.

70

Too many arguments were passed.

OPSCAWTO Function

The OPSCAWTO function can send a message to the CA Event Manager in the form of an SNMP trap. Using Event Manager message rules, CA NSM can take an action.

OPSCAWTO uses the CAIENF component of CCS for z/OS to asynchronously send the SNMP trap. CA NSM does not return a response.

For a list of the CCS for z/OS components required to run OPSCAWTO, see the appendix “CCS for z/OS Component Requirements” in the *Installation Guide*.

Note: You may use the OPSCAWTO function in any OPS/REXX program or AOF rules.

This function has the following format:

```
var = OPSCAWTO('ipaddress','message text',[DE],OID modifier)
```


ipaddress

The TCP/IP address (*ipaddress*) or host name defines the destination node of the TCP/IP network to which the SNMP trap is sent. The OPSCAWTO function receives no notification that the SNMP trap arrived at the specified node successfully. Consult your network communication personnel to obtain the correct IP address or host name of the desired destination.

message text

The message text argument is the text of the message to be sent as an SNMP trap to the desired TCP/IP address. The argument may not contain the quotation mark (") character as part of the sent message data.

Message buffer space must be allocated for fixed length protocol overhead elements, as well as variable length fields which contain the IP address and the OID structure. If the IP address is coded in the basic 15-byte dotted notation format, and the default OID is adequate, the available text length allows up to 432 characters, and might be reduced to 284 if OPSCAWTO is running as a rule. If either the IP address or OID string should change, the available text length may fluctuate. If this limit is exceeded, then the message text is automatically truncated by the amount necessary to stay within the limit. A return code of 1 is returned by the function to indicate that truncation has occurred.

options

(Optional) The options characters can be used to request special handling features of the function. You can specify the following options separately or together in any order as a string:

D (Debug)

The debug option causes the CA Common Services for z/OS ENFSNMPM task to produce detailed trace messages about sending the SNMP trap to the target destination. The trace messages should be your main source of information about problem resolution.

To view the trace messages:

- Display the JESMSG LG ddname or SYSPRINT ddname of the ENFSNMPM started task using CA SYSVIEW or SDSF.
- Verify that the trap message was received, processed, and sent by ENFSNMPM.

Note: If the target host is running the Event Manager component of CA NSM, the CATRAPD service must be running for the trap to appear as a message in the Event Manager GUI console.

E (Enterprise)

The enterprise option inserts the CA OPS/MVS OID (1.3.6.1.4.1.791.2.4.2) into the enterprise ID portion of SNMP trap. If not specified, the default enterprise ID specified for the ENFSNMPM task is used. Specify this option when the receiving SNMP manager uses the enterprise ID of the trap to uniquely identify the sender. An additional 19 characters of the maximum message space is used for this option.

OID modifier

SNMP uses the OID (Object Identifier) to identify the origin and type of information in a message. The default OID in OPSCAWTO is '1.3.6.1.4.1.791.2.4.2'. This OID indicates to OPSCAWTO that the SNMP trap comes from CA OPS/MVS. Other software products that receive and interpret SNMP traps require different or additional identifiers to use the information in a message. The OID modifier argument lets you add on to the '2.4.2' portion of the default OID in OPSCAWTO. The '1.3.6.1.4.1.791.2.4.2' portion of the default OID remains fixed.

Example:

To add on to the '1.3.6.1.4.1.791.2.4.2' portion of the OID, the identifier added requires a starting period. For example, '1.3.6.1.4.1.791.2.4.2' can be changed to '1.3.6.1.4.1.791.2.4.2.3.2.1' by adding '.3.2.1'. We recommend that you create your own OIDs starting with '1.3.6.1.4.1.791.2.4.2.999' to avoid the possibility of future conflicts with CA OPS/MVS OIDs.

Note: When the *OID modifier* argument causes the length of the OID to increase, the space available for the IP address and message text is reduced.

Installation Considerations

The installation and activation of the ENFSNMPM component of CCS for z/OS is described in the installation guide for the CCS for z/OS product. The SNMP data control module (DCM) must be added to the ENF database and the starting of the ENFSNMPM task should be added to the ENF commands or spawn service file of the main ENF task.

Return Codes from OPSCAWTO

The OPSCAWTO function returns one of these codes:

Code	Description
0	The message was successfully passed to CCS for z/OS ENF
1	The message text was truncated and successfully passed to CCS for z/OS ENF
4	CCS for z/OS ENF is inactive
8	CCS for z/OS and ENF logic error
12	CCS for z/OS ENF abend occurred
16	CCS for z/OS ENF unrecoverable error
20	Invalid CCS for z/OS parameter list
40	Function syntax error

OPSCLEDQ Function

The OPSCLEDQ function clears the external data queue (EDQ); it is a short cut to use in place of the following REXX program:

```
DO WHILE QUEUED() > 0
  FULL X
END
RETURN "
```

Note: The OPSCLEDQ function can be used in OPS/REXX or AOF rules.

This function always returns null and has the following format:

```
var = OPSCLEDQ()
```

OPSCOLOR Function

The OPSCOLOR function returns a numeric value and this value is moved into the MSG.COLOR variable while a message rule executes. The color of this message is in the OPSLOG Browse display. Also the OPSCOLOR function returns the color value in text from a single integer or one-byte hexadecimal value argument.

Note: Use the OPSCOLOR function in OPS/REXX or AOF rules. This function is especially useful in AOF rules for changing the color of a message as it appears in OPSLOG Browse.

This function has the following format:

```
var = OPSCOLOR('color' | 0-9 | '00x-09x' )
```

color

Specifies a valid color. Valid colors are NONE, BLUE, WHITE, YELLOW, GREEN, RED, PINK, and TURQ.

Example: OPSCOLOR Function

This example sets a message color (in an MSG rule) to turquoise.

```
)MSG somemsg  
)PROC  
  MSG.COLOR = OPSCOLOR('TURQ')
```

These examples decode the integer or hexadecimal value color codes.

colname=OPSCOLOR(msg.color)	returns 'RED'
colname=OPSCOLOR('03x')	returns 'RED'
colname=OPSCOLOR(3)	returns 'RED'

OPSCPF Function

The OPSCPF function returns information about the z/OS Command Prefix Facility (CPF). It may be used in any environment (that is, AOF rules or OPS/REXX programs).

Various z/OS subsystems define command prefixes to simplify the process of issuing commands and to provide a mechanism for issuing commands on other systems in a sysplex without requiring the use of the z/OS ROUTE command. The z/OS ROUTE command returns data that is similar to the data the z/OS DISPLAY OPDATA command returns. However, you can use the OPSCPF function synchronously in rules or OPS/REXX programs without waiting for a response.

This function has the following format:

```
var = OPSCPF({func[,pfx]][,owner][,system])
```

func

Defines the type of call you want the OPSCPF function to make. Currently, the only valid function call is I, for inquiry about information retrieval. You must always specify the *func* argument. This argument is not case sensitive.

pfx

(Optional) Includes one to eight characters that are matched against the defined CPF prefixes. If the *pfx* matches any of the defined CPF prefixes (using the length of *pfx* for the comparison), those prefixes are returned. If the *pfx* argument is omitted, all CPF information is returned. This argument is not case sensitive.

owner

(Optional) Contains one to eight characters that are matched against the owner of the CPF entry. If the *owner* argument matches any of the defined CPF owner prefixes (using the length of *owner* for the comparison), those CPF records are returned. If the *owner* argument is omitted, all CPF information is returned. Note that the *owner* does not necessarily correspond to the jobname that created the CPF entry. This argument is not case sensitive.

system

(Optional) Contains one to eight characters that are matched against the system on which the CPF entry was defined. If the *system* argument matches the prefix of the system name on which the CPF entry was defined (using the length of *system* for the comparison), those CPF records are returned. If the *system* argument is omitted, all CPF information is returned. This argument is not case sensitive.

Records That OPSCPF Returns

The OPSCPF function places a line in the external data queue (EDQ) for each command prefix it finds, and then returns the number of lines it added to the EDQ. Each line added to the EDQ is formatted as follows:

Word Number	Length	Description
1	8	The CPF prefix string.
2	8	The name of the application that owns the prefix. This is an arbitrary name that the user of CPF services supplies; it does not always correspond to a job name.
3	8	The system to which commands with this prefix are routed.
4	8	The scope of the prefix. Possible values are SYSTEM and SYSPLEX.
5	1	Indicates whether the prefix is removed from the command before it is presented to the target system. Possible values are Y and N.
6	8	The FAILDISP value that was specified when the CPF prefix was defined. Possible values are RETAIN, PURGE, and SYSPURGE. These values indicate the disposition of a defined prefix when the owning address space or system terminates. For further information, see your IBM documentation.

Note: The lengths of the above words may change in a future release, but their relative position (word number) will not change.

Examples: OPSCPF function

- The following sample OPSCPF statement:

```
temp = OPSCPF("I")
do temp
  pull line
  say line
end
```

Produces output similar to the following:

```
$ JES2 XE13 SYSTEM N SYSPURGE
$ JES2 XE03 SYSTEM N SYSPURGE
QA> OPSAOSF XE03 SYSPLEX N PURGE
EA> OPSAECF XE03 SYSTEM N PURGE
```

- The following sample OPSCPF statement returns the number of CPF prefixes that begin with the string “OPS” to the variable temp1:

```
temp1 = OPSCPF("I","OPS") /* Get a count of prefixes starting with OPS */
temp2 = OPSQLEDQ()        /* Clear the EDQ of any OPSCPF output */
```

OPSCPU Function

The OPSCPU function returns information about the processors. You can use this function as follows:

- In OPS/REXX programs or AOF rules
- Only on z/OS 1.6 and higher systems. If used on prior levels of the operating system the function always returns a value of zero and no lines are added to the EDQ.

Note: This function returns data similar to the array of processor data returned by the z/OS DISPLAY M=CPU command. However, you can use the OPSCPU function synchronously in rules or OPS/REXX programs without waiting for a response.

This function has the following format:

```
var = OPSCPU(func)
```

func

Defines the type of call you want the OPSCPU function to make. Currently, the only valid function call is I, for inquiry. You *must always* specify the *func* argument. This argument is not case sensitive.

Records That OPSCPU Returns

The OPSCPU function places a line in the external data queue (EDQ) for each processor, and then returns the number of lines it added to the EDQ. Each line added to the EDQ is formatted as follows:

Word Number	Length	Description
1	3	The processor address as a hexadecimal string (Currently the possible values are 00 - FF)
2	8	The processor type. Possible values are: CPU, ICF, zAAP, zIIP or Unknown
3	8	The processor status. Possible values are: online, offline or notavail
4	3	Whether or not the processor is WLM-managed. Possible values are WLM or *

Word Number	Length	Description
5	10	The CPU ID of the processor (if its status is online) or N/A.

Note: The lengths of the above words may change in a future release and new words may be added after the last documented word, but their relative position (word number) will not change.

Example: OPSCPU function

The following sample OPS/REXX code:

```
processors = OPSCPU("I")
say "Number of processors:" processors
do processors
  pull line
  say line
end
```

Produces output similar to the following:

```
Number of processors: 9
00 CPU   online * 0A08312091
01 CPU   online * 0A08312091
02 CPU   online * 0A08312091
03 CPU   online * 0A08312091
04 zAAP  online * 0A08312091
05 CPU   offline * N/A
06 CPU   offline * N/A
07 zAAP  notavail * N/A
08 zIP   offline * N/A
```


OPSDELV Function

The OPSDELV function deletes only OPS/REXX global variables beginning with a valid global stem prefix (GLOBAL, GLOBALx, GLVTEMPx, or GLVJOBID) and Automate-format variables, and then returns the number of variables that it deleted. It has no effect on other variables. Automate-format variables are defined in the ATMLOCALSCOPE nn and ATMSHAREDScope nn parameters.

Note: The OPSDELV function can be used in OPS/REXX, AOF rules, TSO/E REXX, or CLIST. This function is supplied for compatibility with Automate code. When writing new code, use OPSVALUE instead.

This function has the following format:

```
var = OPSDELV("namemask" [optional keywords])
```

namemask

Specify an exact global variable name for the *namemask* argument, or use the following wildcard characters to select a range of names:

+

Matches any single character in a variable name.

*

Matches any number of trailing characters in a variable name.

The following are valid examples of using the + and * wildcard characters.

- The + wildcard character can be specified anywhere after the first period in a variable name. For example:
 - This is a valid namemask using the + wildcard character: globalx.+
 - This is an invalid namemask using the + wildcard character: global+.x
- The * wildcard character can be specified only at the end of a variable name. For example:
 - This is a valid namemask using the * wildcard character: global.x*
 - This is an invalid namemask using the * wildcard character: Glo+al.*x

keywords

(Optional) Keywords, if present, are separated from the *namemask* and from each other by a space.

For explanations of the keywords you can use with OPSDELV, see OPSDELV Command Processor in the chapter “POI Command Processors.”

Examples: OPSDELV

- This statement does not delete any variables since the variable name is not a global variable name. OPSRC is set to 4 and count is set to 0.

```
count = OPSDELV('MYVARS.*')
```

- This statement removes global variables having the indicated mask. Variables GLOBAL.XYZ1 and GLOBAL.XYZ.1 would both match this example. OPSRC is set to 0 and count is set to the number of variables deleted on the local system.

```
count = OPSDELV('GLOBAL.XYZ*')
```

- This statement removes global variables having the indicated mask. The command is transmitted to all MSF- or CCI-connected systems for execution. Variables GLOBAL.XYZ1 and GLOBAL.XYZ.1 would both match this example. OPSRC is set to 0 and count is set to the number of variables deleted on the local system.

```
count = OPSDELV('GLOBAL.XYZ* SYSTEM(ALL)')
```

For more examples that illustrate how you can use OPSDELV, see OPSDELV Command Processor in the chapter “POI Command Processors.”

OPSDEV Function

The OPSDEV function obtains device information synchronously. Therefore, you can use it in AOF rules.

Note: You can use the OPSDEV function in OPS/REXX or AOF rules.

This function has the following format:

```
var = OPSDEV(({function,qualifier}[.status][.subchannel set])
```

function

Describes the type of information retrieval that OPSDEV does. Possible values are:

U

Extracts device information by UCB. The qualifier is a 3, 4- or 5-digit device number or a mask. When 3- or 4-digit device number is specified the OPSDEV function returns information about the device that is logically in subchannel set 0. Specifying 5-digit device number the OPSDEV function translates it in sdddd format where s is the subchannel set number and dddd is the 4-digit device number. You can use the % character, * character, or both as single character wildcards in the mask. When a mask is specified, information for all device numbers that match the mask is returned. The Subchannel set value cannot be masked. For example, the following REXX statement returns information about all online devices whose device numbers contain 123 in the first three positions:

```
Devices=OPSDEV('U','123*')
```

V

Extracts information by VOLSER. The qualifier is a six-character volume serial. You can use the % as a single character wildcard and the * symbol only as a suffix wildcard. For example, the following REXX statement returns information about all online DASD volumes whose volsers begin with an S and whose third through fifth characters are RES:

```
Devices=OPSDEV('V','S%RES*')
```

Only DASD devices are searched for matching volsers.

D

Extracts information about devices in general. The qualifier determines the type of device about which to return information. The values can be:

- DASD
- TAPE
- UREC (Unit Record devices)
- COMM (Communications devices)
- CTC (Channel to Channel adapters)
- TERM (Terminals)
- (All devices)

For example, this REXX statement returns information about all online Channel to Channel adapters:

```
Devices=OPSDEV('D','CTC')
```

qualifier

Indicates the devices for which information is returned.

status

(Optional) Specifies which devices are to be considered based on online, offline, or unavailable status. Possible values for status are:

O

Only returns information on online devices.

This is the default.

F

Only returns information on offline devices. This includes unavailable devices since unavailable devices are also offline.

U

Only returns information on unavailable devices.

A

Returns information on both online and offline devices, including unavailable devices.

Note: The UNAVAILABLE device status was introduced with z/OS 1.10 and is currently only valid for tape devices on systems running z/OS 1.10 or above. Specifying a status of U for the OPSDEV function on systems prior to z/OS 1.10 results in no devices being returned.

subchannel set

(Optional) Specifies within which subchannel set search for the device. This parameter is valid only for U function and when specifying four-digit qualifier. For the five-digit qualifier this parameter, if specified, is ignored. Possible values for subchannel set are:

0

Search within subchannel set 0.

This is the default.

1

Search within subchannel set 1.

Format of Device Information Returned

The OPSDEV function gets the device information from the Unit Control Block (UCB) and places it in the REXX external data queue. The value returned from OPSDEV is the number of lines placed into the external data queue.

Important! Always use either the REXX WORD function or PARSE instruction to extract the contents of a record from any line.

Each line has the following format:

Word Number: 1

Length: 4

Contents: The device number from the UCB, prefixed with a leading zero. For device number 123, this field would have the value 0123.

Word Number: 2

Length: 6

Contents: The volume serial number from DASD and TAPE devices. For non-DASD and non-TAPE devices, this field contains the character string N/A.

Note: If the volume serial field contains embedded blanks, the blanks are replaced with the underscore character. This may occur when using CA MIA with single character system aliases defined on the DEFSYS statement in MIMINIT. In this case, the pseudo-volser field, which might look like '2=GTA', is displayed by OPSDEV as '2_=GTA'. This preserves the number of words in the output record.

Word Number: 3

Length: 8

Contents: Status of the device (either ONLINE, OFFLINE, or UNAVAIL).

Note: The unavailable device status was introduced with z/OS 1.10 and is currently only valid for tape devices on systems running z/OS 1.10 or above. No tape device can be in the unavailable status on systems prior to z/OS 1.10.

Word Number: 4

Length: 4

Contents: The type of device is one of the following: DASD, TAPE, CTC, COMM, UREC (unit record), TERM (terminal), CONS (console), CHAR (character-oriented device), UNKN (unknown device type).

Word Number: 5

Length: 4

Contents: The qualifier on the type of device is one of the following:

- DASD-3330, 3340, 3350, 3375, 3380, 3390, 9345
- TAPE-3420, 3423, 3480, 348S (3480 tape drive with Automatic Cartridge Loader (ACL) installed), 3490, 349S (3490 tape drive with ACL installed), 3590
- COMM-2701, 2702, 2703, 3705 (including 3725 and 3745 devices), 3791
- Unit Record-1403, 2501, 2540, 3203, 3211, 3505, 3800, 382X, 3890, 3895, 4245, 4248, SWCH, UNKN
- CTC-PCTC (parallel CTC), SCTC (serial CTC), BCTC (basic mode ESCON CTC), RS6K (RS6000 acting like a CTC), 3172 (3172 acting like a CTC), OSA (OSA device), OSAD (OSA diagnostic device), IQD (Internal Queued Device), OSN (OSA NCP DEVICE), FCTC (FICON CTC), UNKN (unknown CTC device)

Word Number: 6

Length: 8

The mount attribute of DASD volumes. Possible values are PUBLIC, PRIVATE, STORAGE, and UNKNOWN. Offline devices are always UNKNOWN. For TAPE devices, this field contains the character string MOUNT-P when a mount is pending or N/A otherwise. For non-DASD and non-TAPE devices, this field contains the character string N/A.

Word Number: 7

Length: 6

Contents: The reserve characteristic of DASD and TAPE volumes. Possible values are RSVD (reserved) and UNRSVD (unreserved). Offline devices are UNRSVD. For non-DASD and non-TAPE devices, this field contains the character string N/A.

Word Number: 8

Length: 4

Contents: The allocation count for a DASD volume. The number represents the number of data sets allocated to the device. The allocation count for offline devices is 0. For TAPE devices, this field contains the character string N/A unless a mount is pending for the device. In that case, this field contains the 4-byte binary WTO ID of the associated z/OS mount message. For non-DASD and non-TAPE devices, this field contains the character string N/A.

Word Number: 9

Length: 4

Contents: The open count for a DASD volume, representing the number of open data sets on the device. The open data set count for offline devices is 0. For TAPE and non-DASD devices, this field contains the character string N/A.

Word Number: 10

Length: 8

Contents: The name of the address space that allocated the TAPE device. For non-TAPE devices, this field contains the character string N/A.

Word Number: 11

Length: 7

Contents: Indicates whether the device is allocated. For online devices, this field contains the string ALLOC if the device is currently allocated or UNALLOC if it is not. For offline devices, this field contains the character string N/A.

Word Number: 12

Length: 8

Contents: Indicates the I/O status of the device. If a hot I/O condition is detected for the device, this field contains the string BOXED. If the device is not ready for I/O, this field contains the string NOTREADY. If an intervention-required message has been issued for the device or is about to be issued, this field contains the string INTREQ. From a technical standpoint, you should consider the device to be in an intervention-required state if this field contains the string NOTREADY or INTREQ. For online devices, this field contains the string OK if none of the above conditions exist. For offline devices, this field contains the character string N/A.

Word Number: 13

Length: 8

Contents: Indicates whether the ACL feature is installed. For TAPE devices, this field contains the string ACLINST if the ACL feature is installed. If the ACL feature is installed and is active, this field contains the string ACLACTIV; if not, it contains the string NOACL. For DASD devices, this field contains the string SMS if the volume is managed by SMS or NOTSMS if it is not. For all other device types or if the TAPE or DASD device is offline, this field contains the character string N/A.

Word Number: 14

Length: 4

Contents: For auto-switch TAPE devices, this field contains the character string AUTS unless the device is assigned to a foreign host. If the device is assigned to a foreign host, this field contains the character string AFH. For online or offline TAPE devices that are not auto-switchable, this field contains the character string NAUT. For all other device types, this field contains the character string N/A.

Word Number: 15

Length: 4

Contents: For TAPE devices that are managed by a device manager, this field contains the character string DEVM. For other online TAPE devices, this field contains the character string NODM. Known device managers include JES3 and CA MIA (when the ASSIGN=(MULTISYSTEM,password) is specified on the GTAINIT statement). Other device managers may exist. For all other device types or if the TAPE device is offline, this field contains the character string N/A.

Word Number: 16

Length: 1

Contents: Subchannel set number for the device number.

Examples: OPSDEV

- This sample OPSDEV statement:

```
say "Count of SYS* volumes =" OPSDEV("V","SYS*")
do while QUEUED() > 0
  pull data
  say data
end
```

Produces output similar to the following:

```
Count of SYS* volumes = 2
02BC SYSRES ONLINE DASD 3380 PRIVATE UNRSVD 87 63...
02BE SYSTST ONLINE DASD 3380 PRIVATE UNRSVD 8 2...
```

- Example of how to use the OPSDEV function to list all online tape devices:

```
OPLines = OPSDEV('D','TAPE')
say "There are" OPLines "online tape devices"
do OPLines
  pull edqline
  say edqline
end
```

- Example of how to use the OPSDEV function to find both online and offline devices by UCB with the subchannel set values specified in UCB value:

```
Dev = OPSDEV('U','05F3*','A')
say "There are" Dev "devices matching request qualifier"
do while QUEUED() > 0
  pull edqline
  say edqline
end
```


OPSDOM Function

The OPSDOM function deletes operator messages. OPSDOM has two arguments: the first is an option character conforming to the standard rules for REXX option characters, and the second is the WTO ID or token value used to delete a message.

Both the token value and the WTO ID value are four bytes long. The caller is responsible for obtaining and passing one of these values.

Note: The OPSDOM function can be used in OPS/REXX or AOF rules.

This function has the following format:

```
var = OPSDOM(option,wto id|token)
```

option

The possible values for the option character are:

D

DOM using a decimal WTO ID value

T

DOM using a four-byte token value

W

DOM using a four-byte binary WTO ID value

Values OPSDOM Returns

Currently, OPSDOM issues no return code. The returned value is always zero.

Example: OPSDOM

Assume that the following statements exist in a MSG rule:

```
GLVTEMP3.WTOID = MSG.WTOID  
GLVTEMP3.TOKEN = MSG.TOKEN
```

Note: You cannot DOM a message in the MSG rule that executed for that message.

One of the following statements may then be coded in a rule or OPS/REXX program that executes at a later time:

```
temp = OPSDOM('D',C2D(GLVTEMP3.WTOID))  
temp = OPSDOM('T',GLVTEMP3.TOKEN)  
temp = OPSDOM('W',GLVTEMP3.WTOID)
```

Note: The TOKEN keyword must be issued under the same address space as the message that is being deleted. Therefore, the utility of this keyword is limited.

OPSDUMP Function

The OPSDUMP function tells CA OPS/MVS to take an SVC dump of the current address space. You can use OPSDUMP only in AOF rules.

This function has the following format:

```
var = OPSDUMP(['title'], ['name'], ['QUIESCE'])
```

title

(Optional) Provides the dump title and contains a string of 1 to 100 characters. If you specify no title, the title is CA OPS/MVS SVC DUMP.

name

(Optional) If you omit this argument, only the HOME address space is dumped. The name also can be one of these values:

OPS

Dumps the CA OPS/MVS address space as well as the HOME address space.

ALL

Dumps the CA OPS/MVS, HOME, PRIMARY, and SECONDARY address spaces.

QUIESCE

(Optional) Sets the system as non-dispatchable while CSA and SQA are being dumped. If you omit this argument, the system continues to run while performing the dump.

Values OPSDUMP Returns

OPSDUMP returns the return code from the SDUMP service routine (zero in most cases). However, when an OPS/REXX program that is not in supervisor state calls OPSDUMP, no SVC dump occurs and the function returns a value of 8.

Example: OPSDUMP

The following example illustrates how you can use the OPSDUMP function to take an SVC dump of an address space that caused an MSG rule to execute. The dump also includes CA OPS/MVS private storage. After executing, the rule disables itself, preventing multiple dumps.

```
)MSG somemsg
)PROC
qq = OPSDUMP("This is just a test of OPSDUMP","OPS")
address AOF "DISABLE" OPSINFO("MAINPGM")
```

OPSECURE Function

The OPSECURE function uses a set of subfunctions to return the following types of information related to CA ACF2, RACF, or CA Top Secret:

- Use subfunction code A to extract data from the standard security control block (ACEE).
- Use subfunction code D to verify data set access.
- Subfunction code F fetches a field from a logon ID record.
- Use subfunction code I to return information about the security product on your system.
- Subfunction code P enables you to validate a user password, set a new one, or both.
- Use subfunction code R to request general resource information.

Note: You can use the OPSECURE function in OPS/REXX or AOF rules.

This function has the following format:

```
var = OPSECURE(subfunction,arguments)
```

You can call OPSECURE only if CA ACF2, RACF, or CA Top Secret are installed and activated. OPSECURE returns the string UNKNOWN SECURITY PRODUCT for the following reasons:

- If no security product is installed
- If the security product has not started yet
- If the security product is started after CA OPS/MVS

If CA OPS/MVS determines that CA ACF2, RACF, or CA Top Secret is installed but not active, OPSECURE returns the string SECURITY PRODUCT INACTIVE.

If the security product was unknown to CA OPS/MVS, issue the following command after the security product is started:

```
F OPSx,RESTART(SEcurity)
```

This will force CA OPS/MVS to recheck and then become aware of the security product. In addition, OPSECURE will return UNKNOWN USER PRODUCT if used in a)SEC rule and the security package is not yet available.

Extract Data from the ACEE

To extract data from the standard security control block (ACEE), issue this version of the OPSECURE function:

```
var = OPSECURE('A','fieldname')
```

Note: All *fieldname* arguments are valid in a RACF system. Different subsets of *fieldname* arguments are valid in CA ACF2 and CA Top Secret.

Fieldname Argument

This argument specifies the type of field information to be returned. You can specify any of the following for fieldname:

Field Name/Description	Valid in CA ACF2?	Valid in CA Top Secret?	Valid in RACF?	Returns...
VERSION: ACEE version field	Yes	Yes	Yes	The ACEE version code as a character string (for example, 2).

Field Name/Description	Valid in CA ACF2?	Valid in CA Top Secret?	Valid in RACF?	Returns...
INSTALLATIONDATA: Installation data	No	Yes	Yes	The installation data as a character string that may contain unprintable characters.
USERDATA: User data field	No	Yes	Yes	The user data as a character string that may contain unprintable characters.
USERID: User ID string	Yes	Yes	Yes	The <i>userid</i> (for example, USERID1).
GROUP: Group name string	Yes	Yes	Yes	The group name (for example, *).
SPECIAL: Special attribute	No	No	Yes	The string 1 if the current user has this attribute or 0 if the current user does not have this attribute.
AUTOMATIC: Automatic data security	No	Yes	Yes	The string 1 if the current user has this attribute or 0 if the current user does not have this attribute.
OPERATIONS: Operations attribute	No	No	Yes	The string 1 if the current user has this attribute or 0 if the current user does not have this attribute.
AUDITOR: Auditor attribute	Yes	No	Yes	The string 1 if the current user has this attribute or 0 if the current user does not have this attribute.
LOG: LOG MOST operation flag	Yes	Yes	Yes	The string 1 if the current user has this attribute or 0 if the current user does not have this attribute.

Field Name/Description	Valid in CA ACF2?	Valid in CA Top Secret?	Valid in RACF?	Returns...
PRIVILEGED: STC with privileged flag	No	No	Yes	The string 1 if the current user has this attribute or 0 if the current user does not have this attribute.
DEFINED: User is defined to the security system flag	Yes	Yes	Yes	The string 1 if the current user has this attribute or 0 if the current user does not have this attribute.
ALTER: ALTER authority flag	No	No	Yes	The string 1 if the current user has this attribute or 0 if the current user does not have this attribute.
CONTROL: CONTROL authority flag	No	No	Yes	The string 1 if the current user has this attribute or 0 if the current user does not have this attribute.
UPDATE: UPDATE authority flag	No	No	Yes	The string 1 if the current user has this attribute or 0 if the current user does not have this attribute.
READ: READ authority flag	No	No	Yes	The string 1 if the current user has this attribute or 0 if the current user does not have this attribute.
NONE: NONE authority flag	No	No	Yes	The string 1 if the current user has this attribute or 0 if the current user does not have this attribute.
GROUPLISTCONTAINS: Group list contents	No	No	Yes	The string 1 if the current user has this attribute or 0 if the current user does not have this attribute.

Field Name/Description	Valid in CA ACF2?	Valid in CA Top Secret?	Valid in RACF?	Returns...
DATE: RACINIT date	Yes	Yes	Yes	The date that security was last initialized (RACINIT) for this user in the form <i>yyyy/mm/dd</i> (for example, 2003/11/06).
STCNAME: Started task name	Yes	Yes	Yes	The data as a character string (for example, OPSMAIN).
DEFINEUSERS: Authorized to define users	No	No	Yes	The string 1 if the current user has this attribute or 0 if the current user does not have this attribute.
PROTECTDASD: Authorized to protect DASD	No	No	Yes	The string 1 if the current user has this attribute or 0 if the current user does not have this attribute.
PROTECTTAPE: Authorized to protect tape	No	No	Yes	The string 1 if the current user has this attribute or 0 if the current user does not have this attribute.
PROTECTTERMINALS: Authorized to protect terminals	No	No	Yes	The string 1 if the current user has this attribute or 0 if the current user does not have this attribute.
APPLICATIONLEVEL: Application level	No	No	Yes	The string 1 if the current user has this attribute or 0 if the current user does not have this attribute.
PORTOFENTRYLEVEL: Port of entry level	No	No	Yes	An integer value.
PORTOFENTRYDATA: Port of entry data	No	No	Yes	The data as a character string that may contain unprintable characters.

Field Name/Description	Valid in CA ACF2?	Valid in CA Top Secret?	Valid in RACF?	Returns...
TERMINALID: Terminal ID	Yes	Yes	Yes	The data as a character string (for example, A55TU071).
CLASSAUTHORIZATIONS: Class authorizations	No	No	Yes	A four-byte hexadecimal string where the meaning of the bits in the string is installation dependent (based on the Class Descriptor entries). You can use the C2X function to convert this field to printable format.
APPLICATION: Application name	Yes	Yes	Yes	The data as a character string.
APPLICATIONDATA: Application data	No	No	Yes	The data as a character string that may contain unprintable characters.
USERNAME: User name field	Yes	Yes	Yes	The user name as a character string (for example, JOHNSMITH).
GROUPLIST: Group list	No	Yes	Yes	A blank, separated list of group names or, if there are no groups defined, a zero length string.
SURROGATEUSERID : Surrogate user ID (audit)	No	No	Yes	The data as a character string.

Verify Data Set Access

To verify that the current user has security authorization to access a data set, issue the following version of the OPSECURE function:

```
var = OPSECURE('D','dsname','accesstype','volser')
```

dsname

Specifies the data set to be checked.

accesstype

Use the *accesstype* argument to specify the type of data set access you want to check. If you omit the *accesstype* argument, OPS/REXX uses a default of R (Read access).

For CA ACF2 systems, the access type can be:

A

Verify Alloc access to a data set

E

Verify Execute access to a data set

R

Verify Read access to a data set

U

Verify Update access to a data set

W

Verify Write access to a data set

For RACF or CA Top Secret systems, the access type can be:

A

Verify Alter access to a data set

C

Verify Control access to a data set

R

Verify Read access to a data set

U

Verify Update access to a data set

volser

Supplies the volume serial number to be validated. If you do not specify a *volser*, the argument is blank by default. If you are running RACF or CA Top Secret without GENERIC resources and no *volser* is specified, you receive a message reporting that the resource has no security protection.

Results of Access Queries

CA ACF2, CA Top Secret, and RACF return the string ALLOW if the specified type of access to the data set is allowed. Otherwise, an error message is returned. The returned value for CA ACF2 is a CA ACF2 message; for RACF or CA Top Secret, the returned value is one of the following messages:

```
RESOURCE NOT PROTECTED  
RESOURCE ACCESS DENIED
```

Excessive Access Queries

For CA ACF2 and CA Top Secret systems, each call to OPSECURE returning a data set access violation message counts as a real access violation. CA ACF2 may cancel the current job or TSO ID for excessive data set access violations. For this reason, avoid calling OPSECURE with every possible data set name to try to find out which data sets you can access. For more information about data set access validation, see the appropriate CA ACF2, CA Top Secret, or RACF guides.

Retrieve a Logon ID Field

To fetch the value of a logon ID field, use this form of the OPSECURE function:

```
var = OPSECURE('F',fieldname)
```

This OPSECURE call is *not* valid for RACF or CA Top Secret systems.

See the appropriate CA ACF2 guide for valid field names, or display a list of valid field names by issuing the TSO command ACF and typing this text:

```
HELP FIELDS
```

Keep in mind that your data center may have defined additional fields not listed in either the HELP or CA ACF2 guides, which you can retrieve as well.

Conversion of Retrieved Fields

OPSECURE always returns fields as character strings without any leading or trailing blanks. OPS/REXX does the following conversions depending on field format:

- Binary fields are converted to signed decimal values without leading zeros or blanks. The number zero is returned as 0.
- Character fields are returned as is, possibly truncated to the OPS/REXX defined maximum valid string length.
- Date fields are converted to the format *yyyy/mm/dd* with leading zeros kept (so that the result is always exactly ten non-blank characters). A zero date field is returned as the string *****/**/***.
- Time fields are converted to the format *hh:mm:ss* with leading zeros kept (so that the result is always exactly eight non-blank characters).
- Bit fields are converted to a 0 (FALSE or off) or a 1 (TRUE or on).
- Clock fields are converted to the format *yy/mm/dd hh:mm:ss* with leading zeros kept (that is, exactly eight non-blank characters, a blank, and eight more non-blank characters). A zero date field is returned as the string ***/**/***.
- Hexadecimal fields are returned as is. You can use the C2X function of OPS/REXX to convert these fields to printable format.
- Encoded/encrypted fields are always returned as null strings. As a result, you cannot use OPSECURE to find out passwords.

Request Security Product Data

To retrieve information about the security product (if any) on your system, use this form of the OPSECURE function:

```
var = OPSECURE('I', 'name')
```

Note: When a name other than PRODUCT, MODE, or RELEASE is specified, the information returned is based on the security environment of the current task or address space. If the current address space does not have a valid security environment (for example, the CONSOLE address space), the result of the function is INVALID SECURITY ENVIRONMENT. For commands issued from real MCS consoles, always use the CMD.USERID environmental variable to obtain the user ID of the command issuer in a CMD rule.

Name Argument

This argument specifies the type of information to be returned. You can specify any of the following values for *name*:

PRODUCT

Returns the name of the security product (TSS, RACF, ACF2, or UNKNOWN SECURITY PRODUCT).

Valid in CA ACF2: Yes

Valid in CA Top Secret: Yes

Valid in RACF: Yes

MODE

Returns one of the following CA ACF2 operating modes:

- QUIET
- LOG
- WARN
- ABORT
- OFF

Valid in CA ACF2: Yes

Valid in CA Top Secret: No

Valid in RACF: No

RELEASE

Returns the release number of CA ACF2, CA Top Secret, or RACF.

- For RACF and CA Top Secret systems, this is a three- to four-character string in one of the following forms:

- *rr.i*
- *rr.rr*

where *rr* or *rr.rr* is the major release number.

- For CA ACF2 systems, OPS/REXX returns one of the following strings:

- *rr.rr*
- UNKNOWN SECURITY RELEASE.

Valid in CA ACF2: Yes

Valid in CA Top Secret: Yes

Valid in RACF: Yes

USERID

Returns the security product user ID of the current user.

Valid in CA ACF2: Yes

Valid in CA Top Secret: Yes

Valid in RACF: Yes

GROUP

Returns the ID of the security group to which the current security product user or job is assigned.

Valid in CA ACF2: No

Valid in CA Top Secret: Yes

Valid in RACF: Yes

UID

Returns the UID string for the current CA ACF2 user.

Valid in CA ACF2: Yes

Valid in CA Top Secret: No

Valid in RACF: No

SOURCE

Returns the submission source for the current job or TSO session. This is the VTAM LU name for TSO sessions or the name of the job, TSO user, or started task that submitted the current job (wrote it to the internal reader).

Valid in CA ACF2: Yes

Valid in CA Top Secret: Yes

Valid in RACF: Yes

SPECIAL

Returns TRUE if the current user ID has the Special attribute, or returns FALSE otherwise.

Valid in CA ACF2: No

Valid in CA Top Secret: Yes

Valid in RACF: Yes

AUDITOR

Returns TRUE if the current user ID has the Auditor attribute or returns FALSE otherwise.

Valid in CA ACF2: No

Valid in CA Top Secret: Yes

Valid in RACF: Yes

OPERATIONS

Returns TRUE if the current user ID has the Operations attribute or FALSE otherwise.

Valid in CA ACF2: No

Valid in CA Top Secret: Yes

Valid in RACF: Yes

USERDATA

Returns the installation data for the current user ID.

Valid in CA ACF2: No

Valid in CA Top Secret: Yes

Valid in RACF: Yes

Important! This value will be removed from the CA OPS/MVS product in a future release. You should modify your code to use the new `OPSECURE('A','INSTALLATIONDATA')` function instead.

Validate a Password

To validate a password, use this form of the OPSECURE function:

```
var = OPSECURE('P', 'userid', 'password', 'newpassword')
```

If you omit the *newpassword* argument, OPSECURE validates the user ID and password. If you specify *newpassword*, OPSECURE changes the password.

For the RACF security environment, you must specify the *newpassword* argument in uppercase characters.

OPSECURE is valid for RACF, CA Top Secret, and CA ACF2 security environments but only AOF rules (in supervisor state) can call this function. Typically, request rules issue OPSECURE function calls.

For all three security products, if the password was correct (and a new password was assigned if specified), the returned value is the string ALLOW. Otherwise, OPS/REXX returns a message (or, for RACF or CA Top Secret, one of the following messages):

```
INVALID SECURITY ENVIRONMENT
USER PROFILE NOT DEFINED
PASSWORD IS NOT AUTHORIZED
PASSWORD HAS EXPIRED
USER NOT DEFINED TO THE GROUP
REJECTED BY INSTALLATION EXIT
ACCESS HAS BEEN REVOKED
SECURITY PRODUCT IS NOT ACTIVE
GROUP ACCESS HAS BEEN REVOKED
NOT AUTHORIZED TO USE THIS TERMINAL
INVALID DAY OR TIME OF DAY
TERMINAL CANNOT BE USED
NOT AUTHORIZED TO USE APPLICATION
```

For CA ACF2, invalid password attempt calls increase the invalid password violation counter for the specified user ID.

Request General Resource Data

To request general resource information, use this form of OPSECURE:

```
var = OPSECURE('R',resourceclass,resource name,requestcode)
```

Requestcode Values

This function call verifies that the current user has access to a generalized resource. The *requestcode* argument specifies the type of access to be verified. If you omit this argument, OPS/REXX uses the default value R (Read access).

For CA ACF2 systems, the request code can be:

Code	Description
R	Verify Read access to resource
A	Verify Add access to resource
D	Verify Delete access to resource
U	Verify Update access to resource

For RACF systems or CA Top Secret, the request code can be:

Code	Description
R	Verify Read access to resource
A	Verify Alter access to resource
C	Verify Control access to resource
U	Verify Update access to resource

For any of the three security products, if the specified access to the resource is allowed, OPS/REXX returns the string ALLOW. Otherwise, OPS/REXX returns an error message:

```
RESOURCE NOT PROTECTED  
RESOURCE ACCESS DENIED
```

For more information on generalized resources, see the appropriate CA ACF2, RACF, or CA Top Secret guides.

OPSENQ Function

This function performs two related classes of operations:

- It interacts with the z/OS ENQ/DEQ services to serialize usage of resources. Any rule or program using the OPSENQ function should use a `SIGNAL ON SYNTAX` statement to dequeue resources if the rule or program fails to run properly. In this case, OPSENQ returns a numeric value that is the return code from the ENQ or DEQ service.

Note: Leaving the resources queued can leave your system inoperable.

- It uses the z/OS GQSCAN service to provide information about outstanding resource serialization in the current serialization complex. In this case, OPSENQ returns a positive numeric value that indicates the number of data lines added to the external data queue (EDQ).

Note: The OPSENQ function can be used in OPS/REXX or AOF rules.

The OPSENQ function has the following format:

```
var = OPSENQ(func [major][minor],type[,scope[,ref]])
```


func

Specify one of the following:

- E - To enqueue on a resource
- D - To dequeue from a resource
- C - To display resource conflicts
- Q - To display resources

Note: Functions E and D are related and have a similar set of rules. Functions C and Q are also related but have a different set of rules.

The following arguments apply only to the E and D functions:

major

(Optional) Identifies the major name of the resource, up to eight characters in length. By convention, the major and minor names together identify the resource. If you omit this argument, the default major name is the value of the ENQ parameter that, in turn, has a default value of OPS/MVS.

minor

(Optional) Identifies the minor name of the resource, up to 255 characters in length. If the OPSENQ function was called from an AOF rule, the default minor name is RULEOPxxssssssss.rrrrrrr, where Opxx is the subsystem name and sssssss.rrrrrrr is the name of the rule set and rule that issued the enqueue. In all other cases, the default name is RULEuuuuuuuppppppppp where uuuuuuu is the current TSO user ID or job name and ppppppppp is the name of the current REXX program that issued the enqueue.

type

(Optional) Specifies the type of enqueue request, either E for exclusive or S for shared. This argument is not used for dequeue function calls. The default is E.

scope

(Optional) Specifies the scope of the enqueue request. You can specify the following values:

- STEP for a job step-wide enqueue request.
- SYSTEM for a system-wide enqueue request. This is the default.
- SYSTEMS for a systems-wide enqueue request.

ret

(Optional) Specifies the type of return for the enqueue request. This value is one of the following:

- HAVE - Returns control when the enqueue has been obtained (it waits if another user holds the resource). This is the default.
- NONE - Returns control when the enqueue has been obtained.
- TEST - Tests whether the desired enqueue is available immediately but does not enqueue on the resource.
- USE - Enqueues the desired resource only if it is available immediately.

Default: HAVE

The following arguments apply only to the C and Q functions:

major

Identifies the major name of the resource for which you want to obtain information. It may be up to eight characters in length. By convention, the major and minor names together identify the resource. If you omit this argument, the default major name used for the scan is * (that is, all major names).

Note: The major names associated with the C function can be *. However, the Q function requires that either the major names or the minor names must be more specific than a wildcard.

minor

Identifies the minor name of the resource for which you want to obtain information. It may contain up to 255 characters. If you omit this argument, the default minor name used for the scan is * (that is, all minor names).

Note: The minor names associated with the C function can be *. However, the Q function requires that either the major names or the minor names must be more specific than a wildcard.

type

This argument must either be omitted or specified as a single wildcard character *. It is currently used only as a placeholder.

scope

Limits the scan to ENQs with the specified scope. You can specify the following values:

- STEP for job step-wide enqueue requests
- SYSTEM for system-wide enqueue requests
- SYSTEMS for systems-wide enqueue requests
- ALL to scan for all possible scopes

Note: There is no default scope value.

ret

This argument is not applicable to the C and Q functions and must be omitted.

OPSRC and OPSRS Variables

When the C or Q functions of OPSEQ are used with valid syntax, OPSRC and OPSRS variables are created to indicate the success or failure of the function. The OPSRC variable that is generated will contain one of the following values:

0

The function was called properly and at least one matching resource was found and returned in the EDQ. The value returned by the function is a positive integer and represents the number of resources about which information is returned on the EDQ. The value of OPSRS is zero.

4

The function was called properly. However, no resources matched the request. No lines were added to the EDQ and the value returned by the function is zero. The value of OPSRS is zero.

10 or 12

An error occurred while processing the request. OPSRS contains a non-zero value. For more information, see the IBM programming documentation of the GQSCAN service. OPSRC matches the GQSCAN return code and OPSRS matches the GQSCAN reason code.

99

The function was called properly; however, too much data was returned and the EDQ overflowed. The returned value is an integer that indicates the number of resources for which information is returned on the EDQ. Either use the information returned or retry the operation with a larger EDQ. The value of OPSRS is zero.

Note: A syntax error in the function call statement causes an OPS/REXX error and terminates the program. A REXX error 40, Incorrect Call to Routine, is generated.

ENQ Information Returned in the EDQ

The following is the format of the data returned in the EDQ for the C and Q function codes:

Word Number: 1

Contains the jobname of the ENQ requestor or the jobname of the ENQ provider and requestor delimited with a slash.

Length: 17

Word Number: 2

Contains the system name of the ENQ requestor.

Length: 8

Word Number: 3

Contains the ENQ major name (QNAME).

Length: 8

Word Number: 4

Contains the scope of the ENQ request:

- SYS-a scope of SYSTEM
- SYSS-a scope of SYSTEMS
- STEP-a scope of STEP
- UNKN-this should never occur

Length: 4

Word Number: 5

Contains the global (G) or local (L) request.

Length: 1

Word Number: 6

Contains the ASID of ENQ requester or the ASID of ENQ provider and requestor delimited with a slash in hexadecimal format.

Length: 9

Word Number: 7

Contains the TCB address of ENQ requestor in hexadecimal format.

Length: 8

Word Number: 8

Contains the status of O = Owns the resource or W = Waiting for the resource.

Length: 1

Word Number: 9

Contains the type of S = Shared or E = Exclusive.

Length: 1

Word Number: 10

Contains the device number (if the request was a RESERVE request) or N/A.

Length: 1

Word Number: 11

Contains the ENQ minor name (RNAME).

Note: The minor name (RNAME) may contain embedded blanks; therefore, everything after the tenth word should be treated as part of the RNAME. Each record contains at least eleven words, however, there may be more.

Length: 255

Examples: OPSENQ function in various scenarios

- This example illustrates serializing the use of a pseudo resource called RESOURCEA on the current system:

```
temp = OPSENQ("E", "RESOURCEA", "E", "SYSTEM", "HAVE")
```

```
.  
.
.
```

OPS/REXX code that requires serialization

```
.  
.
.
```

```
temp = OPSENQ("D", "RESOURCEA")
```

- This example illustrates using OPSENQ to query ENQ information, on the local system, about a CA OPS/MVS rule data set:

```
lines=OPSENQ('Q','SYSDSN','SYS1.OPS.SUPPRESS.RULES', 'SYSTEM')
```

- This example illustrates using OPSENQ to return ENQ information about all the ISPF Editor ENQs held for the high level qualifier of a TSO user ID throughout the global serialization complex:

```
lines=OPSENQ('Q','SPFEDIT','MYUID.*', 'ALL')
```

- This example illustrates using OPSENQ to return information about all ENQ conflicts in the global serialization complex:

```
lines=OPSENQ('C', '*', '*', 'ALL')
```

- This example illustrates the data format returned in the EDQ if one serialization job is executed on behalf of another. The provider and requestor jobnames are delimited with a slash. Their ASIDs are also concatenated with a slash.

```
CATALOG/T01403A CA11 SYSIGGV2 SYS L 001F01E8 006DCBC8 O S 2E6A 0FID1USERCAT2
```

OPSGETVL Function

The OPSGETVL function retrieves the names of global variables that match a variable name mask that you specify, and then returns the number of variables it found to *var* and *prefix.0*. This function also returns the actual variable names it found to an array of variables named *prefix.1*, *prefix.2*, and so on. The default prefix is GETVL.

Notes:

- The OPSGETVL function can be used in OPS/REXX, AOF rules, TSO/E REXX, or CLIST.
- This function is supplied for compatibility with Automate code. When writing new code, use OPSVALUE instead.

This function has the following format:

```
var = OPSGETVL(namemask [prefix][optional keywords])
```

Namemask Argument

Specify an exact global variable name for the *namemask* argument, or use the following wildcard characters to select a range of names:

Character	Description
+	Matches any single character in a variable name.
*	Matches any number of trailing characters in a variable name.

The + wildcard character can be specified anywhere after the first period in a variable name.

Example: Wildcard Characters

The following are valid examples of using the + and * wildcard characters.

- The + wildcard character can be specified anywhere after the first period in a variable name. For example:
 - This is a valid namemask using the + wildcard character:
`globak.+`
 - This is an invalid namemask using the + wildcard character:
`global+.x`

- The * wildcard character can be specified only at the end of a variable name. For example:
 - This is a valid namemask using the * wildcard character:
 globalx*
 - This is an invalid namemask using the * wildcard character:
 Glo+al*x

Keywords for OPSGETVL

For explanations of the keywords you can use with OPSGETVL, see OPSGETVL Command Processor in the chapter “POI Command Processors.”

Example: OPSGETVL Code

Consider the following code sample:

```
globalx.0=1
globalx.1='test1234567890abcdefghij'
globalx.2='abcdefghijklnopqrwxyz12345'
globalx.3='zyxwvrgponmlkjihgfedcba.12345'
a = OPSGETVL('globalx.*')
if opsrc ^= 0 & opsrc ^= 8 then do
  say queued()
  say 'opsrc='||psrc
  exit
end
if opsrc = 8 then
  say 'Some variable names not returned due to MAX limit'
  say 'a' = a
  do i = 1 to a
    say getvli
  end
```

When executed, the above code sample produces the following output:

```
a = 4
GLOBALX.0
GLOBALX.1
GLOBALX.2
GLOBALX.3
```

For additional examples that illustrate how you can use OPSGETVL, see OPSGETVL Command Processor in the chapter “POI Command Processors.”

Usage Notes for OPSGETVL

The MAX keyword is applied before the SORT keyword. For example, if you specify SORT(descend) and MAX(2), then the names of the first two variables are retrieved, not the last two. This behavior is counterintuitive.

OPSHFI Function

The OPSHFI function reads global variables from a VSAM data set, writes global variables to a VSAM data set, or deletes global variables from a VSAM data set. The VSAM data set may be shared among systems, and application code on one system can read and update shared variables written by any system. This function can operate under OPS/REXX or TSO/E REXX. The value returned by this function is the number of global variables read, written, or deleted from VSAM.

CA OPS/MVS global variables are stored in memory, and reading information from or storing information to memory satisfies all OPS/REXX references to the global variables.

The OPSHFI function can be used with only OPS/REXX global variables beginning with a valid global stem prefix (GLOBAL, GLOBALx, or GLVTEMPx). Global stem prefix GLVJOBID is not supported by the OPSHFI function. It is most useful for GLVTEMPx variables because they are not usually backed up to disk. For this function, Automate-format variables are also treated as global. Automate-format variables are defined in the ATMLOCALSCOPE nn and ATMSHAREDScope nn parameters.

When the OPSHFI function is invoked as a function in an AOF rule, for which waits are not allowed, the NOOUTPUT keyword is implied.

Return codes for this function are automatically stored in the OPSRC variable. For an explanation of the OPSHFI return codes, see OPSHFI Command Processor in the chapter “POI Command Processors.”

Note: The OPSHFI function can be used in OPS/REXX, AOF rules, TSO/E REXX, or CLIST.

This function has the following format:

```
var = OPSHF(READ|WRITE|DELETE namemask [optional keywords])
```


Keywords for OPShFI

For explanations of the keywords you can use with OPShFI, see OPShFI Command Processor in the chapter “POI Command Processors.”

Examples: OPShFI

- The following code reads all variables matching the namemask into storage, and replaces existing variables in memory or creates new ones as needed:

```
a = opshfi(read glvtemp1.pjc.* shared)
```
- The following code writes all variables matching the namemask glvtemp1.pjc. to the VSAM data set as local variables:

```
a = opshfi(write glvtemp1.pjc.* local)
```
- The following code deletes all local variables matching the namemask glvtemp1.pjc. from the VSAM data set. LOCAL is the default and can be omitted.

```
b = opshfi(delete glvtemp1.pjc.* local)
```

For additional examples that illustrate how you can use OPShFI, see OPShFI Command Processor in the chapter “POI Command Processors.”

OPSINFO Function

The OPSINFO function returns information about CA OPS/MVS, the environment where the issuing OPS/REXX program is running, or both.

Note: You can use the OPSINFO function in OPS/REXX or AOF rules.

This function has the following format:

```
var = OPSINFO(string)
```

Environmental OPSINFO Functions for JES2

Some of the variations of the OPSINFO function are special JES2 OPSINFO environmental variables. These variables enhance the rule environment by providing information that exists at the time when an event occurs.

Using the JES2 environmental OPSINFO functions requires access to a JES2 offsets table. Module OPJ2DF contains an already-assembled, already-linked default JES2 offsets table with support for the JES2 versions for z/OS 1.8 through z/OS 1.10. At startup, CA OPS/MVS checks to see which JES2 release you have, and then loads and searches the OPJ2DF module. If the module contains an offset table matching your JES2 release, all CA OPS/MVS components use the default offsets table in OPJ2DF.

Important! Even if your site can use the default offset tables in OPJ2DF, you still may need to reassemble and relink the JES2 offsets table that resides in SYS1.OPS.ASM(OPJ2CB). If your system support people perform aggressive maintenance on JES2 and z/OS, we recommend that you reassemble and relink OPJ2CB instead of relying on the OPJ2DF module to support the JES2 environment. You must reassemble and relink OPJ2CB each time someone applies maintenance to or modifies JES2.

The JCL needed to assemble and link the OPJ2CB module resides in SYS1.OPS.CNTL(JES2ASM). For detailed information on installing and starting CA OPS/MVS, see the *Installation Guide* and the *Administration Guide*.

When you start the CA OPS/MVS product on a JES2 system and CA OPS/MVS detects that the default (null) OPJ2CB module has been loaded, it issues error message OPS0160, indicating that the JES2-related OPSINFO functions will not work. In addition, anytime a rule or OPS/REXX program uses one of these functions, message number OPS4220 is issued.

When Work Load Manager (WLM) controls JES2 initiators, JES related information is not available at EOS (end of step) and EOJ (end of job). For example, OPSINFO('JOBID') in an EOJ rule would not be available when WLM controls the JES2 initiators.

Types of OPSINFO Functions

In the following list of OPSINFO functions, functions marked with one asterisk (*) are valid only for JES2 users, and functions marked with two asterisks (**) are valid only for JES3 users. All other functions operate in both JES environments.

- `var = OPSINFO('ACCOUNT')`

Returns ACCOUNT information for the current address space. Multiple accounting fields are returned separated by blanks.

- `var = OPSINFO('AOFTEST')`

Returns the value ACTIVE when the current REXX code is executing in the AOF Test environment, otherwise returns INACTIVE. Use this function to check whether a rule is executing in AOF Test.

- `var = OPSINFO('AMJOBID')`

Returns the job ID in one of the following formats:

- `yyyynnnnn` where `yyy` is STC, JOB, or TSU and `nnnnn` is the JES job ID number; or a zero length string if the job is running SUB=MSTR.
- `yynnnnnnn` where `y` is S (STC), J (JOB), or T (TSU) and `nnnnnnn` is the JES job ID number; or a zero length string if the job is running SUB=MSTR.

Note: OPSINFO('JOBID') returns the same information but in a slightly different format.

- `var = OPSINFO('ARCHLVL')`

Returns a value indicating the architecture level:

- 1 indicates ESA/390 architecture
- 2 indicates z/architecture

- `var = OPSINFO('ASID')`

Returns the z/OS Address Space Identifier (ASID) number of the address space in which the CA OPS/MVS program issuing this function call is running. The value returned is two characters whose binary value is the ASID.

- `var = OPSINFO('CLOCK')`

Returns the eight-byte binary value result of doing a STORE CLOCK on the CPU at the moment the function is called.

- `var = OPSINFO('CPCMODEL')`

Returns a string containing the model number of the central processing complex (CPC) that this program is executing on. This value is related to the value returned by OPSINFO ('CPUMODEL'), but is in more meaningful format. For example, it returns the value R75 on an IBM 9672-R75, whereas the CPUMODEL value is 77. In cases of older processors where this information is unavailable, the value returned is identical to that returned by OPSINFO ('CPUMODEL').

■ `var = OPSINFO('CPCMODEL')`

Returns a string containing the dynamic model number of the central processing complex (CPC) on which this program is executing. The value returned by this function is usually the same as that returned by `OPSINFO('CPCMODEL')`, but it will be different in the following cases:

- A dynamic processor upgrade has been performed and the CPC has not been IMLed.
- The processor does not support the STSI instruction. In this case, this function returns the string `NOSTSI`.
- The STSI instruction failed to retrieve the required information. In this case, this function returns the string `STSIFL`. This should not occur. If it does, then report it to CA OPS/MVS customer support and request assistance.
- The CA OPS/MVS subsystem is not active. In this case, this function returns the string `SS_N/A`.

Notes:

- In AOF Test, this function always returns the same result as `OPSINFO('CPCMODEL')`
- When not in AOF Test and the result is not one of the special strings `NOSTSI`, `STSIFL`, or `SS_N/A`, you can compare the result of this function with that of `OPSINFO('CPCMODEL')` to determine if a dynamic processor upgrade has been performed

■ `var = OPSINFO('CPSZAAPS')`

Returns the count of zAAPs (zSeries Application Assist Processors) currently online.

■ `var = OPSINFO('CPSZIIPS')`

Returns the count of zIIPs (zSeries Integrated Information Processors) currently online.

■ `var = OPSINFO('CPSREGULAR')`

Returns the count of regular central processors (CPs) currently online.

■ `var = OPSINFO('CPUID')`

Returns the six-character CPU identifier of the CPU where the OPS/REXX program or rule is running.

■ `var = OPSINFO('CPUMODEL')`

Returns the two-character CPU sub-model code of the CPU where the OPS/REXX program or rule is running. When running under VM this function returns the value `FF`.

- `var = OPSINFO('CPUSCONFIG')`

Returns the configured CPU count. This value represents the number of CPUs that are in the configured state. A CPU is in the configured state when it is in the configuration and available for executing programs.

- `var = OPSINFO('CPUSONLINE')`

Returns the count of regular CPs and zAAPs (zSeries Application Assist Processors) and zIIPs (zSeries Integrated Information Processors) currently online.

- `var = OPSINFO('CPUSRESERVED')`

Returns the reserved CPU count. This value represents the number of CPUs that are in the reserved state. A CPU is in the reserved state when it is in the configuration, is not available to be used to execute programs, and cannot be made available by issuing instructions to place it in the configured state.

Note: It may be possible to place a reserved CPU in the standby or configured state by means of manual actions.

- `var = OPSINFO('CPUSSTANDBY')`

Returns the standby CPU count. This value represents the number of CPUs that are in the standby state. A CPU is in the standby state when it is in the configuration, is not available to be used to execute programs, and can be made available by issuing instructions to place it in the configured state.

- `var = OPSINFO('CPUSTOTAL')`

Returns the total basic CPU count. This value represents the total number of CPUs in the configuration. This number includes all CPUs in the configured state, the standby state, or the reserved state.

Note: $CPUSTOTAL = CPUSCONFIG + CPUSSTANDBY + CPUSRESERVED$

- `var = OPSINFO('CPUTYPE')`

Returns the four-character CPU type (for instance, 9672) of the CPU where the OPS/REXX program or rule is running.

- `var = OPSINFO('DFPVERSION')`

Returns the level of DFP running on the system where the OPSINFO function call was executed; in the form *r.r*, where *r.r* is the release number (for example, 3.3).

- `var = OPSINFO('DFSMSVERSION')`

Returns the version of DFSMS in the form *r.r*, (for example, 1.5). If DFSMS is not installed, this OPSINFO function returns the string N/A. If you have installed DFSMS, the OPSINFO('DFPVERSION') function always returns the same value.

■ `var = OPSINFO('EVENTTYPE')`

Returns the type of event the rule is processing, such as MSG or DOM. This function is intended for use in external OPS/REXX functions that may be called by more than one kind of rule.

When you use this function outside the rule environment (for example, in an OPS/REXX program in a server), it always returns the string NONE.

This function returns one of these values:

- API
- ARM
- CMD
- DIS
- DOM
- ENA
- EOJ
- EOM
- EOS
- GLV
- MSG
- OMG
- REQ
- SCR
- SEC
- TLM
- TOD
- USS
- NONE

■ `var = OPSINFO('EXECNODEID')`

Returns the JES2 execution node ID. If the primary job entry subsystem is not JES2, or JES2 is not active when this function is called, the EXECNODEID function returns a 15-character string instead of data: *NOT-AVAILABLE*

■ `var = OPSINFO('EXECPGM')`

Returns the name of the program specified on the EXEC PGM=*progrname* JCL statement for the current step.

■ `var = OPSINFO('EXITTYPE')`

Returns the name of the exit in which the AOF captured the event that triggered this rule. This function returns one of these values:

- CA7
- CICS
- DSN
- ERR
- IMS
- JES3
- MVS
- NONE
- OMG
- TRAC
- NIP
- CNSV

A common use of OPSINFO('EXITTYPE') is to prevent processing a single message more than once in the IMS or JES3 environments.

■ `var = OPSINFO('GRSMODE')`

Returns a string indicating the mode in which Global Resource Serialization (GRS) is executing. This function returns one of these values:

- NONE-Global GRS is not active
- RING-Global GRS is in Ring mode
- STAR-Global GRS is in Star mode
- UNKN-Unable to determine the mode (should not occur)

■ `var = OPSINFO('HARDWARENAME')`

Returns the name of the processor configuration.

■ `var = OPSINFO('IMSID')`

Returns the IMS identifier (IMSID) associated with the address space where the OPS/REXX program is running. NONE is returned if the address space is not an IMS Control Region, a DLISAS Region, a DBRC address space, a Message Processing Region, or a Batch Message Processing Region.

IMSID is of greatest interest to AOF message rules. For example, canceling an IMS BMP is dangerous because doing so can bring down all of IMS. A rule can use the OPSINFO function to:

- Determine if the job issuing a z/OS message that usually calls for cancellation is an IMS-dependent region
- Take some other action for these regions

Note: This function is available only if the IOF is licensed, installed, and active at your site. If the IOF is not installed or is inactive, this function always returns the string NONE.

■ `var = OPSINFO('IMSTYPE')`

The type of IMS region for the address space in which the current REXX program or rule is running. The possibilities are:

- NON-The address space where the current REXX program or rule is running was not associated with any IMS Control Region
- CTL-The address space where the current REXX program or rule is running was an IMS Control Region
- DEP-The address space where the current REXX program or rule is running was an IMS MPP region
- BCH-The address space where the current REXX program or rule is running was an IMS BMP region

OPSINFO cannot determine if the address space is using batch DL/I facilities independent of any Control Region (even if using database- or block-level sharing through DBRC/IRLM).

Note: This function is available only if the IOF is licensed, installed, and active at your site. If the IOF is not installed or is inactive, this function always returns the three-character string NON.

■ `var = OPSINFO('INPUTDEV')`

Returns the name of the input device for a job. The input device can be an internal reader, a card reader, a remote workstation, or even a line connected to an NJE system. This variable enables you to specify installation policy based on where a job comes from: for example, to prevent a job that came in from LNE1 from exceeding the output limit. If the primary job entry subsystem is not JES2, or JES2 is not active when this function is called, the INPUTDEV function returns a 15-character string instead of data: *NOT-AVAILABLE*

- `var = OPSINFO('IPLDATE')`

Returns the date when z/OS was last IPLed (for example, 20030428 for April 28, 2003), in standard OPS/REXX date format S.
- `var = OPSINFO('IPLDEVICE')`

Returns the device number of the DASD volume from which the last IPL was performed, (for example, 9C01). This device number can have four hexadecimal digits.
- `var = OPSINFO('IPLTIME')`

Returns the time when z/OS was last IPLed (for example, 13:30:10 for 1:30 P.M. plus 10 seconds), in standard OPS/REXX time format N.
- `var = OPSINFO('IPLTYPE')`

Returns a value representing how the system was last IPLed. This value is one of the following:

 - CLPA-The system was IPLed using the CLPA option
 - CVIO-The system was IPLed using the CVIO option
 - WARM-The system was warm started (IPLed without a complete shutdown)
- `var = OPSINFO('IPLVOLSER')`

Returns the volume serial of the DASD volume from which the last IPL was performed, (for example, SYSRES).
- `var = OPSINFO('ISPF')`

Returns the value ACTIVE when ISPF is active in the current environment otherwise returns NOT ACTIVE. Use this function to check whether ISPF services can be used in the current OPS/REXX program.
- `var = OPSINFO('JCTUSER')`

Returns a 64-byte hexadecimal string containing data obtained from the consecutive four byte fields in the JES2 JCT (\$JCT), starting at fields JCTUSER0 through JCTUSERF. This information is available to the installation and can be accessed in a CA OPS/MVS rule. If the primary job entry subsystem is not JES2, or JES2 is not active when this function is called, the JCTUSER function returns a 15-character string instead of data: *NOT-AVAILABLE*
- `var = OPSINFO('JES')`

Returns the name of the Primary Job Entry Subsystem. Typically, this value is either JES2 or JES3, but it can be any subsystem name that you have assigned to the primary JES (JES5, JES9, and so on). Contrast with the description of OPSINFO('JESTYPE').

■ `var = OPSINFO('JES3GBLNAME')`

Returns one of these values:

- The JES3 global processor name if JES3 is the primary job entry subsystem
- NULL if JES3 is not the primary job entry subsystem or JES3 is not active in the system

■ `var = OPSINFO('JES3SPOOL')`

Available only in rules, the JES3SPOOL keyword obtains information about the prefix character JES3 is placing in front of each console message.

The OPSINFO('JES3SPOOL') function returns one of these values:

- N/A if you use the function from in the AOF test environment
- NONE if this is a JES3 global system and no spool space problems exist
- NOTJES3 if the host system is not running JES3 or if the ignore JES3 flag is set
- JES3DOWN if JES3 has failed or if JES3 is not started
- JES3LOCAL if the current system is a JES3 local
- MARGINALSPACE if JES3 spool space is marginal
- MINIMALBUFFERS if JES3 JSAM buffers are minimal
- MINIMALSPACE if JES3 spool space is minimal

The last three conditions listed above correspond to separate message prefix characters.

■ `var = OPSINFO('JESSYSNAME')`

Returns the ID of the JES on which CA OPS/MVS is active. For systems running JES2, this value is the SMF ID of the current z/OS system. For those running JES3, it is the MP name.

■ `var = OPSINFO('JESTYPE')`

Returns a value that indicates whether the primary JES subsystem is a JES2 or JES3 subsystem. The returned value is always either JES2 or JES3, regardless of the actual subsystem name. For example, if the primary JES is called JES9 but it is a JES2 subsystem, the following values are returned:

`OPSINFO('JES') = 'JES9'`

`OPSINFO('JESTYPE') = 'JES2'`

Note: The value returned by OPSINFO('JESTYPE') may not be valid when the primary JES is not active in the system (for example, immediately following an IPL).

■ `var = OPSINFO('JOBCLASS')`

Returns the current job class. If used in a JES3 environment, this function returns the eight-character JES3 job class.

- `var = OPSINFO('JOBID')`

Returns the JES job identifier of the current home address space. This variable has one of the following formats:

- `xnnnnn`, where *x* is J (JOB), T (TSU), or S (STC) and *nnnnn* is the job number.
- `xnnnnnnnn`, where *x* is J (JOB), T (TSU), or S (STC) and *nnnnnnnn* is the job number.

If you are using this function in a message rule, then for more information see the description of the MSG.JOBID variable in the chapter “Coding Each AOF Rule Type” in the *AOF Rules User Guide*.

- `var = OPSINFO('JOBNAME')`

Returns the JOBNAME associated with the address space where the OPS/REXX program is running. For a message rule being processed in the z/OS SS09 exit, OPS/REXX returns the job name of the issuer of the WTO that triggered the rule. This means that for a reissued message that originated on another system the job name is the one that actually issued the WTO on the originating system.

- `var = OPSINFO('LCPUSCONFIG')`

Returns the LPAR configured CPU count. This value represents the number of logical CPUs for this level-2 configuration that are in the configured state. A logical CPU is in the configured state when it is in the level-2 configuration and is available to be used to execute programs.

- `var = OPSINFO('LCPUSRESERVED')`

Returns the LPAR reserved CPU count. This value represents the number of CPUs for this level-2 configuration that are in the reserved state. A logical CPU is in the reserved state when it is in the level-2 configuration, is not available to be used to execute programs, and cannot be made available by issuing instructions to place it in the configured state. (It may be possible to place the reserved CPU in the standby or configured state through manual actions.)

- `var = OPSINFO('LCPUSSTANDBY')`

Returns the LPAR standby CPU count. This value represents the number of logical CPUs for this level-2 configuration that are in the standby state. A logical CPU is in the standby state when it is in the level-2 configuration, is not available to be used to execute programs, and can be made available by issuing instructions to place it in the configured state.

- `var = OPSINFO('LPUSTOTAL')`

Returns the total LPAR CPU count. This value represents the total number of logical CPUs that are provided for this level-2 configuration. This number includes all of the logical CPUs that are in the configured state, the standby state, or the reserved state.

Note: `LPUSTOTAL = LCPUSCONFIG + LCPUSSTANDBY + LCPUSRESERVED`

- `var = OPSINFO('LCSSID')`

Returns the logical channel subsystem ID as a hexadecimal number. There are four logical channel subsystems in one processor. Each processor can have up to 1024 CHPIDs

- `var = OPSINFO('LEAPSECONDS')`

Returns the leap seconds value in seconds. Use this value in conjunction with the OPSINFO('TIMEZONE') value to convert a GMT time to local time.

Note: Civil time is occasionally adjusted by one second increments to ensure that the difference between a uniform time scale defined by atomic clocks does not differ from the Earth's rotational time by more than 0.9 seconds. Coordinated Universal Time (UTC), an atomic time, is the basis for civil time.

- `var = OPSINFO('LOADPARAM')`

Returns the eight-character IPL Load Parameter string specified on the service console when the system was last IPLed. For older versions of z/OS, OPS/REXX returns a zero length string.

- `var = OPSINFO('LOCMSTCONSNM')`

Returns the first console name that has master authority on the local system. If there is no console that has master authority on the local system, the function returns the name of the sysplex master console. When a no consoles condition exists in the system, this function returns a null string.

- `var = OPSINFO('LPARNAME')`

Returns the LPAR name of the processor configuration. Returns a null string if not in LPAR mode.

- `var = OPSINFO('LPARNUM')`

Returns the LPAR number as a hexadecimal number. This number distinguishes the configuration from all other level-2 configurations provided by the same LPAR hypervisor.

- `var = OPSINFO('MAINPGM')`

Returns the REXX program name or the *ruleset.rulename* of the rule that invoked the currently executing REXX program. When the main program or rule is still executing, the returned value is the same as the value of OPSINFO('PROGRAM'). For example, suppose that program A calls program B, which, in turn, calls program C. The values that are returned to these programs would be as follows:

- If invoked from program A, both the OPSINFO('PROGRAM') and OPSINFO('MAINPGM') functions return the value A.
- If invoked from program B, the OPSINFO('PROGRAM') function returns the value B and OPSINFO('MAINPGM'), the value A.
- If invoked from program C, the OPSINFO('PROGRAM') function returns the value C and OPSINFO('MAINPGM'), the value A.

- `var = OPSINFO('MAINPRM')`

Returns the character string that was passed to CA OPS/MVS from the PARM field on its EXEC card when CA OPS/MVS was started. This string can contain up to 100 characters.

- `var = OPSINFO('MAXEDQ')`

Returns the maximum number of lines that can be placed in the REXX External Data Queue (EDQ) at any one time for the current OPS/REXX program or rule (for example, 3000).

- `var = OPSINFO('MODULE')`

Returns the name of the module that triggered the AOF event. For example, in a message rule it returns the name of the module (program) that issued the WTO. When used in an OPS/REXX program running in a TSO address space or in a server it returns the name of a CA OPS/MVS module.

To match products to modules use the following table:

Product	Module	Environment
SDSF	ISFMAIN	TSO or batch
TSOOPER	OPER	Command issued by TSO OPER
MCS	IEAVMQWR	Command issued from an MCS console
RCS	BDNRCSCH	Command or message issued by CA Remote Console
MIC	MIMCNSCH	Command or message from CA MIC
OPS/MVS	OPSAEX	Event created by a CA OPS/MVS command processor
NetView	DSIOST	NetView
NetView	DSIPPT	NetView

- `var = OPSINFO('MSFDEFAULTSYS')`

Returns the string that contains the default system setting for the current OPS/REXX program. The current default system is set through the ADDRESS OPSCTL MSF DEFAULT SYSTEM(xxxx) host command.

- `var = OPSINFO('MSFID')`

Returns the value of the SYSID parameter. For details, see the description of the SYSID parameter in the *Parameter Reference*.

- `var = OPSINFO('MSGCLASS')`

Returns the message class of a job. You can use it to determine output requirements for various printers. If the primary job entry subsystem is not JES2, or JES2 is not active when this function is called, the MSGCLASS function returns a 15-character string instead of data: *NOT-AVAILABLE*
- `var = OPSINFO('MSTCONSNM')`

Returns the name of the current MCS master console. When a no consoles condition exists in the system, this function returns a null string.
- `var = OPSINFO('MVSVERSION')`

Returns the level of z/OS on which the OPSINFO function call was executed in the form *version.release.modification level*.
- `var = OPSINFO('NOTIFYID')`

Returns the TSO user identifier who is notified when the job completes. This enables automation developers to send TSO user messages concerning their jobs. Under some conditions, the NOTIFYID function returns a 15-character string instead of data: *NOT-AVAILABLE*
- `var = OPSINFO('OMVS')`

Returns the current status of the OMVS kernel ('READY' or 'NOT READY').
- `var = OPSINFO('OPSLOGMSGNUM')`

Returns a string containing the OPSLOG message number associated with the current event. The result string always contains zeros when used in a non-AOF rule environment. This function is useful when used in conjunction with the OPSLOG() OPS/REXX function.

The result string contains zeros under the following conditions:

 - When called in the non-AOF rule environment
 - When the event type is not being recorded in OPSLOG, for example, a TOD rule and the BROWSETOD parameter is set to NO.
 - An MSG rule for which NOOPSLOG is specified.
- `var = OPSINFO('ORIGINNODE')`

Returns the originating node of a job. You can use it to implement a security policy for your system, or (in combination with the Multi-System Facility) to inform users about the status of their jobs. If the primary job entry subsystem is not JES2, or JES2 is not active when this function is called, the ORIGINNODE function returns a 15-character string instead of data: *NOT-AVAILABLE*
- `var = OPSINFO('PGN')`

Returns the Performance Group Number (PGN) of the currently executing address space. This value is always 0 when the system is running in GOAL mode.

- `var = OPSINFO('PRODUCT')`

Returns the value of the `PRODUCTNAME` parameter. For details, see the description of the `PRODUCTNAME` parameter in the *Parameter Reference*.

- `var = OPSINFO('PRODUCTSTARTS')`

Returns the number of times that the current CA OPS/MVS subsystem has been started since the last time an IPL of z/OS occurred, or 0 if the subsystem has not been started since the last IPL.

When used in an OPS/REXX program, this function returns a valid value even if the CA OPS/MVS subsystem being queried is inactive.

Note: The only time that you can absolutely infer that the CA OPS/MVS product is down from using this function is if it returns 0. If it returns any other value, CA OPS/MVS could be either up or down. For example, a value of 1 could mean that it is either up or down but that it has been started once since the last IPL. To determine if a particular product subsystem is active, check if the RC value set by this command is 0:

```
ADDRESS AOF "SUBSYS ssid"
```

- `var = OPSINFO('PRODUCTSTATUS')`

Returns the one of the following values:

- INIT-CA OPS/MVS is initializing.
- ACTIVE-CA OPS/MVS is active and processing system events.
- TERM-CA OPS/MVS is terminating.

Only rules can invoke this function. It returns a null string when a REXX program tries to invoke it. Typically, you might use this function:

- In the initialization section of a rule, to determine if this is the CA OPS/MVS initial startup.
- In the termination section of a rule, to determine if CA OPS/MVS is terminating.

- `var = OPSINFO('PRODUCTSTCNAME')`

Returns the name of the CA OPS/MVS started task or job, or INACTIVE if the product is not active.

- `var = OPSINFO('PROGRAM')`

In any environment other than a rule running under the CA OPS/MVS Automated Operations Facility (AOF), `OPSINFO('PROGRAM')` returns the name of the member from which OPS/REXX read the REXX program. If the OPS/REXX program was run from a sequential data set, which is rare, then `OPSINFO('PROGRAM')` returns a null string. In the AOF environment, `OPSINFO('PROGRAM')` returns *ruleset.rule*.

- `var = OPSINFO('PROGRAMMER')`

Returns the programmer information associated with the current address space. In the AOF environment, except when running in a server, the information returned is for the address space that triggered the AOF event.

- `var = OPSINFO('REALSTORAGE')`

Returns the size, in kilobytes, of the actual real storage online.

- `var = OPSINFO('RELEASE')`

(Synonym for `OPSINFO('VERSION')`). Returns an eight-byte string that provides the release and service pack number of the CA OPS/MVS product. The format of this string is *rr.rr* where *rr.rr* is the release number.

Thus, when using the `OPSINFO('RELEASE')` function to differentiate between code that may run on multiple versions of the CA OPS/MVS product, take into account that the modification level changes with each maintenance tape.

For example, to check for code that can run *only* with r11.5, use this function:

```
if SUBSTR(OPSINFO("RELEASE"),1,6)=="11.06." then ...
```

To check for code that can run with r11.6 or higher levels, use this function:

```
if SUBSTR(OPSINFO("RELEASE"),1,6)>="11.06." then ...
```

- `var = OPSINFO('SCHENV')`

Returns the room number that is associated with the output of the current address space. If the primary job entry subsystem is not JES2, or JES2 is not active when this function is called, the `SCHENV` function returns a 15-character string instead of data: **NOT-AVAILABLE**

- `var = OPSINFO('SERVICECLASS')`

Returns the WLM service class name, up to a maximum of 8 characters. If the name does not exist, then N/A is returned.

- `var = OPSINFO('SMFID')`

Returns the SMF identifier that your site has assigned to the z/OS image on which CA OPS/MVS is running. z/OS gets this value at IPL time from the appropriate `SMFPRMnn` member of the Logical Parmlib Concatenation.

■ `var = OPSINFO('SSMSUBSTOP')`

Returns a value indicating whether the SSM subtask has been requested to terminate. This value should be checked periodically by the AOF request rule called by the SSM subtask to determine if immediate exit from the rule is required.

The function returns one of these values:

- YES - The SSM subtask must terminate.
- NO - The SSM subtask has no pending request to terminate.
- N/A - This function is not applicable when executed outside of the CA OPS/MVS address space.

■ `var = OPSINFO('STATEMANSTATUS')`

Returns the current status of the System State Manager task. Possible values are:

- ACTIVE-System State Manager is active.
- INACTIVE-System State Manager is not active.
- NONE-System State Manager is not being used.
- NOPREREQ-Same as ACTIVE, but will not do any prerequisite checking.
- PASSIVE-System State Manager is active but is not monitoring resources.
- *NOT-AVAILABLE*-The CA OPS/MVS product is not active.

■ `var = OPSINFO('STATEMANTABLE')`

Returns the name of the relational table that System State Manager is currently using or would use if it were active. Possible values are:

- NOPREREQ-System State Manager is active, but not checking the status of prerequisites for system resources.
- *NOT-AVAILABLE*-The CA OPS/MVS product is not active.

■ `var = OPSINFO('STEPNAME')`

Returns the stepname or alternate task ID of the current address space. For example, if you issue the command `S CICS.CICSA`, the stepname for that address space is `CICSA`.

■ `var = OPSINFO('SUBMITTER')`

Returns the user ID that submitted the current job. If the primary job entry subsystem is not JES2, or JES2 is not active when this function is called, the SUBMITTER function returns a 15-character data string instead of data:
NOT-AVAILABLE

■ `var = OPSINFO('SUBSYS')`

Returns the four-character z/OS subsystem name that CA OPS/MVS is using. The CA OPS/MVS product obtains this value from the first four characters of the PARM field on the EXEC statement in the JCL procedure used for CA OPS/MVS. The default subsystem name is `OPSS`.

- `var = OPSINFO('SYSCClone')`

Returns the one- or two-byte shorthand notation for the system name from the IEASYMxx member in the Logical Parmlib Concatenation.
- `var = OPSINFO('SYSNAME')`

Returns the system name. Global Resource Serialization (GRS) uses this name to identify different systems in a GRS complex/sysplex. z/OS gets this value at IPL time from the SYSNAME parameter in the IEASYSxx member in the Logical Parmlib Concatenation.
- `var = OPSINFO('SYSOPSYS')`

Returns a string containing the z/OS name, version, release, modification level, and FMID. For example, on a z/OS V1R5 system, the following string is returned:

z/OS 01.05.00 HBB7708
- `var = OPSINFO('SYSplex')`

Returns the z/OS sysplex name from the COUPLExx or LOADxx members in the Logical Parmlib Concatenation.
- `var = OPSINFO('TMP')`

Returns the value ACTIVE when TMP is active in the current environment, otherwise returns NOT ACTIVE. Use this function to check whether TSO services can be used in the current OPS/REXX program.
- `var = OPSINFO('TIMEZONE')`

Returns the time zone offset from GMT in seconds. Use this value in conjunction with the OPSINFO('LEAPSECONDS') value to convert a GMT time to local time.

Note: This value may be a negative number.
- `var = OPSINFO('TSOEVERSION')`

Returns the TSO/E release level in the form *v.rr.m* (for example, 2.06.0). If TSO/E is not installed, this function returns the value N/A.
- `var = OPSINFO('USSEGID')`

Returns the effective numeric USS group ID. An effective group ID may be different from the real group ID of the user that created the process. Effective group ID is the group ID used for security checks. Authorized programs are allowed to change their group ID when required. If this ID is unavailable or undefined, this function returns the value N/A.
- `var = OPSINFO('USSEUID')`

Returns the effective numeric USS user ID. An effective user ID may be different from the real user ID of the user that created the process. Effective user ID is the group ID used for security checks. Authorized programs are allowed to change their user ID when required. If this ID is unavailable or undefined, this function returns the value N/A.

- `var = OPSINFO('USSGID')`

Returns the real numeric USS group ID. The real numeric USS group ID is the original group ID of the user that created the process. If this ID is unavailable or undefined, this function returns the value N/A.
- `var = OPSINFO('USSPGID')`

Returns the numeric process USS group ID. The process group ID is the group ID of the main process that created the current process. If this ID is unavailable or undefined, this function returns the value N/A.
- `var = OPSINFO('USSPID')`

Returns the UNIX System Services process ID number (PID), in decimal format, for the current z/OS task. The primary use of this function is to capture the PID from long-running USS processes that issue messages resulting in AOF message events. Possible values are:

 - `nnnnnnnnnn`-The process ID number of the current z/OS task.

Important! This number is *not* the same as the PID of the parent process when UNIX services fork or spawn are used to issue a message. These UNIX services may generate a short-term process resulting in a message that triggers an AOF message event.
 - N/A-No process ID exists for the current task.
- `var = OPSINFO('USSPPID')`

Returns the parent process USS ID for the current z/OS task. The parent process ID is the process ID of the process that created the current process. If this ID is unavailable or undefined, this function returns the value N/A.
- `var = OPSINFO('USSUID')`

Returns the USS real user ID for the current z/OS task. The real USS user ID is the original user ID of the user that created the current process. If this ID is unavailable or undefined, this function returns the value N/A.
- `var = OPSINFO('UTC')`

Returns the coordinated universal time (UTC) in yyyy/mm/dd hh:mm:ss format.

- `var = OPSINFO('VERSION')`

(Synonym for OPSINFO('RELEASE'). Returns an eight-byte string that provides the release number of the product. The format of this string is *rr.rr*, where *rr.rr* is the release number.

Thus, when using the OPSINFO('VERSION') function to differentiate between code that may run on multiple versions, take into account that the modification level changes with each maintenance tape.

For example, to check for code that can run *only* with r11.5, use this function:

```
if SUBSTR(OPSINFO("VERSION"),1,6)=="11.05." then ...
```

To check for code that can run with r11.5 or higher levels, use this function:

```
if SUBSTR(OPSINFO("VERSION"),1,6)>="11.05." then ...
```

- `var = OPSINFO('VMUSERID')`

Returns the VM user ID of the virtual machine on which this z/OS image is a guest. Returns a null string if z/OS is not running under VM.

- `var = OPSINFO(WSSIZE')`

Returns the size in bytes of the REXX work space for the current OPS/REXX program or rule. The REXX work space is used to contain REXX variables (for example, 1572864).

OPSIPL Function

The OPSIPL function obtains the parameter library member suffix and value of the requested IPL parameter. This information is obtained from the IPA control block of z/OS.

Note: The OPSIPL function can be used in OPS/REXX, AOF rules, TSO/E REXX, or CLIST.

This function has the following format:

```
var= OPSIPL(member prefix,parameter)
```

member prefix

The *member prefix* argument specifies the fixed portion of the Logical Parmlib Concatenation member name, which is associated with the subsequent parameter specification. The first data item returned is always the two-character suffix of the member name from which the associated parameter was derived by the system at IPL.

Possible values for the member `prefix` are:

LOAD

The subsequent parameter keyword is associated with a `LOADxx` member of the Logical Parmlib Concatenation, or the IPL parameters specified by the operator from the service console.

IEASYS

The subsequent parameter keyword is associated with an `IEASYSxx` member of the Logical Parmlib Concatenation. Since multiple `IEASYSxx` members may be used for the IPL, the suffix value returned is important in determining the actual `IEASYSxx` member that is the source of the parameter value.

parameter

The parameter argument identifies the keyword data item of the member `prefix` for which the value is desired.

The following are the `LOADxx`-related parameters:

- `HWNAME`-The HCD name of the central processor complex
- `IEASYM`-The suffix string of the `IEASYMxx` members
- `IODF`-The I/O definition file statement image
- `IODFUNIT`-The IODF unit address
- `IPLDEVICE`-The IPL load parameter data set device number
- `IPLDSN`-The IPL load parameter data set name
- `IPLFLAGS`-The `LOADxx` usage flags (1 byte binary).

Possible values are:

- `X'80'`-If this bit is on, the system used the Master JCL from `PARMLIB`. If this bit is off, the system used the Master JCL from `LINKLIB`.
- `X'40'`-If this bit is on, the system used the Master JCL `IEFPARMS DD` parameter libraries. If this bit is off, the system used the `LOADxx` parameter libraries.

The `LOADxx` usage flags (2 bytes binary). Possible value is:

- `X'00'`-The Master JCL came from `LINKLIB`
- `IPLIMSI`-The initial message suppression character from the IPL
- `IPLTOD`-The load parameter suffix followed by the date and time of completion of system initialization.

(ssyy/mm/dd H:MMSS)

- LPARNAME-The partition name, in HCD or IOCP
- MTLSHARE-The MTLSHARE value(Y or N), enables a full-support MTL system to treat manual tape library defined devices as stand-alone devices.
- NUCLEUS-The one-digit suffix of the selected nucleus IEANUC0n
- NUCLST-The suffix of the NUCLSTxx parameter library member and wait flag
- PARMLIB0-The number of parmlib data sets defined
- PARMLIB1-The first parameter library DSN, volume serial number, and usage flag (1 byte binary)
- PARMLIB2-The second parameter library DSN, volume serial number, and usage flag
- PARMLIB3-The third parameter library DSN, volume serial number, and usage flag
- PARMLIB4-The fourth parameter library DSN, volume serial number, and usage flag
- PARMLIB5-The fifth parameter library DSN, volume serial number, and usage flag
- PARMLIB6-The sixth parameter library DSN, volume serial number, and usage flag
- PARMLIB7-The seventh parameter library DSN, volume serial number, and usage flag
- PARMLIB8-The eighth parameter library DSN, volume serial number, and usage flag

- PARMLIB9-The ninth parameter library DSN, volume serial number, and usage flag
- PARMLIB10-The tenth parameter library DSN, volume serial number, and usage flag
- PARMLIB11-The eleventh parameter library DSN, volume serial number, and usage flag.
- PARMLIB12-The twelfth parameter library DSN, volume serial number, and usage flag.
- PARMLIB13-The thirteenth parameter library DSN, volume serial number, and usage flag.
- PARMLIB14-The fourteenth parameter library DSN, volume serial number, and usage flag.
- PARMLIB15-The fifteenth parameter library DSN, volume serial number, and usage flag.
- PARMLIB16-The sixteenth parameter library DSN, volume serial number, and usage flag.
- PARMLIB17-The seventeenth parameter library DSN, volume serial number, and usage flag.
- SYSCAT-The Master catalog definition statement image
- SYSPARM-The suffix string for IEASYSxx members
- SYSPLEX-The name of the sysplex that the system participates in
- VMUSERID-The z/VM userid under which the guest system is running

Following are the IEASYSxx-related parameters:

- ALLOC-The suffix string for ALLOCxx members
- APF-The suffix for IEAAPFxx member
- APG-The automatic priority group dispatching priority
- BLDL-The obsolete suffix for the IEABLDxx member
- BLDLF-The obsolete suffix for the IEABLDxx member
- CEE-The suffix string for the CEEPRMxx members
- CLOCK-The suffix string for the CLOCKxx members
- CLPA-The create link pack area option
- CMB-The additional device classes for measurement data
- CMD-The suffix string for the COMMNDxx members
- CON-The suffix string for the CONSOLxx member and the JES3 option
- CONT-The operator used CONT on console override reply

- COUPLE-The suffix string for COUPLExx member
 - CPQE-Unknown parameter
 - CSA-The size of CSA and ECSA
 - CSCBLOC-The location of CSCB control blocks ABOVE/BELOW
 - CVIO-The clear VIO data set option
 - DEVSUP-The suffix string for DEVSUPxx members
 - DIAG-The suffix string for DIAGxx members
 - DRMODE-Specifies that an IPL of a recovery system occurs as part of a disaster recovery scenario that requires special handling of certain resources
 - DUMP-The SYS1.DUMPxx data sets on the DASD option and the suffix range
 - DUPLEX-The data set name of the duplex page data set
 - EXIT-The suffix string for the EXITxx member
 - FIX-The suffix string for the IEAFIXxx members and the protect option
 - GRS-The GRS participation initialization parameter
 - GRSCNF-The suffix string for the GRSCNFxx member for GRS
 - GRSRNL-The suffix string for the GRSRNLxx members or EXCLUDE
 - ICS-The suffix string for the IEAICSxx member for SRM
 - IKJTSO-The suffix string for the IKJTSOxx member
 - IOS-The suffix string for the IECIOSxx member for MIH and I/O
 - IPS-The suffix string for the IEAIPSxx member for SRM
 - LFAREA-Specifies the amount of central storage to be made available to back large pages.
- Note:** This value is only available at z/OS 1.9 and higher
- LICENSE-Specifies whether this is a z/OS or a z/OS.e system.
 - LNK-The suffix string for LNKLSTxx
 - LNKAUTH-The linklist authorization option LNKLST/APFTAB
 - LOGCLS-The JES output class for SYSLOG data sets
 - LOGLMT-The number of WTLs per SYSLOG file
 - LOGREC-The LOGREC data set name or recording medium
 - LPA-The suffix string for LPALSTxx members
 - MAXCAD-The number of SCOPE=COMMON data spaces available to all users
 - MAXUSER-The maximum number of concurrent address spaces
 - MLPA-The suffix string for the IEALPAXx members and the protect option

- MSTRJCL-The suffix string of the MSTRJCLxx member for the Master JCL
- NONVIO-The list of NON-VIO eligible page data set names
- NSYSLX-The number of additional system linkage indexes
- NUCMAP-Obsolete parameter
- OMVS-The suffix string for the BPXPRMxx members for OpenEdition
- OPI-Allow operator overrides of IEASYSxx members YES/NO
- OPT-The suffix string of the IEAOPTxx member for SRM
- PAGE-The list of page data set names
- PAGEO-The list of page data set names entered by the operator
- PAGNUM-The obsolete parameter replaced by PAGTOTL
- PAGTOTL-The maximum number of PAGE and SWAP data sets
- PAK-The suffix string for IEAPAKxx members
- PLEXCFG-The type of system configuration into which system can start
- PRESCPU-Causes system initialization processing to bring logically online the CPUs that appear to MVS to be physically online, without regard to the number of CPUs defined to be initially online in the logical partition profile
- PROD-The suffix string for the IFAPRDxx members
- PROG-The suffix string for the PROGxx members
- PURGE-The obsolete option to dismount MSS volumes
- RDE-Include reliability data extractor YES/NO
- REAL-The maximum storage for V=R jobs
- RER-The reduced error recovery for magnetic tapes YES/NO
- RSU-The central storage units available for reconfiguration
- RSVNONR-The number of reserved entries in ASVT for non-reusable entries
- RSVSTRT-The number of reserved entries in ASVT for START
- RTLS-The suffix string for the CSVRTLxx member
- SCH-The suffix string for the SCHEDxx members
- SMF-The suffix string for the SMFPRMxx member for SMF
- SMS-The suffix string for IGDSMSxx member for SMS
- SQA-SQA and ESQA sizes
- SSN-The suffix string for the IEFSSNxx members
- SVC-The suffix string for the IEASVCxx members

- SWAP-The SWAP data set name list
- SYSNAME-The system name
- SYSP-The suffix string for the IEASYSxx members used for IPL
- UNI-The suffix string for the CUNUNIXx members
- VAL-Suffix string for VATLSTxx members
- VIODSN-The name of the VIO journal data set
- VRREGN-V=R region size

Documentation on the meaning and format of all parameters can be found in the *z/OS Initialization and Tuning Reference* and in the z/OS macro IHAIPA.

Examples: OPSIPL

■ Example 1

Display the IODF statement of the LOADxx member and parse the result:

```
iodf = OPSIPL('LOAD' , 'IODF')
>>> iodf = '00 00 IODF      CONFIGPR'
Parse Var iodf load_suffix iodf_stmt
Parse Var iodf_stmt iodf_suffix 3 . 4 iodf_hlq 12 . 13,
        iodf_config 21 . 22 iodf_edt 24 . 25 iodf_devsup 26
```

■ Example 2

1. Display the value of the CMD parameter of the IEFSYSxx member.
2. Display the result.
3. Read each member into the external data queue:

```
cmdx = OPSIPL('IEASYS','CMD')
>>> cmdx = '01 (00,44,A2)'
Parse Var cmdx ieasys_suffix cmdval
cmdval = Strip(Translate(cmdval,','))
Do while cmdval <> ''
    Parse Var cmdval suffix cmdval
    pcnt = OPSRMLB('LISTMEM','COMMND'|suffix)
End
```

■ Example 3

- Display the value of the CSCBLOC parameter of IEASYSxx.
- Assume that it was not specified, and that the system used the default value:

```
Say OPSIPL('IEASYS','CSCBLOC')
>>> D ABOVE
```

- Example 4

Some parameters yield null values when they are not specified:

```
Say OPSIPL('IEASYS','DEV/SUP')
>>> '
```

OPSIPL Returned Values

If the parameter has no value, a null string is returned. Otherwise, word one of the returned string is one of the following:

- The suffix of the LOADxx or IEASYSxx member from which the parameter was derived
- D - A system default value
- An operator specified value

The remainder of the string is the value of the parameter. Suffix strings and multi-valued parameters for IEASYSxx are generally enclosed in parentheses and separated by commas. LOADxx keyword statement images generally follow the operand layouts in the parameter library member description. The exact layout is mapped in macro IHAIPA. Usability flag meanings are also documented in macro IHAIPA. To determine the documented meaning, you can use the C2X() REXX function on the OPSIPL return value.

OPSJES2 Function

The OPSJES2 function returns the number of lines of JES2-related resource data from the primary JES2 that have been placed on the OPS/REXX external data queue. Using this function, you can inquire about the status of initiators, the input queue, both local and remote printers, spool function, NJE/RJE lines, and multi-access spool (MAS) information.

Note: You can use the OPSJES2 function in OPS/REXX or AOF rules.

Platform: The OPSJES2 call operates only if you have JES2 on your system.

This function has the following format:

```
var = OPSJES2('func','resource',['class'],['status'],['order'])
```

func

Specifies the type of call you want the OPSJES2 function to make. Currently, the only valid function call is I, for inquiry on a JES2 resource. You *must* always specify the *func* argument.

resource

Identifies the resource for which you want data.

Values are:

This resource value...	Indicates you want to inquire on...
INIT	Initiators
INPQ	Input queue
LINE	NJE/RJE lines
MASI	Multi-access spool definition
PRTS	Printers
SPOL	Spool

CA OPS/MVS treats certain OPSJES2 arguments differently, depending upon the value you specify for *resource*. The following chart indicates how the other arguments are treated:

If this is the value of <i>resource</i> ...	The <i>class</i> argument is...	The <i>status</i> argument is...	The <i>order</i> argument is...
INIT	optional	optional	optional
INPQ	optional	ignored	ignored
LINE	ignored	ignored	ignored
MASI	ignored	optional	ignored
PRTS	optional	optional	optional
SPOL	ignored	ignored	ignored

class

Specifies the search criteria for your inquiry by class definition. The *class* argument is valid only for the following *resource* values: INIT, INPQ, and PRTS:

- When the value of *resource* is INIT, the *class* argument can specify up to 36 classes. Each one-byte value that you specify is treated as a class.
- When the value of *resource* is INPQ, the *class* argument can specify only 1 one-byte *class* value, since jobs run in a certain class.
- When the value of *resource* is PRTS, the *class* argument can specify up to 16 classes. Each one-byte value that you specify is treated as a class.

The *class* argument is not required; if omitted, it defaults to *. If you allow *class* to default, the OPSJES2 function returns all data in all classes for which the *class* argument is applicable.

The following summarizes the usage information of the *class* argument:

INIT

The number of classes to specify is a minimum of 1; maximum of 36.

Default: *

INPQ

The number of classes to specify is 1.

Default: *

LINES

The number of classes to specify is 0; *class* is ignored.

Default: *

MASI

The number of classes to specify is 0; *class* is ignored.

Default: *

PRTS

The number of classes to specify is Minimum of 1; maximum of 16.

Default: *

SPOL

The number of classes to specify is 0; *class* is ignored.

Default: *

status

Specifies the status of the resource for which you want data. Valid values for the *status* argument are all one-byte values. You may specify only one *status* per OPSJES2 function call. The *status* argument is valid only for the following *resource* values: INIT and PRTS. Valid *status* values are:

A

Active

B

Busy (printers only)

I

Inactive

D

Drained

P

Draining

H

Halted

Z

Halting

Wildcard

The *status* argument is not required; if omitted, it defaults to *. If you allow *status* to default, the OPSJES2 function returns all data with any status for which the *status* argument is applicable.

The following summarizes the usage information of the *status* argument:

INIT

The value of the *status* argument is one of these values: A, I, D, P, H, Z.

Default: *

INPQ

The value of the *status* argument is ignored.

Default: *

LINE

The value of the *status* argument is ignored.

Default: *

MASI

The value of the *status* argument is A.

Default: *

PRTS

The value of the *status* argument is one of these values: A, B, I, D, P, H, Z.

Default: *

SPOL

The value of the *status* argument is ignored.

Default: *

order

Specifies the search order for your inquiry. The *order* argument is valid only for the following *resource* values: INIT and PRTS.

Valid *order* values are:

A

Any match

E

Exact match

S

Position significant

Wildcard

The *order* argument is not required; if omitted, it defaults to *. If you allow *order* to default, the OPSJES2 function call returns all data in the order in which it appears to JES2.

The following summarizes the usage information of the *order* argument:

INIT

The value of the *order* argument is one of these values: A, E, S.

Default: *

INPQ

The value of the *order* argument is ignored.

Default: *

LINE

The value of the *order* argument is ignored.

Default: *

MASI

The value of the *order* argument is ignored.

Default: *

PRTS

The value of the *order* argument is one of these values: E, S.

Default: *

SPOL

The value of the *order* argument is ignored.

Default: *

Record Fields for Inquiries

The following sections map the record fields for resource inquiries that are returned to the external data queue.

Record Fields for Initiator (INIT) Inquiries

Record fields for inquiries about initiators (defined as such with the INIT value) contain the following:

Word Number: 1

Length: 4

Contains the initiator ID or name.

Word Number: 2

Length: 8

Contains the initiator status. Possible values are:

DRAINED

DRAINING

HALTED

HALTING

ACTIVE

INACTIVE

STARTING

Word Number: 3

Length: 8

Contains the job name.

Word Number: 4

Length: 8

Contains the job number.

Word Number: 5

Length: 1

Contains the job class.

Word Number: 6

Length: 2

Contains the execution priority (1 through 15).

Word Number: 7

Length: 36

Contains the classes for initiators.

Word Number: 8

Length: 4

Contains the initiator number.

Word Number: 9

Contains the ASID of the initiator.

Length: 4

Word Number: 10

Length: 4

Specifies the service of either JES2 or WLM.

Word Number: 11

Length: 16

Contains the WLM scheduling environment name

Note: Word 10 (Service) indicates the class of initiator. If the initiator is a WLM managed initiator, this word contains the value 'WLM'. If the initiator is a JES2 managed initiator, this word contains the value 'JES2'. When the initiator status is INACTIVE, DRAINED, or HALTED this word contains a value of four asterisks.

Record Fields for Input Queue (INPQ) Inquiries

Record fields for inquiries about input queues (defined as such with the INPQ value) contain the following:

Word Number: 1

Length: 8

Contains the job name

Word Number: 2

Length: 8

Contains the job number

Word Number: 3

Length: 1

Contains the job class

Word Number: 4

Length: 2

Contains the execution priority

Word Number: 5

Length: 18

Contains the status information. This field may contain either of these values: Awaiting_Execution or Executing.

Word Number: 6

Length: 28

Contains the hold data. This field may contain one of the following values: Duplicate_Jobname, Purge, Cancelled, or Held.

Word Number: 7

Length: 4

Contains the system affinity of batch jobs. Each system affinity is up to four bytes long, separated by a comma.

Note: There are no blanks in this string of names, and REXX considers this string to be a single word. Using this information, users with shared spool environments can inquire about any batch job in their data center.

You can define up to 32 system affinities for each batch job using the job parameter `S=(sys1,sys2...)`. If a job has no system affinity, this field contains the following literal string:

ANY,****,****,....

Record Fields for RJE/NJE Line (LINE) Inquiries

Record fields for inquiries about SNA RJE/NJE lines (defined as such with the LINE value) contain the following:

Word Number: 1

Length: 8

Contains the line name as defined in JES2PARM.

Word Number: 2

Length: 4

Contains the unit name (UCB device number or the character string 'SNA').

Word Number: 3

Length: 8

Contains the node name.

Word Number: 4

Length: 8

Contains the VTAM application ID.

Word Number: 5

Length: 8

Contains the status. This field may contain one of the following values: ACTIVE, INACTIVE, DRAINED, or DRAINING.

Word Number: 6

Length: 10

Contains the last transmission date (in the form `yyyy/mm/dd`).

Word Number: 7

Length: 8

Contains the last transmission time (in the form *hh:mm:ss*).

Word Number: 8

Length: 8

Contains the total EXCPs done on this line (BSC lines) or the total count of VTAM requests processed (SNA lines)

Record Fields for Printer (PRTS) Inquiries

Record fields for inquiries about printers (defined as such with the PRTS value) contain the following:

Word Number: 1

Length: 8

Contains the printer name ID.

Word Number: 2

Length: 4

Contains the device number of the printer. This field contains FSS if the printer is an FSS mode printer.

Word Number: 3

Length: 8

Contains the printer status. This field may contain one of the following values: ACTIVE, INACTIVE, BUSY, HALTING, HALTED, DRAINING, or DRAINED.

Word Number: 4

Length: 8

Contains the job name.

Word Number: 5

Length: 8

Contains the FSS name (if an FSS printer).

Word Number: 6

Length: 8

Contains the FSS procedure name (if an FSS printer).

Word Number: 7

Length: 16

Contains the list of output classes the printer supports.

Record Fields for Spool (SPOL) Inquiries

Record fields for inquiries about spool volumes (defined as such with the SPOL value) contain the following:

Word Number: 1

Length: 6

Contains the current number of active spool volumes.

Word Number: 2

Length: 6

Contains the maximum number of spool volumes specified.

Word Number: 3

Length: 6

Contains the spool volume prefix.

Word Number: 4

Length: 6

Contains the percentage of spool space used.

Record Fields for the JES2 Multi-Access Spool Information (MASI)

Record fields for inquiries about multi-access spool information contain the following:

Word Number: 1

Length: 4

Contains the member name of the system.

Word Number: 2

Length: 4

Contains the system ID number. This is a numeric value (for example, 1).

Word Number: 3

Length: 8

Contains the status of this member (ACTIVE, INACTIVE, DORMANT, or UNDEF). For systems in undefined or INACTIVE (UNDEF or INACTIVE) status, all the words after this word contain an asterisk as a placeholder since no additional information is available for those systems.

Word Number: 4

Length: 1

Contains the independent mode indicator (Y or N).

Word Number: 5

Length: 1

Contains the boss indicator (Y or N).

Word Number: 6

Length: 10

Contains the last checkpoint date in the format yyyy/mm/dd.

Word Number: 7

Length: 8

Contains the last checkpoint time in the format hh:mm:ss.

Word Number: 8

Length: 8

Contains the sysplex name or an * if the system is not in a sysplex.

Word Number: 9

Length: 8

Contains the system name of the z/OS image in which this system is active. This value is obtained by the system at IPL time from the SYSNAME parameter in logical parmlib member IEASYSxx.

Word Number: 10

Length: 8

Contains the JES2 version string. Note that any blanks in the JES2 version string are replaced with underscores (for example, OS_2.10).

Word Number: 11

Length: 4

Contains the JES2 product level. This is a numeric value (for example, 33).

Word Number: 12

Length: 4

Contains the JES2 service level. This is a numeric value (for example, 0).

Word Number: 13

Length: 8

Contains the last start type for this member. This field contains one of the following values:

- SMWARM-Single member warm start
- HOT-Hot start
- QUICK-Quickstart
- ALLMEMWM-All-member warm start
- \$EMEMWM-\$E MEMBER(x) warm start
- COLD-Cold start
- MVS IPL-MVS was IPLed
- COLDFMT-Cold start with format

Word Number: 14

Length: 10

Contains the last JES2 start date in the format yyyy/mm/dd.

Word Number: 15

Length: 8

Contains the last JES2 start time in the format hh:mm:ss.

Word Number: 16

Length: 4

Contains the JES2 subsystem name (for example, JES2).

Word Number: 17

Length: 8

Contains the checkpoint HOLD value (in hundredths of a second). This value is specified on the HOLD parameter of the JES2 initialization MASDEF statement or the \$T MASDEF command.

Word Number: 18

Length: 8

Contains the minimum DORMANCY value (in hundredths of a second). This value is specified on the DORMANCY parameter of the JES2 initialization MASDEF statement or the \$T MASDEF command.

Word Number: 19

Length: 8

Contains the maximum DORMANCY value (in hundredths of a second). This value is specified on the DORMANCY parameter of the JES2 initialization MASDEF statement or the \$T MASDEF command.

Word Number: 20

Length: 8

Contains the SYNCTOL value (in seconds). Specify this value on the SYNCTOL parameter of the JES2 initialization MASDEF statement or the \$T MASDEF command.

Word Number: 21

Length: 8

Contains the actual current HOLD value (in hundredths of a second).

Word Number: 22

Length: 8

Contains the actual current DORMANCY value (in hundredths of a second).

Word Number: 23

Length: 4

Contains the ID number of the MAS member that initiated the restart for this system (\$ESYS).

Examples: OPSJES2 Functions

- The following list shows sample initiator inquiries:
 - The following example inquires about initiators that are class A and halted, and then it displays the results:

```
NumLines = OPSJES2('I','INIT','A','H')
say 'There are' NumLines 'halted class A initiators'
do NumLines
  pull line
  say line
end
```
 - The following example returns all initiators:

```
NumLines = OPSJES2('I','INIT')
```
 - The following example returns all initiators:

```
NumLines = OPSJES2('I','INIT','*')
```
 - The following example returns all initiators:

```
NumLines = OPSJES2('I','INIT','*','*')
```


- The following example returns all initiators:

```
NumLines = OPSJES2('I','INIT','*','*','*')
```

- The following example returns any initiators that has class A, B, C, and D, in any order:

```
NumLines = OPSJES2('I','INIT','ABCD','*','A')
```

- The following example returns any initiators that has class A, B, C, and D, in that exact order:

```
NumLines = OPSJES2('I','INIT','ABCD','*','E')
```

- The following example returns any initiators that has class A or B, in that order anywhere in the class definition:

```
NumLines = OPSJES2('I','INIT','AB','*','S')
```

- The following example returns all active initiators:

```
NumLines = OPSJES2('I','INIT','*','*','A')
```

- The following example returns all inactive initiators:

```
NumLines = OPSJES2('I','INIT','*','I','*')
```

- The following example returns initiators that have classes A, B, C, and D, in that order, and are active:

```
NumLines = OPSJES2('I','INIT','ABCD','A','S')
```

- The following example returns all initiators that have classes A, B, C, and D, anywhere in their setup and that are inactive:

```
NumLines = OPSJES2('I','INIT','ABCD','I')
```

■ The following list shows sample input queue inquiries:

- The following example returns information on all jobs in the input queue in class A:

```
NumLines = OPSJES2('I','INPQ','A')
```

- The following example returns information on all jobs in the input queue:

```
NumLines = OPSJES2('I','INPQ')
```

- The following example returns information on all jobs in the input queue:

```
NumLines = OPSJES2('I','INPQ','*')
```

- The following example returns information on all of the NJE/RJE lines:

```
NumLines = OPSJES2('I','LINE')
```

- The following list shows sample printer inquiries:
 - The following example returns information on the printers for class B:
`NumLines = OPSJES2('I','PRTS','B')`
 - The following example returns information on all printers in any class:
`NumLines = OPSJES2('I','PRTS')`
 - The following example returns information on all printers in any class:
`NumLines = OPSJES2('I','PRTS','*', '*', '*')`
 - The following example returns information on all printers with a print class of A, B, C, and D:
`NumLines = OPSJES2('I','PRTS','ABCD','*', '*')`
 - The following example returns information on all printers with a print class of A, B, C, and D that are active:
`NumLines = OPSJES2('I','PRTS','ABCD','A','*')`
 - The following example returns information on all drained printers:
`NumLines = OPSJES2('I','PRTS','*', 'D','*')`
 - The following example returns the spool information:
`NumLines = OPSJES2('I','SPOL')`
- The following list shows sample multi-access spool information:
 - The following example returns the multi-access spool information for all members of the MAS complex:
`NumLines = OPSJES2('I','MASI')`
 - The following example returns the multi-access spool information for active members of the MAS only:
`NumLines = OPSJES2('I','MASI', '*', 'A')`

OPSJESX Function

The OPSJESX function obtains information about jobs in the JES queue. Through REXX variables, the function returns information for each job matching the user's input filters. The value returned by the function itself is the number of jobs returned in the output.

Note: You can use the OPSJESX function in OPS/REXX. It can also be used in TOD and REQ AOF rules. The use within REQ rules is further restricted to be used within System State Manager (SSM). It cannot be used in any other AOF rule type.

Platform: The OPSJESX call operates for both JES2 and JES3.

This function has the following format:

```
var = OPSJESX('S','list of keywords')
```

S

Indicates an extended status request. This must always be the first parameter.

Keywords for OPSJESX

You can use any combination of keywords. The two basic groups of keywords are:

- Filter keywords, which are combined as logical conjunctions
- Processing keywords

Some filters allow the following standard wildcards:

- * (asterisk) for none or more characters
- ? (question mark) for just one character

Filter Keywords

The filter keywords provide filtering for JES2 and JES3 jobs. The output variables contain information for only those jobs matching the input filter criteria. For the keywords JOBID, JOBNAME, MEMBER, OJOBID, OWNER, SCHEDUL, SECLABEL, SPOOLVOL, SYSTEM, and WLMSSERVICE values should be UPPERCASE.

- The available filter keywords for the OPSJESX function are:

CLASS(*class1, class2, .., class16*)

Specifies classes for filtering.

You can use the following classes, which allow ranges:

- A-Z
- 0-9
- @ (TSU)
- \$ (started task)

Length: 1

Parameters: 16

Example: CLASS(@,A,B,C,M-R, 0-3)

DESTINATION(*destination1, ..., destination4*)

Specifies to filter jobs with specific destinations.

Destination is the default print or punch destination that is assigned to the job.

Length: 8

Parameters: 4

Wildcards: In JES2, the user ID part of the destination can contain the generic characters * and ?. This can filter jobs with a default print route code that contains a corresponding user ID routing.

Example: DESTINATION(PRODSYS1)

EXECNODE(*execnode*)

Specifies to filter jobs with specific execution nodes (NJE node where the job is to, or was, executed). This filter is only available on JES2.

Length: 8

Example: EXECNODE(PRODSYS1)

HELDJOBS(YES|NO|DUPL)

Specifies to filter the type of held jobs.

The valid values are:

YES

Jobs that are held.

NO

Jobs that are not currently held.

DUPL

Jobs are held because they are duplicate jobs.

Example: HELDJOBS(DUPL)

JOBID(*jobid*)

Specifies to filter jobs with specific job IDs.

The JOBID must start with one of the prefixes: JOB, TSU, STC, INT, or any abbreviation of these prefixes: J, T, S, I.

Length: 8

Example: JOBID(JOB12345)

JOBNAME(*jobname1*, ..., *jobname8*)

Specifies job names for filtering.

Length: 8

Parameters: 8

Wildcards: * and ?

Example: JOBNAME(*TEST*, INIT??1)

JOBTYPE(STC|TSU|JOB|APPC)

Specifies job types for filtering.

Parameters: 4

Example: JOBTYP(STC, TSU)

MEMBER(*member*)

Specifies to filter jobs that are active on specific JES MAS members. This parameter is valid on JES2 only. The job can be actively executing or active on a device on that member.

Length: 8

Wildcards: * and ?

Example: MEMBER(TEST30)

OJOBID(*ojobid*)

Specifies to filter jobs with specific original job IDs. This parameter is not supported on JES3. If the job was sent by NJE the original job ID can differ from the job ID. The *ojobid* must start with either the character J or JOB and followed by the original job number.

Length: 8

Example: OJOBID(JOB54321)

ORIGINODE(*onode*)

Specifies to filter jobs with a specific original node (NJE node where the job originated). This parameter is valid on JES2 only.

Length: 8

Example: ORIGINODE(PRODSYS1)

OWNER(*userID*)

Specifies to filter jobs with the user ID of the owner.

Length: 8

Wildcards: * and ?

Example: OWNER(JO?H*)

PHASE(*phase*)

Specifies to filter jobs with a specific job phase. This parameter is valid on JES2 and JES3.

Note: For the list of JES2 and JES3 phases, see the two bullet items in this section.

Example: PHASE(EXEC)

PRIORITY(*priority*)

Specifies to filter jobs with a specific priority. In JES2, priorities are from 0 to 15.

Length: 3

Example: PRIORITY(15)

SCHEDUL(*schedule*)

Specifies to filter jobs with a specific scheduling environment. Jobs have scheduling environments that are assigned to them only if they have completed conversion processing and the jobs have not completed execution processing.

Length: 16

Wildcards: * and ?

Example: SCHEDUL(DFG)

SECLABEL(*seclabel*)

Specifies to filter jobs with a specific *seclabel* that the security product has assigned to the job.

Length: 8

Wildcards: * and ?

Example: SECLABEL(SYSHIGH)

SPOOLVOL(*svol1*, ..., *svol4*)

Specifies the spool volume. This parameter is valid on JES2 only.

Length: 6

Parameters: 4

Example: SPOOLVOL(SPOL10)

SUBMITTER(*submitter*)

Specifies to filter jobs with a specific submitter. This parameter is valid for JES3 on z/OS v1r7 or higher only.

Length: 8

Wildcards: * and ?

Example: SUBMITTER(ADMIN*)

SYSTEM(*system*)

Specifies to filter jobs that are active on the specified MVS system. The job can be actively executing or active on a device on that system.

Length: 8

Wildcards: * and ?

Example: SYSTEM(SYS1)

WLMSERVICE(*service*)

Specifies to filter jobs with a specific WLM service class. This parameter is valid on JES2 only.

Length: 8

Example: WLMSERVICE(ONLTEST)

- The JES2 job phase keycodes are:

INPUT

Job is active in input processing.

WTCONV

Job is queued for the, conversion.

CONV

Job is actively converting.

VOLWT

Job is queued for SETUP (not currently used by JES2 code).

SETUP

Job is active in SETUP (not currently used by JES2 code).

SELECT

Job is queued for the execution.

ONMAIN

Job is actively executing.

SPIN

JES2 is processing SPIN data sets for the JOB.

WTBKDN

Job is queued for output processing.

BRKDOWN

Job is active in output processing.

OUTPT

Job is on the hard copy queue.

WTPURG

Job is queued for a purge.

PURG

Job is being purged.

RECV

Job is active on an NJE SYSOUT receiver.

WTXMIT

Job is queued for execution on another NJE node.

XMIT

Job is active on an NJE JOB transmitter.

EXEC

Job has not completed execution (combines multiple states in one phase request)

POSTEX

Job has completed execution (combines multiple states in one phase request)

- The JES3 job phase keycodes are:

NOSUB

No subchain exists.

FSSCI

Job is active in conversion/interpretation in an FSS address space.

PSCBAT

Job is awaiting postscan (batch)

PSCDSL

Job is awaiting postscan (demand select).

FETCH

Job is awaiting volume fetch.

VOLWT

Job is awaiting start setup.

SYSSEL

Job is awaiting or active in the MDS system select processing.

ALLOC

Job is awaiting resource allocation.

VOLUAV

Job is awaiting unavailable volumes.

VERIFY

Job is awaiting volume mounts.

SYSVER

Job is awaiting or active in the MDS system verification processing.

ERROR

Job encountered an error during the MDS processing.

SELECT

Job is awaiting selection on main.

ONMAIN

Job is scheduled on main.

BRKDWN

Job is awaiting breakdown.

RESTRT

Job is awaiting the MDS restart processing.

DONE

The main part and the MDS processing complete for the job.

OUTPT

Job is awaiting output service.

OUTQUE

Job is awaiting output service writer.

OSWAIT

Job is awaiting rsvd services.

CMPLT

Output service complete for this job.

DEMSEL

Job is awaiting selection on main (demand select job).

EFWAIT

The ending function request waiting for an I/O completion.

EFBAD

The ending function request that is not processed.

MAXNDX

The maximum request index value.

Processing Keywords

This section provides the processing keywords.

STEM(*varStem*)

Creates an output variable with the specified name. If omitted, the default name OPSJESX will be used.

varStem.0

Indicates the number of returned jobs.

Length: 50

Example:

STEM(MYOUT)

Note: The output variable is used as a stem for the number of jobs returned and for each output section returned. See the Output Variables section for further details.

JESSUBSYS(*name*)

Defines the name of the JES subsystem. If omitted, the JES subsystem provided by the system is used.

Length: 4

Example:

JESSUBSYS(JES3)

MAX(*n*)

Limits the number of output variables.

- If omitted, the default value of 2000 is used.
- If MAX(0) is specified, the output is unlimited.

Length: 10

Example:

MAX(2000)

SECTIONS(*keycodes*)

Specifies which sections to return in the output.

Valid parameter values are:

AFFS

Indicates the Member Affinity section

ALL

Indicates all sections

J2TR

Indicates the JES2 terse section

J3TR

Indicates the JES3 terse section

SCHD

Indicates the Execution Scheduling section

SCHS

Indicates the Schedulable Systems section

SCLF

Indicates the SECLABEL Availability section

You can use any combination of keycodes. However, the ALL keycode cannot be combined with other keycodes. Use the ALL keycode if you want all sections returned.

The JQTR section is always returned. Therefore, JQTR is not included as a keycode.

If SECTION is omitted, only the JQTR is returned.

The job information returned for each section is provided in Section Variable Output Mappings. Refer to the mapping section for details on what job information is returned for each section.

Note: The sections above correspond to the Job Information Elements for the Extended Status Function Call (SSI 80) as described in the IBM guide *z/OS MVS Using the Subsystem Interface*. To determine the name of the Job Information Element that corresponds to a keycode, add the prefix STAT to the keycode. For example, STATAFFS is the name of the Job Information Element that corresponds with keycode AFFS.

Example:

```
SECTIONS(AFFS, J3TR)
```

Output Variables

The output is stored in an output variable. The default output variable name is OPSJESX. You can change the default name by specifying a varStem value on the STEM keyword.

There are two kinds of output variables:

- The number of jobs

varStem.0

The number of jobs in the output is returned to the variable named varStem.0. The OPSJESX function returns the same value that is contained in varStem.0.

- The sections

`varStem_SeNa.n`

SeNa

Specifies the section name, which can be the following: JQTR, J2TR, AFFS, SCHD, SCHS, SCLF, J3TR

n

Identifies the output number of the corresponding job (n=1 for 1st returned job, n=2 for 2nd returned job, and so on).

The sections returned correspond to the sections requested by keycode value on the input SECTIONS keyword. The JQTR section is always returned.

If a section is empty (does not exist) for a job, the variable for that section is not created for that job.

The individual pieces of job information in each section are Words in the section variable. The mapping of the WORDs in each section is provided in Section Variable Output Mappings.

Section Variable Output Mappings

These sections correspond to the Job Information Elements for the Extended Status Function Call (SSI 80) as described in the IBM guide *z/OS MVS Using the Subsystem Interface*. To determine the name of the Job Information Element that corresponds to a keycode, add the prefix STAT to the keycode. For example, STATJQTR is the name of the Job Information Element that corresponds with keycode JQTR.

The Words returned for each section correlate to fields provided in the IBM mapping for each Job Information Element in IAZSSST macro. The fields are also described under Job Information Elements for the Extended Status Function Call in the IBM guide *z/OS MVS Using the Subsystem Interface*. In the descriptions below, the correlating IBM field name is given in parenthesis.

■ JQTR

This section contains common fields, which are common for all job types.

Word Number 1

Length: 8

Contains: Job name (STTRNAME)

Word Number 2

Length: 8

Contains: Job id (STTRJID)

Word Number 3

Length: 8

Contains: Original Job id (STTROJID)

Word Number 4

Length: 8

Contains: Job class (STTRCLAS)

Word Number 5

Length: 8

Contains: Origin Node (STTRONOD)

Word Number 6

Length: 8

Contains: Execution Node (STTRXNOD)

Word Number 7

Length: 8

Contains: Default Print Node (STTRPRND)

Word Number 8

Length: 8

Contains: Default Print Remote Name (STTRPRRE)

Word Number 9

Length: 8

Contains: Default Punch Node (STTRPUND)

Word Number 10

Length: 8

Contains: Default Punch Remote (STTRPURE)

Word Number 11

Length: 8

Contains: Owner User ID (STTROUID)

Word Number 12

Length: 8

Contains: SECLABEL (STTRSECL) – will show INACTIVE if job is not active

Word Number 13

Length: 8

Contains: MVS system (STTRSYS) – will show INACTIVE if job is not active

Word Number 14

Length: 8

Contains: JES2 member (STTRMEM) – will show INACTIVE if job is not active

Word Number 15

Length: 18

Contains: Name of device for job (STTRDEVN) – will show N/A if job is not active

Word Number 16

Length: 6

Contains: Job Phase (STTRPHAZ) – in IBM mapping this is a one byte field. For OPSJESX, it is translated into text for the Phase. The possible values are the same as listed for the PHASE keycode above.

Word Number 17

Length: 9

Contains: Job Hold indicator (STTRHOLD) – in IBM mapping this is a one byte field. For OPSJESX, it is translated into text for the hold status. The possible values are: NOTHELD, HELD, or DUPLICITY.

Word Number 18

Length: 4

Contains: Job type (STTRJTYP) – in IBM mapping this is a one byte field. For OPSJESX, it is translated into text for the job type. The possible values are: STC, TSU, JOB, or APPC.

Word Number 19

Length: 3

Contains: Job priority (STTRPRIO)

Word Number 20

Length: 2

Contains: Job ARM status (STTRARMS). It contains 2 hexadecimal numbers.

Word Number 21

Length: 2

Contains: Job miscellaneous indicators (STTRMISC). It contains 2 hexadecimal numbers.

Word Number 22

Length: 8

Contains: Job completion indicator (STTRXIND) – in IBM mapping this is a one byte field. For OPSJESX, it is translated into text for the job completion indicator. The possible values are: N/A, NORMAL, CC, JCLERR, CANCELED, ABEND, CABEND, SECERR, or EOM.

Word Number 23

Length: 7

Contains: Max return code (STTRMXCC) – in IBM mapping this is a 4 byte field. For OPSJESX, it is translated into a 7 byte field formatted as:

SACUAC

- SAC – (System abend code) contains three hexadecimal numbers
- UAC – (User abend code) contains three hexadecimal numbers

Word Number 24

Length: 11

Contains: Job position on class or phase queue (STTRQPOS)

Word Number 25

Length: 8

Contains: Binary job number (STTRJNUM)

Word Number 26

Length: 7

Contains: Percent SPOOL Utilization (STTRSPUS) – format x.xxxx%

Word Number 27

Length: 8

Contains: MVS system name for log if SYSLOG job (STTRSLOG)

■ J2TR

This section is created only if the job came from a JES2 subsystem. It contains JES2 specific information common for all job types.

Word Number 1

Length: 2

Contains: IBM mapping field (STJ2FLG1). It contains 2 hexadecimal numbers.

Word Number 2

Length: 8

Contains: The JES2 job key for the job (STJ2JKEY). It contains 8 hexadecimal numbers.

Word Number 3

Length: 16

Contains: The spool token associated with the job (STJ2SPOL). It contains 16 hexadecimal numbers.

Word Number 4

Length: 11

Contains: The number of track groups of SPOOL space used by the job (STJ2SPAC). Possible value -1 indicates the count is not available.

Word Number 5

Length: 4

Contains: Default print node (STJ2DPNO). It contains 4 hexadecimal numbers.

Word Number 6

Length: 4

Contains: default print remote (STJ2DPRM). It contains 4 hexadecimal numbers.

Word Number 7

Length: 8

Contains: Default print user ID (STJ2DPUS)

Word Number 8

Length: 4

Contains: Input node (STJ2INPN). It contains 4 hexadecimal numbers.

Word Number 9

Length: 4

Contains: Execution node (STJ2XEQN). It contains 4 hexadecimal numbers.

Word Number 10

Length: 8

Contains: Index of JQE (STJ2JQEI). It contains 8 hexadecimal numbers.

Word Number 11

Length: 2

Contains: Offload status mask (STJ2OFSL). It contains 2 hexadecimal numbers.

Word Number 12

Length: 2

Contains: Busy byte (STJ2BUSY). It contains 2 hexadecimal numbers.

- ADFS

This section is created if the job has affinities to a subset of members.

Word Number 1

Length: 5

Contains: The number of members for which the job has affinity (STAFNUM)

Word Number 2..Word Number n ,

where n = (number of members from Word 1) + 1 to a maximum of n = 65

Length: 8

Contains: Each Word contains a member from the list of members (STAFMEMB).

Note: If the number of members is greater than 64, then only first 64 members are in the output variable, but the number of members remains unchanged.

- SCHED

This section is created if the job is scheduled for execution. This section cannot be returned by JES3 subsystem.

Word Number 1

Length: 2

Contains: Reasons why the job will not run (STSCAHL). It contains 2 hexadecimal numbers.

Word Number 2

Length: 3

Contains: Job class mode (STSCFLG1). Possible values are: JES or WLM.

Word Number 3

Length: 4

Contains: ASID where job is executing (STSCASID). It contains 4 hexadecimal numbers.

Word Number 4

Length: 8

Contains: Service class (STSCSRVC)

Word Number 5

Length: 11

Contains: Estimated time to execute in seconds for the job. (STSCSTT). It is available only if the job is awaiting execution or is scheduled to WLM class or is not held or can currently run. -1 is possible value.

Word Number 6

Length: 16

Contains: Scheduling environment required by the job (STSCSENV)

Word Number 7

Length: 11

Contains: Position of the job on a WLM service class queue (STSCQPOS)

Word Number 8

Length: 11

Contains: Number of jobs on this WLM service class queue (STSCQNUM)

Word Number 9

Length: 11

Contains: Number of active jobs on this WLM service class queue (STSCQACT)

Word Number 10

Length: 11

Contains: Average queue time for jobs in the WLM service class (STSCAVGQ).
-1 is possible value.

Word Number 11

Length: 11

Contains: Actual queue time for the job (STSCQTIM). -1 is possible value.

■ SCHS

This section is created if the job is scheduled for execution, requires a scheduling environment (environment is available on at least one system). This section cannot be returned by JES3 subsystems.

Word Number 1

Length: 5

Contains: The number of systems which have required scheduling environment (STSSNUM)

Word Number 2..Word Number n ,

where $n = (\text{number of systems from Word 1}) + 1$ to a maximum of $n = 65$

Length: 8

Contains: Each Word contains a system from the list of systems (STSSSYS)

Note: If the number of systems is greater than 64, then only first 64 systems are in the output variable, but the number of systems remains unchanged.

■ SCLF

This section is created if the SECLABEL by system RACF option is enabled and the job is queued for conversion processing or execution. This section cannot be returned by JES3 subsystems.

Word Number 1

Length: 5

Contains: The number of systems where the SECLABEL is active. (STSLNUM)

Word Number 2..Word Number n ,

where $n = (\text{number of systems from Word 1}) + 1$ to a maximum of $n = 65$

Length: 8

Contains: Each Word contains a system from the list of systems (STLSYS).

Note: If the number of systems is greater than 64, then only first 64 systems are in the output variable, but the number of systems remains unchanged.

■ J3TR

This section is created only if the job is owned by a JES3 subsystem.

Word Number 1

Length: 16

Contains: Spool data token (STJ3SPOL). Token contains 16 hexadecimal numbers.

Word Number 2

Length: 64

Contains: List of reasons, why job is waiting to run (STJ3JSTT). Each reason contains 2 hexadecimal numbers. Reasons are concatenated. The maximum number of reasons is 32.

Word Number 3..Word number n ,

where $n = (\text{number of reasons}) + 2$ to a maximum of $n = 34$

Length: 8

Contains: Each Word contains a system from the list of systems (STJ3JSTM). Each system name corresponds to the reason.

Output Variable Example

Note: Additional examples are provided in the topic OPSJESX Function Examples.

The following OPSJESX call returns job information for all class A jobs. For each job, the JQTR and J2TR sections are returned.

```
rtvl = OPSJESX("S","CLASS(A) SECTION(J2TR)")
```

If there are 3 class A jobs returned in the output, the output variables are:

OPSJESX.0 = 3

number of jobs returned

OPSJESX_JQTR.1

JQTR section for 1st job returned

WORDS 1 -27 contain job information as documented for JQTR above

OPSJESX_JQTR.2

JQTR section for 2nd job returned

WORDS 1 -27 contain job information as documented for JQTR above

OPSJESX_JQTR.3

JQTR section for 3rd job returned

WORDS 1 -27 contain job information as documented for JQTR above

OPSJESX_J2TR.1

J2TR section for 1st job returned

WORDS 1 -12 contain job information as documented for J2TR above

OPSJESX_J2TR.2

J2TR section for 2nd job returned

WORDS 1 -12 contain job information as documented for J2TR above

OPSJESX_J2TR.3

J2TR section for 3rd job returned

WORDS 1 -12 contain job information as documented for J2TR above

Note: If a section does not exist for a job, the section variable for that job will not be created. For example, if a J2TR section did not exist for job 2, the OPSJESX_J2TR.2 variable would not be created.

Return and Reason Codes

The following return codes are returned in the OPSRC variable. Associated reason codes are returned in the OPSRE variable.

0 – Successful completion.

1 -- Error in OPS/MVS related processing

Reason codes:

- 2 – parsing error
- 4 – GREXX variable creation error

Note: if the creation of OPSRC or OPSRE variable fails, then this message will be generated: "CLIST/REXX variable access error, RC=4, detected at OPSJESX+X'nnnnnnnn'", where nnnnnnnn is offset in OPSJESX module.

- 8 – Invalid AOF rule detected
- 12 - The subsystem is not active
- 16 - A serious product control block error

Note: If an ABEND occurs, the ABEND code will be used as the reason code. The ABEND code can be eight hexadecimal numbers.

2 – Subsystem error as reported by IBM IEFSSREQ processing.

Reason codes:

Note: The reason codes below correspond to the IBM IEFSSREQ return codes as documented in the IBM manual *z/OS Using the Subsystem Interface*. For more information, see the IBM manual.

- 4 - The specified subsystem does not support the extended status function call.
- 8 - The specified subsystem exists but is not active.
- 12 - The specified subsystem is not defined to MVS.
- 16 - The function code 80 (Extended Status Function Call) is greater than the maximum number of functions supported by the specified subsystem.
- 20 - Either the SSIB control block or the SSOB control block has incorrect lengths or formats.
- 24 - The SSI has not been initialized.

Note: The return codes below (RC=4 and higher) correspond to the SSOBRETN return codes for the Extended Status Function Call (SSI 80) as documented in the IBM manual *z/OS MVS Using the Subsystem Interface*. Refer to the IBM manual for more information.

4 – search argument error – the search arguments cannot be used as specified

8 – search argument logic error – the reason code further explains the logic error

- The reason code values correspond directly to the STATREAS value returned for the Extended Status Function Call (SSI 80) as documented in the IBM manual *z/OS MVS Using the Subsystem Interface*. Due to the significant number of possible values, they are not documented individually here. Please see the IBM *z/OS MVS Using the Subsystem Interface* manual for the specific codes and their meanings.

12 – Request type error – the request type in STATTYPE is not valid.

Examples of OPSJESX Function

The following examples provide scenarios for using the OPSJESX Function.

■ Example 1 – Duplicated jobs

This example will print the job name and job ID of all duplicated jobs.

```
rtvl = OPSJESX("S","HELDJOB(DUPL)")

if OPSRC = 0 then
  do i = 1 to OPSJESX.0
    say word(OPSJESX_JQTR,i,1) word(OPSJESX_JQTR,i,2)
  end
```

■ Example 2 - JOBNAME

This example will print the job name and job ID of all jobs where the job name either contains word TEST or contains three characters.

```
rtvl = OPSJESX("S","JOBNAME(*TEST*,'???')")

if OPSRC = 0 then
  do i = 1 to OPSJESX.0
    say word(OPSJESX_JQTR,i,1) word(OPSJESX_JQTR,i,2)
  end
```


■ Example 3 – More parameters

This example will print the job name, job ID, class, and J2TR section if exists. It uses filters for class (from A to K, class Z and from 1 to 3) and owner starting with AB. The output variable name is O. Also J2TR section is created and the output is limited to 25 jobs.

```
rtvl = OPSJESX("S","CLASS(A-K,Z,1-3) OWNER(AB*)",
               "STEM(O) SECTION(J2TR) MAX(25)")
```

```
if OPSRC = 0 then
do i = 1 to O.O
  parse var O_JQTR.i jobName jobId .class .
  say jobName jobId class
  if symbol("O_J2TR."||i) = "VAR" then
    say O_J2TR.i
  say
end
```

■ Example 4 - JES3

This example will:

- Print the job name of all jobs, which are APPC, TSU or STC and with priority 15
- Run the OPSJESX function under JES3 subsystem, with an unlimited number of output jobs
- Create the J3TR section

```
rtvl = OPSJESX("S","JOBTYPE(APPC, TSU, STC) PRIORITY(15)",
               "JESSUBSYS(JES3) MAX(0) SECTIONS(J3TR)")
```

```
if OPSRC = 0 then
do i = 1 to OPSJESX.O
  say word(OPSJESX_JQTR.i,1)
  if symbol("OPSJESX_J3TR."||i) = "VAR" then
    say OPSJESX_J3TR.i
  say
end
```

OPSLIKE Function

The OPSLIKE function lets you use wildcard characters to simplify coding text strings in OPS/REXX or AOF rules.

This function has the following format:

```
var = OPSLIKE(wildcard string, target string, [wildcard character string])
```

wildcard string

Contains the wild card characters.

target string

Contains the resulting string that is being tested.

wildcard characters string

Contains an optional argument to specify the character designated to match any number of characters (default='*') followed by one or more characters that are designated to match any single character value (default='?').

Default: '*?'

The possible condition codes returned by the OPSLIKE function follow:

1

True, a match is found.

0

False, a match is not found.

Note: When the target string is matched and the remaining pattern string contains only wild card characters and blanks, then a match condition, or 1, is returned.

Examples: OPSLIKE Statement

- The following statement returns a value of 0 (false):

```
var=OPSLIKE('XOOOO','OPSLOGOOOOOOOOOOOOOOO','*?')
```

- The following statement returns a value of 0 (false):

```
var=OPSLIKE('XOOOO','OPSLOGOOOOOOOOOOOOOOO')
```

- The following statement returns a value of 1 (true):

```
var=OPSLIKE('XYZ?ABC*','XYZ$ABCDEFG')
```

- The following statement returns a value of 1 (true):

```
var=OPSLIKE('*XYZ?ABC','123XYZ+ABC')
```

- The following statement returns a value of 0 (false):

```
var=OPSLIKE('*XYZ','XYZ1234')
```

- The following statement returns a value of 1 (true):

```
var=OPSLIKE('*XYZ*','XYZ1234')
```

Note: The following two syntax error messages in the function call statement will cause an OPS/REXX error and terminate the program:

- ERROR 69 - FUNCTION HAS TOO FEW ARGUMENTS
- ERROR 70 - FUNCTION HAS TOO MANY ARGUMENTS

OPSLOG Function

The OPSLOG function provides the ability to extract messages from OPSLOG using filtering criteria. The function returns one or more messages from the OPSLOG data area to the external data queue, REXX or CLIST variables or the terminal.

Note: Except for cross system OPSLOG retrieval, you may use the OPSLOG function in AOF rules.

This function has two formats:

- The basic OPS/REXX function multi-argument format provides simple message extraction by message time or number from the active OPSLOG with only a single message id filter. Output is always routed to the external data queue and consists of only the message text.
- The expanded REXX function single argument format provides keyword syntax operands that allow specification of multiple filters with multiple values. The capabilities provided are comparable to the OPSVIEW OPSLOG profile and display commands. Different OPSLOG names and cross-system operation are also supported. Output consists of the message text preceded by keyword selected message attribute fields. This REXX function format can also be invoked as a TSO/E REXX function or command processor.

Basic Format

This Basic format is:

```
var = OPSLOG(opcode,select,n,msgid)
```

opcode

Currently, supports only uppercase L for List.

select

Identifies which message is retrieved first.

- If the *select* value has the form *hh:mm:ss*, CA OPS/MVS retrieves messages starting at the specified time (or later if no message occurs at the indicated time).
- If *select* is a positive integer, it indicates an absolute message number. Messages numbered from *select* to *select+n* are retrieved.
- If *select* is a negative integer, it indicates a relative time offset. OPS/REXX retrieves messages starting from the current time minus the indicated number of seconds. For example, the following function call retrieves two messages starting at the one issued five seconds ago:
`var = OPSLOG('L', -5, 2)`

n

Indicates the maximum number of messages to be retrieved and must be a positive number. The value returned is the number of lines retrieved.

msgid

Specifies an optional, specific, or wild card message ID filter that is applied to the specified range of messages. Wild card specification is done with a trailing '*'.

Expanded Format

This function is universal for calling from OPS/REXX scripts, CLIST scripts, or from the TSO environment and has the following format:

```
var = OPSLOG('opcode TIME(value) DATE(ddmmmyyyy) | MSGNUM(number) [optional keywords]')
      {opcode}
      {TIME(hh:mm:ss|-sec) DATE(ddmmmyyy)|MSGNUM(message number)}
      [MSGCOUNT(n)]
      [LOGNAME(logname)]
      [JOBNAME(jobname1,...jobname8)]
      [MSGID(msgid1,...msgid8)]
      [SYSNAME(sysna1,...sysna4)]
      [USER(user1,...user4)]
      [COLORS(color1,...color4)]
      [RULESET(ruleset1.rulename1,ruleset2.rulename2)]
      [EVENTTYPE(evntype1,...evntype20)]
      [ASD(asid1,...asid4)]
      [EXTTYPE(extype1,...extype3)]
      [SCANTEXT1('Case sensitive text,fromcol,tocol)]
      [SCANTEXT2('Case sensitive text,fromcol,tocol)]
      [SCANTEXT3('Case sensitive text,fromcol,tocol)]
      [OUTCOLS(outcol1,...outcol44)]
      [CMDRESP(cmdresp)]
      [PREFIX(variable prefix)]
      [SUBSYS(ssid)]
      [SYSTEM(msfid)]
      [SYSWAIT(syswait)]
```

opcode

Currently, supports only the EXTRACT option.

TIME(hh:mm:ss|-sec) DATE(ddmmmyyyy) | MSGNUM(message_number)

These keywords enable you to start message selection at an explicit date and/or time. Time may be specified as hh:mm:ss or as a negative seconds offset relative to the current time. Alternatively, you may start message selection at an explicit OPSLOG message number. This could be useful when many messages are logged at the same time of day.

TIME(hh:mm:ss|-sec)

hh:mm:ss selects messages starting at the specified time or later when no messages exist at the indicated time. The minimum specification for time is h. When time is omitted in conjunction with DATE, the time 00:00:00 is assumed. When minutes or seconds are specified, the value must be two numeric digits.

-sec indicates a relative time offset. Selected messages are retrieved starting from the current time minus the indicated number of seconds.

DATE(ddmmmyyyy)

ddmmmyyyy selects messages starting at the specified date or later when messages exist on the indicated date. The mmm portion of the date must be one of the three letter month abbreviations such as JAN for January. The year can be four digits, the last two digits, or omitted for the current year. When DATE, is omitted the current date is assumed.

Note: The year portion can be either 'nothing' (default to current year), 'yy' (last two digits of the yyyy format), 'yyyy' (all four characters of the yyyy format).

Valid OPSLOG function calls with various date specifications follow:

```
rtvl = OPSLOG('Extract TIME(-10) DATE(22JUL)')
rtvl = OPSLOG('Extract TIME(-10) DATE(22JUL11)')
rtvl = OPSLOG('Extract TIME(-10) DATE(22JUL2011)')
```

MSGNUM(message_number)

MSGNUM is the exact OPSLOG sequence number of the starting message. All messages numbered starting with the MSGNUM specified are retrieved.

Note: TIME and DATE keywords are mutually exclusive with MSGNUM.

Example: Retrieve Messages by Specific Time of Day

This example retrieves messages that were logged starting at 10:15:00 AM:

```
TIME(10:15:00)
```

Example: Retrieve Messages by Specific Time of Day and Date

This example retrieves messages that were logged starting at 8:20 AM on July 7:

```
TIME(08:20) DATE(07JUL)
```

Example: Retrieve Messages by Time Offset

This example retrieves messages that were logged starting 240 seconds ago:

```
TIME(-240)
```

Example: Retrieve Messages by Number

This example retrieves messages that have absolute sequence numbers with a value of 10 or greater:

```
MSGNUM(10)
```

MSGCOUNT(*n*)

(Optional) Specifies the maximum number of messages to be retrieved and must be a positive number. The value returned from OPSLOG is the real number of lines retrieved. The number of messages that can actually be retrieved is limited by the size of the external data queue and the REXX workspace size.

Default: 1000

Example: Retrieve a Maximum Number of Messages

This example retrieves up to 500 messages.

```
MSGCOUNT(500)
```

LOGNAME(*logname*)

(Optional) Retrieves a particular OPSLOG by name.

Limit: Up to 16 characters.

Default: LIVE

Example: Retrieve Messages from a Particular OPSLOG by Name

This example retrieves messages from the OPSLOGBCK log:

```
LOGNAME(OPSLOGBCK)
```

JOBNAME(*jobname1*,...,*jobname8*)

(Optional) Filters retrieved messages generated by specific jobs.

Note: Use \ in front of the job name to exclude it from retrieval.

Note: You can filter using a wildcard *.

Limit: Up to 8 characters plus one character for the exclusion character (\).

Example: Retrieve Messages Generated by a Particular Job

This example filters retrieved messages by job name MYJOB:

```
JOBNAME(MYJOB)
```

MSGID(*msgid1*,...,*msgid8*)

(Optional) Retrieves messages according to their specific event identifier.

Note: Use \ in front of the event identifier to exclude it from retrieval.

Note: You can filter using a wildcard *.

Limit: Up to 10 characters plus one character for the exclusion character (\).

Example: Retrieve Messages by Message ID

This example only retrieves messages with the event identifier IEF403I:

```
MSGID(IEF403I)
```

SYSNAME(*sysna1*,...,*sysna4*)

(Optional) Retrieves messages generated on a specific system.

Note: Use \ in front of the event identifier to exclude it from retrieval.

Note: You can filter using a wildcard *.

Limit: Up to 8 characters plus one character for the exclusion character (\).

Example: Retrieved Messages by the System on Which They Were Created

This example filters retrieved messages that were created on the system:

SYSNAME(PROD1)

USER(*user1*,...,*user4*)

(Optional) Retrieves messages with the event user data value from the last AOF rule processed.

Note: Use \ in front of the event identifier to exclude it from retrieval.

Note: You can filter using a wildcard *.

Limit: Up to 8 characters plus one character for the exclusion character (\).

COLORS(*color1*,...,*color4*)

(Optional) Retrieves messages based on the color of the message.

Note: Use \ in front of the keyword to exclude it from the filter.

Possible Values: GREEN, BLUE, RED, WHITE, PINK, YELLOW, TURQ

Example: Retrieve Messages by Specific Colors

This example filters retrieved messages that are either GREEN or PINK:

COLORS(GREEN,PINK)

RULESET(ruleset1.ruleaname1,ruleset2.ruleaname2)

(Optional) Retrieves messages according to the first ruleset.ruleaname that processed the event.

Note: Use \ in front of the keyword to exclude it from the filter.

Note: You can filter using a wildcard *.

Example: Retrieve Messages First Processed by a Specific Rule

This example filters messages first processed by the rule OPSTEST in ruleset TEST:

```
RULESET(TEST.OPSTEST)
```

Example: Exclude Messages First processed by a Specific Ruleset:

This example excludes messages that were processed by any rule in ruleset TEST

```
RULESET(\TEST.*)
```

Example: Retrieve Messages First Processed by a Specific Rule

This example filters messages first processed by rule OPSTEST in any ruleset:

```
RULESET(*.OPSTEST)
```

EVENTTYPE(evntype1,...,evntype20)

(Optional) Filters returned messages according to event type.

Note: Use \ in front of the keyword to exclude it from the filter

Possible Values: MSG, RUL, CMD, DIS, DOM, ENA, EOM, GLV, OMG, REQ, SEC, TOD, SCR, ARM, EOS, EOJ, TLM, USS, API, ALL

Example: Retrieve Messages by Event Type

This example filters messages by the MSG event type:

```
EVENTTYPE(MSG)
```

ASID(asid1,...,asid4)

(Optional) Retrieves messages generated by the address space identifier of the creator of the event.

Limit: Up to four ASIDs could be specified each with a four-character hexadecimal value.

Example: Retrieves Messages by Address Space Identifiers:

This example filters messages by ASID 028F:

```
ASID(028F)
```

EXITTYPE(*exttype1*,...,*exttype3*)

(Optional) Retrieves messages based on the exit type that captured the event.

Possible Values: NONE, MVS, JES3, IMS, OMG, DSN, TRA, NIP, CICS, CNSV, CA7

Example: Retrieve Messages by Exit Type

This example filters messages by the JES3 exit type:

```
EXITTYPE(JES3)
```

SCANTEXT1(*'text'*,*fromcol*,*tocol*)****SCANTEXT2(*'text'*,*fromcol*,*to**col*)****SCANTEXT3(*'text'*,*fromcol*,*to**col*)**

(Optional) Retrieve messages with specific case sensitive text between the variables *fromcol* and *to**col*.

Example: Search for a Text String Between Two Columns

This example searches for the text string "my text" between columns 1 and 7.

```
SCANTEXT1('my text',1,7)
```

OUTCOLS(*outcol1*,...,*outcol44*)

(Optional) Choose columns to be displayed in word-delimited output record order prior to the message text. Column data is displayed in the order that column names are specified. Each column value is a single REXX word except for ROUTE and ROUTEX which create two words. Blank or null values are replaced with the value NONE. The column values correspond to the values displayed by the OPSVIEW OPSLOG dialogue except where a blank value is encountered.

Possible values: JOBNAME, TIME, DATE, ASID, MSFID, MSFDEST, MSGNO, ELAPSED, SYSID, AFLAGS, FLAGS, DISP, OPSFLAGS, RELEASE, JOBNM, JOBID, AUTOTOKN, XCONID, CONID, CONSNAME, IMSID, IMSTYPE, JES3CLAS, ROUTE, WTOID, TIMESTMP, SYSNAME, LENGTH, SPECIAL, DSPNAME, USER, USERX, TERMNAME, USERID, EVENT, ADDRESS, COLOR, TOKEN, AUTOTOKX, EXITTYPE, MSGID, RULESET, COUNT, EVENTID, ROUTEX

Example: Display the TIME and JOBNAME Columns

This example displays the TIME and JOBNAME columns:

```
OUTCOLS(TIME,JOBNAME)
```

CMDRESP(*XDQ*|*REXX*|*CLIST*|*TERM*)

(Optional) Determines where the output from OPSLOG is directed.

Possible Values:

XDQ

External data queue

REXX

REXX compound variables

CLIST

CLIST variables

TERM

Output is directed to the terminal

The output default values of this command depend on the environment in which you are running the command. The following are the default values according to the execution environment in which you run OPSLOG:

- When executing OPSLOG as TSO command processor, the default value is TERM.
- When executing OPSLOG in REXX, the default value is XDQ.
- When executing OPSLOG in a CLIST, the default value is CLIST.
- The following are possible run-time environment and output combinations:
- When executing OPSLOG as a TSO command processor, the only possible output is TERM.
- When executing OPSLOG in REXX, the possible outputs are REXX, TERM, and XDQ.
- When executing OPSLOG in a CLIST, the possible outputs are CLIST and TERM.

Example: Direct Output Records

This example directs output records into REXX variables:

```
CMDRESP(REXX)
```

PREFIX(*name*)

(Optional) Defines a prefix name for created variables.

Default:

For REXX variables OPLG. (followed by a period)

For CLIST variables OPLG (not followed by a period)

Example: Define a Prefix for Created Variables

This example defines a REXX stem variable prefix for the output variables:

```
PREFIX(MSGLOG.)
```

SUBSYS(ssid)

(Optional) Directs the OPSLOG request to a specific CA OPS/MVS subsystem name on the same system.

Example: Specify an Alternate Subsystem Name on the Same System.

This example directs the OPSLOG request to CA OPS/MVS subsystem OPSQ on the same system.

SUBSYS(OPSQ)**SYSTEM(msfid)**

(Optional) Specifies the name of the remote or local MSF system that is to receive the command. The value of MSFID can contain from one to eight characters.

If you have issued the MSF DEFAULT SYSTEM command, the value you specify for the SYSTEM keyword overrides any SYSTEM values you may have previously specified for the REXX program or rule.

Example: Specify the MSFID of the System that Receives the Command

This example sends the command to the system with MSFID SYSX:

```
SYSTEM(SYSX)
```

SYSWAIT(seconds)

(Optional) Specifies the maximum time that the command waits for output from the remote system. You may specify a value between 0 and 300 seconds.

If you have issued the MSF DEFAULT SYSWAIT command, the value you specify for the SYSWAIT keyword overrides any SYSWAIT values you may have previously specified for the REXX program or rule.

Example: Define a Wait Time of 30 Seconds

This example defines a wait time of 30 seconds:

```
SYSWAIT(30)
```

OPSLOG Function Examples

The following examples show common ways that you can use the OPSLOG function in the available environments:

Example: OPSLOG Basic Format for OPS/REXX

This example retrieves two messages from the external data queue starting with the message issued 5 seconds ago with var representing the number of lines to be retrieved.

```
var = OPSLOG('L',-5,2)
```

Example: OPSLOG Expanded Format for OPS/REXX or TSO/E REXX

This example retrieves four messages starting with the message issued at or after 8:30 AM. The output goes to REXX compound variables with the prefix ABC. Variable ABC.0 has the count of retrieved messages. ABC.n variables contain the retrieved messages. The message time and date are prefixed to the message text. A REXX parse statement can be used to obtain the individual data elements: Parse Var ABC.1 time date msgtext.

```
var = OPSLOG('EXTRACT MSGCOUNT(4) TIME(08:30) OUTCOLS(TIME,DATE)
CMDRESP(REXX) PREFIX(ABC)')
```

Example: OPSLOG Expanded Format for TSO/E Command

This example retrieves ten messages not generated by job JOB00001 starting at the current time minus 10 seconds. Any output records are created with the TIME and JOBNAME columns prefixed to the message text. The output goes to the terminal.

```
OPSLOG EXTRACT MSGCOUNT(10) TIME(-10) JOBNAME(\JOB00001)
OUTCOLS(TIME,JOBNAME) CMDRESP(TERM)
```

OPSLOG Function Return Codes

The OPSLOG Function command produces these return codes:

40

Indicates a bad call or function.

41

Indicates a numeric conversion error.

48

Indicates a system failure.

Security for the OPSLOG Function

The OPSLOG function requires the same security authorization as OPSLOG Browse (OPSVIEW option 1) when invoked from a TSO or batch address space. When an AOF rule calls the OPSLOG function (either on your system or under the AOF Test Facility), the rule does not check to see if the user has authority to access OPSLOG Browse.

Sample Rules Programs

Sample rules can be found in your opsmvshlq.CCLXRULS and sample OPS/REXX pgms can be found in your opsmvshlq.CCLXSAMP. These rules and programs are supplied to demonstrate utilizing the OPSLOG function. All available sample rules and programs can be found in the Appendixes in the *CA OPS/MVS Event Management and Automation Users Guide*.

A list of sample names, type with descriptions follows.

OPSLGSCN

Obtain and email any OPS/REXX execution errors for AOF rules

Type: PGM

OPSLGEXT

Obtain OPSLOG data for a specific job name and write output to a file

Type: PGM

OPSMTRAP Function

The OPSMTRAP function causes the generation of an SNMP trap by the SNMP subagent component of the product to support the displaying of System State Manager (SSM) resources on a workstation running CA NSM with the CA NSM SSM CA OPS/MVS Option. The cold start trap causes a complete rediscovery of the SSM resources by the workstation. In a large network, a cold start trap can generate a significant amount of network traffic. Therefore, cold start traps should not be sent out on a regular basis. A warm start trap will initiate a verification of the current SSM resources by the workstation. OPSMTRAP can be used to resynchronize the workstation on a regular basis using a TOD rule or on demand when an SNMP communication failure is suspected. An enterprise trap performs a specific action against an SSM monitored table or resource.

The OPSMTRAP function can be used in OPS/REXX, AOF rules, TSO/E REXX, or CLIST.

This function has the following format:

```
var = OPSMTRAP(type,func,table,resource)
```

type

Specifies the type of SNMP trap to be sent. You can specify one of the following values for type:

- C - Cold start trap
- E - Enterprise trap (requires all four arguments to be specified)
- W - (Default) Warm start trap

func

Use the *func* argument with the E value specified for the *type* argument to perform one of the following actions:

- A - Adds a new table or resource
- D - Deletes a table or resource
- S - Sets existing table or resource values (this is the default)

table

Defines the name of an RDF table with a column structure that is known to CA NSM SSM CA OPS/MVS Option. Use the *table* argument with the E value specified for the *type* argument to specify the SSM controlled RDF table that is intended to have the action, specified by the *func* argument, performed against it.

resource

Identifies the resource name of the row in the RDF table (primary key). Use the *resource* argument with the E value specified on the *func* argument and the *table* argument to specify the name of the SSM monitored resource that is intended to have the action, specified by the *func* argument, performed against it.

Example: OPSMTRAP

Issue a warm start trap every night at midnight to ensure that the System State Manager resource data display of the CA Event Manager is synchronized with the current states and values of the SSM resources.

```
)TOD 00:00:00,,,,,CATCHUPYES
)PROC
  rc = OPSMTRAP('W')
  return
)END
```

Note: When a large number of systems are being monitored by a single workstation, it is recommended to have each system generate its warm start trap at a different time to avoid overloading the workstation and the network.

Values Returned

The OPSMTRAP function returns a return code value, which may also be accompanied by an error message. Possible values are:

- 0**
Request successfully processed
- 4**
SNMP subagent is not active

8	Trap request failed
40	Invalid function arguments
69	Too few function arguments
70	Too many function arguments

OPSPDS Function

The OPSPDS function performs a variety of operations on partitioned data sets (PDS) and returns a numeric string value that is used to determine the success of the operation. In some cases it creates REXX stem variables containing additional information.

Note: You can only use the OPSPDS function in OPS/REXX programs. It cannot be used in AOF rules.

This function has the following format:

```
var = OPSPDS(RequestCode,DataSetName|DDName,MemName|StemName)
```

The function arguments describe the type of operation that OPSPDS performs. Possible arguments are:

RequestCode

Specifies the operation to be performed on a PDS. The possible values are:

- EXISTS-Determines if the member name (MemName) exists as a member in the PDS or in a DDName concatenation as defined by the DataSetName or DDName argument.
- DELETE-Deletes the member (MemName) in the PDS specified by DataSetName.
- READDIR-Reads the PDS directory or the first PDS in a DDName allocation and returns the list of all the member names in a series of stem variables.

Note: The request code should be specified as an upper case text string. Mixed case text strings may or may not be supported in the future. There are additional OPSPDS request codes designated for internal CA use only. Do not use any request code other than those documented here.

DataSetName

Defines the valid partitioned data set name (PDS or PDSE). A data set name can be from 1 to 44 characters in length and must meet the IBM naming restrictions for PDS and/or PDSE data set names. For example: SYS1.LINKLIB

DDName

Specifies a DDName for the EXISTS and READDIR request codes. You may use the DDName as an alternative to specifying a partitioned data set name. The DELETE request code will only work with a partitioned data set name.

Note: In the case of READDIR, only members in the partitioned data set at the top of a DDName concatenation are returned.

MemName

Specifies the member name for the EXISTS and DELETE request codes. The OPSPDS function supports 1- to 8-character member names that do not contain periods (.) and do not start with leading numeric digits. While it is possible through programmatic means to create member names that do not meet these restrictions, this is what OPSPDS supports.

StemName

Specifies a stem name for the READDIR function. The stem can be from 1 to 32 characters including an optional trailing period and must conform to the standard REXX rules for stem names. If a period is not provided, one is appended to the end of the stem name value. The stem cannot be a global variable stem of any kind.

Example: Using OPSPDS

These examples show how to verify, delete, and list members.

- The following example verifies that member RULE1 exists in data set OPSMVS.MY.RULES:

```
Qspdsrsc = OPSPDS('EXISTS','OPSMVS.MY.RULES','RULE1')
```

Assuming that OPSMVS.MY.RULES is a valid partitioned data set, the result string is 0 if the member exists or 16 if the member does not exist.

- The following example deletes member REXXPGM1 in data set MY.REXX:

```
Qspdsrsc = OPSPDS('DELETE','MY.REXX','REXXPGM1')
```

Assuming that MY.REXX is a valid partitioned data set, the result string is 0 if the member is successfully deleted or 16 if the member does not exist.

- The following example lists all of the members in data set MY.PDS:

```
Qspdsrsc = OPSPDS('READDIR','MY.PDS','PDSVAR.')
```

Assuming that MY.PDS is a valid partitioned data set, the result string is 0 and variable PDSVAR.0 contains an integer value indicating how many additional stem variables of the type PDSVAR.1, PDSVAR.2, and so on, have been created containing the list of member names.

- The following example demonstrates how to verify the existence of member REXX1 in an uncataloged data set, OPSMVS.MY.REXX, by using OPSDYNAM in conjunction with OPSPDS:

```
FRC = OPSDYNAM('ALLOC DD(MYREXX) DSN('OPSMVS.MY.REXX') SHR VOL(TSU001) UNIT(3390)')
```

```
Qspdsrsc = OPSPDS('EXISTS','MYREXX','REXX1')
```

The uncataloged data set is allocated to the DDName MYREXX. You can then use OPSPDS to determine the existence of member REXX1 within OPSMVS.MY.REXX using the DDName instead of the data set name. Assuming that OPSMVS.MY.REXX is a valid partitioned data set, the result string is 0 if the member exists or 16 if the member does not exist.

Values Returned

The OPSPDS function returns one of the following numeric string values:

0	Request successfully processed
4	Invalid request code
8	Invalid data set name
12	Invalid member or variable name
16	Member/dataset does not exist
20	Data set open failed
24	BLDL failed
28	STOW delete failed
32	Data set close failed
36	Stem variable create failed
40	Error reading data set directory
44	Data set not yet open
48	OPSPDS cannot be used in AOF rules

Variable Output

For the READDIR function OPSPDS creates REXX stem variables using the stem supplied in the third function argument.

- stem.0 contains a numeric integer value indicating how many additional stem variables are created.
- stem.1 through stem.n contain the names of the members in the PDS separated by blanks. You must process all of the blank separated member names in all of the stem variables to process every member of a PDS. If the member list is large, you may need to process multiple REXX variables.

OPSPRM Function

Use the OPSPRM function of OPS/REXX to set or display the values of CA OPS/MVS parameters. The OPSPRM function is used in OPSVIEW option 4.1.1, Set/Display product parameters. Although you can also use the OPSPARM command processor to set or display parameter values, the OPSPRM function processes change or display requests more rapidly, so use OPSPRM if faster performance is important to you. In addition, you can use OPSPRM directly in an AOF rule. The numeric value returned by the OPSPRM function indicates the success or failure of the operation.

Note: You can use the OPSPRM function in OPS/REXX or AOF rules.

The OPSPRM function has two formats.

To display parameters, use this format:

```
var = OPSPRM("SHOW",parmname[,["INFO"][,["NAMES"][,system]])
```

To set parameters or load module information, use this format:

```
var = OPSPRM("SET",parmname,value[,system])
```

When the display format of the OPSPRM function (SHOW keyword) executes successfully, some number of records, containing information about the parameter, are placed on the OPS/REXX external data queue.

OPSPRM can also display information about the load modules that comprise CA OPS/MVS. Enter a module name instead of a parameter name to see the storage location, the assembly date, and other statistics about a module.

Note: When a SHOW function executes, it returns data in the EDQ.

Parmname Argument

The *parmname* argument specifies the name of the CA OPS/MVS parameter (for example, OSFCPU) to be displayed or set. This name can contain no more than fifty characters.

When used to display parameter values (SHOW argument), *parmname* can be any of the following:

- name - Displays the value of the named parameter or the values of all parameters in the named group.
- GROUPS - Displays a list of the parameter groups.
- ALL - Displays all parameter values.

Value Argument

The *value* specifies the new value you are assigning to a parameter. For example, setting *value* to YES changes the value of the parameter to YES. OPS/REXX requires this argument only when you are setting or resetting parameters.

Using OPSPRM to Display and Set Message Severity Codes

While initializing CA OPS/MVS, or after it becomes active, you can use the OPSPRM function to display and set severity codes for CA OPS/MVS messages.

Display a Message Severity Code

To display the severity code *c* for message OPSnnnnn, use this form of the OPSPRM function:

```
T=OPSPRM("SHOW",OPSnnnnn)
```

Change a Message Severity Code

To change the severity code *c* to *d* for message OPSnnnnn, use this form of the OPSPRM function:

```
T=OPSPRM("SET",OPSnnnnn,"d")
```

Important! Before changing the severity code of any CA OPS/MVS message, see the chapter “Understanding CA OPS/MVS Messages” in the *User Guide*.

OPSTATUS Function

The OPSTATUS function returns a count of the address spaces currently in the system or in the *LOGON* state, outstanding WTORs, or IMS systems that are currently in the system and that match the selection criteria. OPSTATUS also optionally writes one or more records to the external data queue.

OPSTATUS is especially useful for retrieving either the names of address spaces that are active or have outstanding replies, or the IMS IDs of active IMS systems.

Note: Using the OPSTATUS function to count the number of batch jobs from a \$HASP message rule may not provide you with accurate results. The message may be issued on behalf of a batch job before the OPSTATUS function counts it or after OPSTATUS counts it. Batch jobs may show up as active before the program is executing, or after the program has finished executing but before JES has finished handling work on its behalf.

Note: The OPSTATUS function can be used in OPS/REXX or AOF rules.

This function has the following format:

```
var = OPSTATUS(func, subfunc, entity[,system])
```

func

Specifies the type of entity whose status is to be queried; it has one of these values:

A

Address spaces

I

IMS systems

J

Batch jobs

P

Attached APPC transactions (ATX)

R

Outstanding replies (WTORs)

S

Started tasks and system address spaces

T

TSO users

subfunc

The *subfunc* argument specifies one of these information types:

A

Based on the function code used, returns only the count of address spaces, outstanding WTORS, or IMS systems that match the selection criteria. No records are added to the external data queue.

L

Returns the count and add a WTOR or IMS record to the external data queue for each WTOR or IMS system found. A subfunction of L is valid only for function values of R (WTORS) and I (IMS systems).

I

Returns the count and do the following:

- For function values of A, J, P, S, or T, add an ASID record to the external data queue for each address space found.
- For a function value of I, add an IMS record (followed by an ASID record) to the external data queue for each IMS system found. The ASID record identifies the IMS system.
- For the function value R, add a WTOR record (followed by an ASID record) to the external data queue for each outstanding WTOR found. The ASID record identifies the issuer of the WTOR.

S

Returns the count and add two records to the external data queue for each outstanding WTOR in the sysplex. The first record contains information about the WTOR and is identical to that returned by the L subfunction. The second record identifies the WTOR issuer on the originating system and indicates how long the WTOR has been outstanding. A subfunction of S is valid only for a function value of R (WTORS).

W

For function values of A, J, P, S, or T, adds an ASID Workload Manager record to the external data queue for each address space that matches the selection criteria.

A subfunction of W is not valid for function values of I (IMS systems) and R (WTORs).

X

Returns the count and adds the following two records to the external data queue for each WTOR on the local system that also originated from the local system:

- The first record is the WTOR itself.
- The second record identifies the WTOR issuer and indicates how long the WTOR has been outstanding.

A subfunction of X is valid only for a function value of R (WTORs).

entity

This argument restricts the returned information to only those entities with names matching the value of entity. Entity can be:

- An asterisk to indicate all entities of a given type.

Note: Be sure to use this when searching for IDs currently in the *LOGON* state.

A string of characters ending with an asterisk to return information about only the entities with names that begins with the specified string.

- A string of characters with no asterisk to return information about only entities with names that exactly match the string.

For all function codes except I, the entity name refers to the jobname/TSO ID/started task name of each address space. For function code I, entity names refer to the four-character IMS ID of each IMS system.

system

(Optional) Enables CA OPS/MVS to execute the OPSTATUS function on external systems that have been defined to the MSF. The MSF establishes VTAM or CCI sessions between copies of CA OPS/MVS, permitting any copy to issue a command on any other copy and to receive its response.

Valid values for *system* are:

- The MSF name or alias of an individual system on which the function is to execute.
- *DEFAULT-Tells CA OPS/MVS to use the default system name set by the ADDRESS OPSCTL MSF DEFAULT HOST command.

If the local MSF system name is used or implied, the system argument is ignored.

The number of seconds that the function waits for a response is determined by the ADDRESS OPSCTL MSF DEFAULT SYSWAIT value when it is greater than zero or the value of the MSFSYSWAIT parameter.

When the system argument is specified for a remote system within a no-wait AOF rule, no output for the function is returned.

An invalid or inactive MSF system name causes the REXX program to be interrupted for a SYNTAX error. Use the ADDRESS OPSCTL MSF LIST HOST command to validate the MSF system name before using it as an argument for any REXX function.

Example: Use the OPSTATUS Function

This example uses OPSTATUS function to determine whether the CICS PROD address space is active on SYSB.

```
jobname = "CICS PROD"
ADDRESS OPSCTL "MSF DEFAULT SYSWAIT(30)"
status = OPSTATUS("A","A",jobname,'SYSB')
if status=1 then TASKSTAT = "UP"
```

ASID Records That OPSTATUS Returns

If the OPSTATUS function specifies a *func* argument of A, J, P, S, or T, and a *subfunc* argument of I, the function generates ASID records with the following format:

Word Number: 1

Length: 8

Contains the Jobname/STCname/TSO ID ('INIT' for idle initiators)

Word Number: 2

Length: 8

Contains the stepname ('NONE' if not available; job class name for idle initiators under JES3 control; TSO logon procedure name for TSO users).

Word Number: 3

Length: 8

Contains the procstepname: 'NONE' if not available; 'IEFPROC' for idle initiators

Word Number: 4

Length: 4

Contains the Address Space Identifier (ASID) (4 hexadecimal digits with leading zeroes).

Word Number: 5

Length: 3

Contains the address space status:

- NSW-Nonswappable
- IN*-Swapping in
- OU*-Swapping out
- IN-Swapped in
- OWT-Swapped out and waiting
- OUT-Swapped out and ready

Word Number: 6

Length: 3

Contains the address space type:

- SYS-System address space
- STC-Started task
- TSU-TSO user
- JOB-Batch job
- ATX-Attached APPC address space
- INI-Initiator

Word Number: 7

Length: 3

Contains the number of Step-Must-Completes.

Word Number: 8

Length: 3

Contains the performance group.

Note: When the system is running in GOAL mode this value will always be zero. Use the W subfunc for WLM information when in GOAL mode.

Word Number: 9

Length: 3

Contains the domain.

Note: When the system is running in GOAL mode, this value is always zero. Use the W subfunc for WLM information when in GOAL mode.

Word Number: 10

Length: 17

Contains the CPU time (in the form SSSSSSSS.MMMMMM, where S = Seconds and M = Microseconds); this is the total CPU time (TCB + SRB) consumed by the current job step only.

Note: The value is 'NONE' if the data is unavailable for any reason.

Word Number: 11

Length: 17

Contains the elapsed Time (in the form SSSSSSSS.MMMMMM, where S = Seconds and M = Microseconds).

Note: The value is 'NONE' if the data is unavailable for any reason.

Word Number: 12

Length: 8

Contains the JES job ID

The value is NONE if the address space associated with the record was not started under JES and has not used the subsystem interface to dynamically acquire a JES job ID, as described below.

Servers and other address spaces may request a job ID later even if they are not started under JES (that is, if the address space was started under the master subsystem-MSTR). In such cases, JES builds the correct data structures, and then creates a job ID for the address space. Job IDs can also be returned. Having a JES job ID at a particular moment means JES allows use of any of its services, such as internal readers for job submission, writers for SYSOUT data sets and allocation, and so on. Without a JES job ID, JES services cannot be used, and errors occur as a result.

This list illustrates the format of data that is returned for each type of job:

- For a started task-STC12345
- For a TSO user-TSU12345
- For a batch job-JOB12345

Note: The JES job ID of an idle initiator is always in the STCnnnnn format.

Word Number: 13

Length: 17

Contains the CPU Time in the format of SSSSSSSS.MMMMMM.

S indicates seconds and M indicates microseconds.

This is the total task time (TCB) used by all jobs in the address space (the same value returned by the z/OS DISPLAY command)

There is no direct relationship between the value of word ten and the value of word thirteen. They are two different measures of CPU time. For started tasks, TSO users, and single-step batch jobs, the two values may or may not be fairly close. For multi-step jobs that are in the second or subsequent step, the values probably differ greatly.

Note: The value is NONE if the data is unavailable for any reason.

Word Number: 14

Length: 1

Contains a value indicating whether the address space is associated with USS; O if yes and N if no.

Word Number: 15

Length: 10

Contains the total number of auxiliary storage (paging space) slots used by the address space. Each slot is 4096 bytes, or 4 KB.

Word Number: 16

Length: 10

Contains the total number of real storage frames that are in the private region used by the address space. Each frame is 4096 bytes, or 4 KB.

Word Number: 17

Length: 10

Contains the total number of expanded storage pages used by the address space. Each page is 4096 bytes, or 4 KB.

Note: On z/OS systems running in 64-bit mode, expanded storage is not supported and this value will always contain zeros.

Word Number: 18

Length: 8

Contains the USERID associated with the address space. When no USERID is available, this field contains the value NONE.

Note: In the case of a multi-user address space, the USERID reflects the USERID that was initially associated with the address space/job.

Word Number: 19

Length: 10

Contains the number of bytes of CSA associated with the address space. Contains the value NONE if Common Storage Tracking is inactive.

Word Number: 20

Length: 10

Contains the number of bytes of ECSA associated with the address space. Contains the value NONE if Common Storage Tracking is inactive.

Word Number: 21

Length: 10

Contains the number of bytes of SQA associated with the address space. Contains the value NONE if Common Storage Tracking is inactive.

Word Number: 22

Length: 10

Contains the number of bytes of ESQA associated with the address space. Contains the value NONE if Common Storage Tracking is inactive.

Word Number: 23

Length: 10

Contains the CPU Time in the format of *SSSSSSSS.MMMMMM*.

S indicates seconds and *M* indicates microseconds.

This is the total CPU time (TCB + SRB) used by all enclaves owned by the current address space with the current jobname.

An enclave is a transaction that can span multiple dispatchable units (SRBs and tasks) in one or more address spaces and is reported on and managed as a unit. It is managed separately from the address space it runs in. CPU and I/O resources associated with processing the transaction are managed by the transaction's performance goal and reported to the transaction.

The value is NONE if the data is unavailable for any reason.

Notice that the time format given for CPU Time and Elapsed Time deviates from that displayed in OPSVIEW option 3.1. Since OPSTATUS is intended for automation rather than for producing displayable records, the format shown here reduces the need to convert time data for use in automation. Times returned by OPSTATUS for the use of OPS/REXX still require separation into seconds and microseconds.

The following OPS/REXX statements separate seconds and microseconds:

```
CPUTIME = OPSTATUS('A','I',VTAM)
FULL CPUTIME
SECONDS = WORD(TRANSLATE(CPUTIME,' '),10)
MICROSECS = WORD(TRANSLATE(CPUTIME,' '),11)
```

IMS Records That OPSTATUS Returns

If the OPSTATUS function specifies a *func* argument of I and a *subfunc* argument of I or L, the function generates IMS records with the following format:

Word Number: 1

Length: 4

Contains the IMS system identifier.

Word Number: 2

Length: 1

Contains the IMS subsystem character.

WTOR Records That OPSTATUS Returns

If the OPSTATUS function specifies a *func* argument of R and a *subfunc* argument of I, L, S, or X the function generates WTOR records with the following format:

Word Number: 1

Length: 2 or 4

Contains the outstanding reply number

Word Number: 2

Length: 126

Contains WTOR message text (starts with the three-digit reply ID, followed by a space)

If the OPSTATUS function specifies a function argument of R and a subfunction argument of S or X, the function generates a second record for each WTOR containing sysplex wide WTOR records with the following format:

Word Number: 1

Length: 8

Originating address space name.

Word Number: 2

Length: 8

JES job ID associated with the issuing address space.

This list illustrates the format of data that is returned for each type of job:

- For a started task-STC12345
- For a TSO user-TSU12345
- For a batch job-JOB12345

Note: The JES job ID of an idle initiator is always in the STCnnnnn format.

Word Number: 3

Length: 4

Address Space Identifier (ASID) (4 hexadecimal digits with leading zeroes) on the originating system.

Word Number: 4

Length: 8

System name on which the WTOR was issued.

Word Number: 5

Length: 8

Security user ID of the WTOR issuer.

Word Number: 6

Length: 10

Elapsed time in seconds that the WTOR has been outstanding.

ASID Workload Manager Records That OPSTATUS Returns

When you use *subfunc W*, the ASID record that is returned for each address space contains Workload Manager (WLM) information for each ASID, JOB, STC, or TSO address space that matches the selection criteria. The format is as follows:

Word Number: 1

Length: 8

Contains the address space name or MASTER*

Word Number: 2

Length: 8

Stepname or NONE

Word Number: 3

Length: 8

Contains the procstepname or NONE

Word Number: 4

Length: 4

Contains the ASID or NONE

Word Number: 5

Length: 8

Contains ccccccc, which can be any number of alphanumeric characters as defined by IBM for Service Class Names. If this string contains /, it is either an error condition or WLM is in COMPAT mode.

Word Number: 6

Length: 3

Contains the WLM service class period number or 0

Word Number: 7

Length: 8

Contains the workload name, or N/A if either of the following is true:

- No workload name is defined
- The system is in COMPAT mode

Word Number: 8

Length: 8

Contains the resource group name, or N/A if either of the following is true:

- No resourcegroup name is defined
- The system is in COMPAT mode

Word Number: 9

Length: 1

Contains the following:

- Y - If the address space is designated as a server address space to WLM
- N - If the address space is not designated as a server address space or the system is in COMPAT mode

Word Number: 10

Length: 1

Contains the following:

- Y - If the address space was reset through the RESET XXX,QUIESCE command
- N - If the QUIESCE delay indicator is not set or the system is in COMPAT mode

WLM Returns and Descriptions

The following list describes the WLM return codes.

cccccccc

WLM returned a Service Class Name.

N/A

WLM is in COMPAT mode or WLM is not available on this system.

WLM/ERR

There is an error in communicating with WLM. Reissue the command after 2-3 seconds to try to obtain results.

SS_N/A

The CA OPS/MVS subsystem is not available, or internal communications are not possible.

Sample Output from OPSTATUS

These REXX statements produce output similar to that shown in the following example:

```
say 'Count of address spaces:' OPSTATUS('A','I','*')
do while queued() > 0
  pull x
  say x
end
```

Output:

```
Count of address spaces: 12
*MASTER* NONE NONE 0001 NSW SYS ....(more columns here)...
PCAUTH PCAUTH NONE 0002 NSW SYS .....
TRACE TRACE NONE 0003 NSW SYS .....
GRS GRS NONE 0004 NSW SYS .....
DUMPSRV DUMPSRV DUMPSRV 0005 OWT SYS .....
CONSOLE CONSOLE NONE 0006 NSW SYS .....
ALLOCAS ALLOCAS NONE 0006 NSW SYS .....
SMF SMF IEPROC 0007 NSW SYS .....
ACF2 ACF2 IEPROC 0008 NSW STC .....
NT STEP2 NONE 0011 OWT IN .....
BATJOB1 STEP2 NONE 0012 NSW JOB .....
TSOD1 NONE NONE 0009 OWT TSU .....
```

Examples: OPSTATUS

Reviewing the following examples gives you a better understanding of OPSTATUS:

- Example 1: The following example shows you how to use the OPSTATUS function to determine whether the CICSPROD address space is active:

```
jobname = "CICSPROD"
status = OPSTATUS("A","A",jobname)
if status=1 then TASKSTAT = "UP"
```

- Example 2: The following example illustrates using OPSTATUS to determine whether a TSO user (whose user ID is TSOUSER) is logged on:

```
userid = "TSOUSER" /* Replace TSOUSER with a valid TSO user ID */
if OPSTATUS("T","A",userid) < 1 then
  say userid "is not logged on at this time."
```

- Example 3: The following example shows you how to use OPSTATUS to extract information about outstanding WTORS and the address spaces that issued them:

```
say "Count of Outstanding Replies" OPSTATUS("R","I","*")
do while QUEUED() > 0
  pull line
  say line
end
```

- Example 4: The following example shows you how to use OPSTATUS to extract information about outstanding WTORS that were issued on this system and that have been outstanding for more than five minutes:

```
ThisSys = OPSINFO("SYSNAME")
temp = OPSTATUS("R","S","*")
do while QUEUED() > 0
  parse pull ReplyID Text
  parse pull jbnm jbid asid syna uid secs .
  if (syna = ThisSys) & (secs > 300) then
    say jbnm jbid asid uid secs ReplyID Text
  end
```

OPSTORE Function

The OPSTORE function returns a character string of the specified *length* containing a copy of the virtual storage at the specified *address*.

At present, the only *func* value supported is S for Storage.

Note: The OPSTORE function can be used in OPS/REXX or AOF rules.

This function has the following format:

```
var = OPSTORE(func,address,length)
```

address

Address may be any virtual storage address in the home address space. When a REXX program runs in an APF-authorized environment, all virtual storage locations in the home address space can be accessed. Under usual circumstances, only production (not test) AOF rules run as APF-authorized. However, CA OPS/MVS TSO command processors that have been added to the TSO authorization tables are an exception to this rule. When a REXX program running in a non-APF-authorized environment invokes the OPSTORE function to access fetch-protected storage, a zero-length result is returned.

length

The *length* argument can be any non-negative integer value up to 256 (inclusive). This specifies the length of the virtual storage data to be returned in the result character string.

Examples: OPSTORE

The following are examples of the OPSTORE function:

- Example 1: To display the address of the CVT, issue the OPSTORE function as follows:

```
CVT=OPSTORE(S,'10X',4)
SAY C2X(CVT)
```

- Example 2: The following REXX EXEC can be executed with OPS/REXX or TSO/E REXX; it shows the differences between the OPS/REXX OPSTORE function and the TSO/E REXX STORAGE function:

```
/* REXX */

parse source rextype .
if rextype = "OPS/REXX" then
do
  CVT   = OPSTORE("S",10x,4)
  asvt  = OPSTORE("S",D2C(C2D(CVT) + 556),4)
  asvtnor = C2D(OPSTORE("S",D2C(C2D(asvt) + 488),4))
  asvtanr = C2D(OPSTORE("S",D2C(C2D(asvt) + 496),4))
end
else
do
  CVT   = C2D(STORAGE(10,4))
  asvt  = C2D(STORAGE(D2X(CVT + 556),4))
  asvtnor = C2D(STORAGE(D2X(asVT + 488),4))
  asvtanr = C2D(STORAGE(D2X(asVT + 496),4))
end

say "asvtnor" = "asvtnor"
say "asvtanr" = "asvtanr"

result = asvtanr - asvtnor
```

OPSUBMIT Function

The OPSUBMIT function lets you submit a batch job directly from an OPS/REXX program. The JCL that is submitted can be in either the external data queue or a set of OPS/REXX stem variables.

Note: The OPSUBMIT function can be used only in OPS/REXX.

This function has the following two formats:

- Format 1: This format submits the data from the external data queue.
lines = OPSUBMIT()
- Format 2: This format submits the data from the OPS/REXX compound symbols in the form *stem.1*, *stem.2*, *stem.3*, and so on until the next variable is found.

lines = OPSUBMIT(*stem*.)

Example: OPSUBMIT

This example illustrates using the OPSUBMIT function to submit the same rule twice, first from stem variables, and then from the external data queue:

```
qqq.1 = "//TESTJOB    JOB REGION=1M"
qqq.2 = "//          MSGCLASS=X,NOTIFY=USERID1"
qqq.3 = "//#"
qqq.4 = "// EXEC  PGM=HEFBR14"
temp = OPSUBMIT(qqq)
say temp "lines submitted from stem variables"
do i = 1 to 4
  queue qqq.i
end
temp = OPSUBMIT()
```

Usage Notes for OPSUBMIT

Keep these notes in mind when using OPSUBMIT:

- All lines/variables are either truncated or extended to a length of 80 bytes.
- OPSUBMIT returns a value that reflects the number of JCL lines that were submitted.
- The OPSUBMIT function cannot be used in any AOF rule. If used in an AOF rule, the function is treated as a null operation and it returns a null string.
- When used in the SUB-MSTR OSF environment, the OSFGETJOBID parameter must be set to YES to use OPSUBMIT.

OPSUSS Function

The OPSUSS function displays information about UNIX System Services (USS) processes, the ID of a user and group, and debug status of the current task in REXX variables. If the caller is properly authorized by the security system, the effective user or group ID of the caller may also be changed dynamically. USS security systems do not usually allow users to display process information for tasks that do not belong to them. A temporary switch to user ID 0 may be required to obtain the desired process data. Process information may be selected by PID (process number), UID (user number), GID (group number), job name, command name, or path name to limit the amount of output produced.

Note: This function may also be used in a TSO/E REXX program. This function can only be used for TOD and REQ rules.

This function has the following format:

```
var = OPSUSS(function,subfunction,entity,{keyword arguments})
```

Values Returned

The OPSUSS function returns a count of matching processes, the ID of the user, or the ID of the group for the PROCESS, USER, and GROUP function codes respectively. For the INFO and SET functions, 1 is returned if the function succeeded and 0 if the function failed. Messages for serious errors encountered by the underlying USS callable service calls are also produced as appropriate and result in termination of the REXX program.

Variable Output

Output variables produced by OPSUSS vary in format depending on the variable prefix value specified. The default prefix is USSDA_. When the prefix ends with a period, it is used as is to form a true REXX stem variable that is easy to drop. When the prefix does not end with a period, an underscore is appended to the prefix to form multiple REXX stem variables.

Prefix	Resulting Variable Name
USSDA_	USSDA_PID.1
USSDA	USSDA_PID.1

Output variables vary by the function specified and whether the details keyword is specified. The following output variables are produced by function code.

Process Function Variables

The following are Basic variables:

USSDA_PID.n

The unique numeric process ID of this USS process.

USSDA_PPID.n

The parent process ID of this USS process.

USSDA_PROCSTATUS.n

The one-byte hexadecimal status flag of the process. X'80' means the process is stopped. For all possible flag bit meanings, see IBM macro BPXYPGPS.

USSDA_TASKSTATUS.n

Single alphabetic character code value of the state of the USS process. For code meanings or the description of the USS ps command, see IBM macro BPXYPGPS.

USSDA_SID.n

The numeric session ID of this USS process.

USSDA_PGPID.n

The numeric group ID of this USS process.

USSDA_FGPID.n

The foreground numeric group ID of this USS process.

USSDA_EUID.n

The effective numeric user ID of this USS process.

USSDA_RUID.n

The real numeric user ID of this USS process.

USSDA_EGID.n

The effective numeric group ID of this USS process.

USSDA_RGID.n

The real numeric group ID of this USS process.

USSDA_SERVERNAME.n

The server name of this process if it registered with USS as a file server.

USSDA_SYSCALLS.n

The number of USS system service calls made by this process.

USSDA_STARTDATE.n

The date this USS process was created in YYYYMMDD format.

USSDA_STARTTIME.n

The time this USS process was created in HH:MM:SS format.

USSDA_SYSTEMCPU.n

The amount of system overhead CPU time in seconds (ssss.hh) consumed by this USS process.

USSDA_USERCPU.n

The amount of user CPU time in seconds (ssss.hh) consumed by this USS process.

USSDA_CONTTY.n

The controlling terminal name for this USS process, if any.

USSDA_PATH.n

The path name or z/OS program name of the program associated with this USS process.

USSDA_COMMAND.n

The USS command name and its arguments or the z/OS program name associated with this USS process.

USSDA_JOBNAME.n

The z/OS job name of the ASID in which this USS process is running.

USSDA_ASID.n

The address space identifier in which this USS process is running in hexadecimal character format.

The following are Details Keyword variables:

USSDA_SUID.n

The saved numeric user ID for this USS process.

USSDA_SGID.n

The saved numeric group ID for this process.

USSDA_TOTSIZE.n

The amount of virtual storage in bytes that this USS process is using.

USSDA_MVSSWAP.n

The z/OS swap status of this USS process. Values are IN or OUT.

USSDA_SERVERTYPE.n

An integer indicating the virtual file system server type. Values are:

0-Not a server

1-File server

2-Lock server

3-File exporter

USSDA_SERVERFLAGS.n

The four-byte binary value of the server flag field mapped in macro BPXYNREG.

USSDA_MAXVNODES.n

The maximum number of active file server tokens allowed for the server.

USSDA_USEDVNODES.n

The current number of active file server tokens in use by the server.

USSDA_KERNELWAIT.n

The number of elapsed seconds (ssss.hh) since this USS process entered a kernel wait state.

User Function Variables

The following are Basic variables:

USSDA_USERNAME.n

The security system user name corresponding to the user ID number.

USSDA_USERID.n

The USS user ID number corresponding to the user name.

USSDA_GROUPID.n

The USS group ID number for this user ID.

The following are Details Keyword variables:

USSDA_PATH.n

The path name of the initial working directory at logon for this user.

USSDA_PROGRAM.n

The name of the initial program to execute at logon for this user.

Group Function Variables

The following are Basic variables:

USSDA_GROUPNAME.n

The name of the USS group corresponding to the group ID number.

USSDA_GROUPID.n

The USS group ID number corresponding to group name.

USSDA_MEMCOUNT.n

The number of USS user names belonging to this group.

The following are Details variables:

USSDA_MEMLIST.n

The string of USS user names separated by blanks that belong to this group.

Info Function Variables

The following are Basic variables:

USSDA_DUBSTATUS.n

The USS dub status of the current task. Possible values are:

- DUBBED_FIRST-Task is dubbed and was caused by this program.
- DUBBED-Task is dubbed and was caused by another task.
- DUB_MAY_FAIL-Task is not dubbed and any dub attempt may fail due to an incomplete user security profile.
- DUB_OKAY-Task is not dubbed and any dub attempt would most likely succeed.
- DUB_AS_PROCESS-The task has not been dubbed but the address space has been dubbed. If this task is dubbed it will become a new process in this address space.
- DUB_AS_THREAD-The task has not been dubbed but the address space has been dubbed. If this task is dubbed, it will become a new thread in the address space process.

No keyword variables are defined.

Function Argument

The function argument determines the type of USS information that is to be returned or processed by the subfunction argument.

The valid functions are:

PROCESS

Retrieve matching USS process (task) information for the subfunction selection criteria.

USER

Retrieve user information for a particular USS user ID or user name.

GROUP

Retrieve group information for a particular USS group ID or group name.

INFO

Retrieve the USS dub status for the current task.

SET

Change the effective user ID or group ID of the current process to the value specified if allowed by the security system.

Subfunction Argument

The subfunction argument designates the type of information contained in the entity argument or the explicit data to be retrieved depending on the function specified. The following table shows the valid subfunction values and the functions with which they may be used:

Subfunction	Description	Used with...
PID	The USS process ID number	PROCESS
PPID	The USS parent process ID number	PROCESS
UID	The real USS user ID number	PROCESS, USER
EUID	The effective user ID number	PROCESS, SET
GID	The real group ID number	PROCESS, GROUP
EGID	The effective group ID number	PROCESS, SET
ALLDATA	All USS processes permitted	PROCESS
PATH	The path name or z/OS program name associated with a process	PROCESS

Subfunction	Description	Used with...
COMMAND	The command invocation string or z/OS program name that started a process	PROCESS
JOBNAME	The z/OS job name of the address space in which the process is running	PROCESS
NAME	The USS user or group name	USER, GROUP
DUB	The USS dub status of the current task	INFO

Entity Argument

The entity must be a character string or integer number corresponding to the subfunction specified. The DUB and ALLDATA subfunction should have a null entity argument. The COMMAND, PATH, and JOBNAME subfunctions may be prefixed or suffixed with an '*' and may also contain the wildcard matching character '%' to match any character of the subfunction value retrieved. As a prefix, '*' specifies 'the value must end with' the entity value specified. As a suffix, '*' specifies 'the value must begin with' the entity value specified. '*' and '%' may be used alone or in combination.

For example, if OPSUSS('PROCESS','PATH','*/bin/%h') found a process whose path value was '/user/bin/sh' it would be selected as matching the subfunction entity whereas '/sbin/sh' would not be selected.

Keyword Argument

The keyword argument contains a string of optional keywords that affect the quantity and names of the output variables. The valid keywords are:

DETAILS

If specified, this keyword causes the complete set of output variables to be created for each retrieved function element that is selected. If omitted, only the basic output variables are created.

PREFIX (output variable prefix)

A REXX stem name or character string that is appended to the beginning of the output variable names. The default is USSDA_.

Example: OPSUSS

The following example uses OPSUSS to get the pid of CAIOPR (CA Event Manager).

```

procpid=0
procstat=DOWN'
if OPSINFO('OMVS') = 'READY' then do
  pid=OPSINFO('USSPID')
  /* If not dubbed yet, change to superuser mode */
  if pid = 'NA' then
    uid=OPSUSS('SET';EUID,0)
  /* Find caiopr processes */
  status=OPSUSS('PROCESS';'PATH;'/tngfw/opr/bin/caiopr')
  select
  when status = 1 then
    do
      procstat=UP
      procpid=ussda_pid.1
    end
  /* Handle multiple occurrences of the same path but with one */
  /* parent pid (caiopr) */
  when status > 1 then
    do
      pgid=ussda_pgid.1
      do ix = 2 to status
        if ussda_pgid.ix <> pgid then
          leave
        end
      if ix > status then
        do
          procpid=pgid
          procstat=UP
        end
      end
    end
  otherwise
    nop
  end
end
end

```

OPSVALUE Function

The OPSVALUE function lets you manipulate compound symbols in ways that are not possible in standard OPS/REXX. For example, the OPSVALUE function lets you use compound symbols, especially global compound symbols, as a kind of database.

Note: The OPSVALUE function can be used in OPS/REXX or AOF rules.

This function has the following format:

```
var = OPSVALUE(derivedname[[actioncode]][[newvalue]]oldvalue]])
```

derivedname

Defines the name of the symbol to be acted on. When you use this argument without quotation marks, simple symbols (which are case sensitive) following the stem are replaced by their values. Although the derived name may be as long as 84 bytes, global variable rules can process only the first 50 bytes of the name. Therefore, you may want to limit the length of global variable names to 50 bytes.

actioncode

(Optional) Specifies the action to be taken on that symbol.

newvalue

(Optional) Supplies the new value, if any, to assign to the symbol, and

oldvalue

(Optional) Fetches the value of the symbol before the *actioncode* action takes place.

Limits for Global Variable Stems

You should not create too many global variables under a single global variable stem. If you do, you will no longer be able to view the global variables under OPSVIEW option 4.8, or access them using the OPSVALUE function.

The absolute product limit is 32,768 variables under a single global variable stem. However, in practice, we strongly recommend that no more than 10,000 global variables exist at any given instant under a single global variable stem.

Note: There is a restriction in viewing too many variables when going cross-system. Trying to access several hundred variables may cause a time-out on the MSF link.

OPSVALUE and Cross-system Operations

To use the OPSVALUE function for cross-system operations, change the default system name by using the following command:

```
ADDRESS OPSTCL "MSF DEFAULT SYSTEM(sysname)"
```

Any subsequent OPSVALUE function is routed to the *sysname* system. You must specify an individual system name on the ADDRESS OPSTCL MSF DEFAULT host command that is associated with the OPSVALUE function. Values of ALL and EXT are invalid when used with OPSVALUE.

To return to the local system, use this command:

```
ADDRESS OPSTCL "MSF DEFAULT SYSTEM(*)"
```

For more information about the ADDRESS OPSTCL MSF DEFAULT command, see the chapter "Host Environment Commands."

Actions OPSVALUE Takes

OPSVALUE returns a value from the function call, and, in the case of some action codes, also places information in the external data queue.

You can specify the following *actioncode* values. If you omit the *actioncode*, then OPS/REXX uses the code V by default.

- 0 (Zero)
 - Obtains the value of a global variable. If the global variable does not exist, then OPS/REXX returns a zero.
 - Does not change the external data queue.

Note: If you use this function, you will not be able to differentiate between a global variable that has zero as its value and a global variable that does not exist.

Example:

```
RTVL = OPSVALUE(derivedname,0)
```

■ 6 (Delete single variable)

- Removes the node specified by the derived name without removing any of its subnodes
- Returns 1 if the node was deleted; returns 0 if the node was not found
- Does not change the external data queue
- Does not allow other accessors of compound symbols to see partially updated symbol names

Example:

```
RTVL = OPSVALUE(derivedname,6')
```

■ A (Add)

- Adds a number specified by *increment* to the existing compound symbol given by *derivedname*.
- Returns the sum of the compound symbol and the increment.
- Does not change the external data queue.
- All references to the compound symbol are serialized during the ADD operation. That is, you can use this function safely to increment a counter that is set by concurrent tasks.

Example:

```
RTVL = OPSVALUE(derivedname, 'A', increment)
```

■ C (Compare and update)

- Updates a compound symbol after verifying its current value. If the node does not exist, then OPS/REXX creates it but assigns it no value (giving the symbol the same value as its name).
- Safely updates Global Compound Symbols shared by more than one rule or Global Compound Symbols that multiple copies of the same rule may access and update.
- Does not change the OPS/REXX external data queue.
- Returns the REXX true value (1), if the comparison found the pre-action value of the symbol to be equal to *old value* and the compound symbol was updated, or the REXX false value (0), if the comparison found unequal values and therefore did not update the value of the compound symbol.
- Serializes the compare and update operations for global variables.

Example: To perform the Compare and Update action, four rather than three operands must be used with the 'C' code. The syntax is as follows:

```
RTVL = OPSVALUE(derivedname, 'C', newvalue, oldvalue)
```


Usage Example: The following example demonstrates how to safely initialize a counter in the)PROC section of a rule using the Compare and Update action code. After possible initialization, the counter is incremented using the Add action code. By using serialized OPSVALUE functions to initialize and increment the counter, this example safely allows for the possibility that multiple tasks running in the same address space may issue message IEF13241 concurrently.

```
)MSG IEF13241
)PROC
/* Load name of OPS job variable used to hold message */
count per job. */
counter='GLVJOBID.IEF13241'
/*
*/
/* Initialize the counter if it does not exist (i.e. */
/* this is the first IEF13241 message to be issued by */
/* this job since this rule was enabled.) To allow for */
/* the possibility that another task in this same job */
/* has simultaneously issued another "first" IEF13241 */
/* message and is executing this same rule concurrent */
/* to this task, use the serialized OPSVALUE Compare and */
/* Update function: */
Temp = OPSVALUE(counter,'C',0,counter)
/* We do not care about the success of the above Compare */
/* and Update. It will succeed only the first time that */
/* it is executed by this job. In that case, since the */
/* variable does not yet exist, OPSVALUE will create the */
/* variable and give it a value of its own name, causing */
/* the Compare to succeed and the value to be Updated */
/* to 0. The Compare will fail for all subsequent */
/* executions by this same job, since the variable will */
/* already exist, and its value will not be equal to its */
/* own name. In that case, the variable may have been */
/* initialized days ago by a task that is no longer */
/* attached, or it may have been initialized a */
/* microsecond ago by a task that is executing this same */
/* rule concurrent to this task. It does not matter. */
/* The counter is initialized only once per job. */
/*
*/
/* Increment the counter by 1. To allow for the */
/* possibility that another task in this same job has */
/* simultaneously issued another IEF13241 message and is */
/* executing this same rule concurrent to this task, use */
/* the serialized OPSVALUE Add function: */
temp = OPSVALUE(counter,'A',1)
RETURN
```

■ D (Drop)

- Performs the OPS/REXX DROP operation on the compound symbol specified by *derivedname*. The compound symbol is reset to its uninitialized value; that is, its derived name. If *derivedname* is the name of a stem, then all compound symbols belonging to that stem are not only dropped, but also rendered nonexistent and the virtual storage allocated to them is released.
- Returns the value of *derivedname*.
- Does not change the external data queue.
- All other references to the *derivedname* will either see the compound symbol as it existed before the DROP operation began, or as it is after the DROP operation completes.

Example:

```
RTVL = OPSVALUE(derivedname, 'D')
```

■ E (Existence)

- Checks to see whether a given global variable exists
- Does not change the OPS/REXX external data queue
- Returns the status of a given global variable as one of these characters:

■ I-Initialized

■ U-Uninitialized

■ N-does Not exist

Example:

```
RTVL = OPSVALUE(derivedname, 'E')
```

Note: For most types of variables, N and U have interchangeable meanings. However, for global variables, N means that no storage exists for a variable; and U means that the variable exists in storage, but is uninitialized and is set to the value of its name.

■ F (Find)

- Checks to see if a given global variable exists. The F action is more efficient and more reliable than using the E and O functions together.

- Returns the status of a given global variable as one of these characters:

I-Initialized

U-Uninitialized

N-does Not exist

When the returned value is not N (meaning that the derived name exists), the value of the node is returned on the external data queue. The maximum length of a string pulled from the external data queue is 350 bytes. CA OPS/MVS truncates longer values.

Example:

```
RTVL = OPSVALUE(derivedname, 'F')
```

■ H (High-level security)

- Establishes an authorization status for subsequent global variable and SQL access requests made by an OPS/REXX program. This provides more efficient access to global variables and SQL tables for an OPS/REXX program.

You can use this function from any environment that supports OPSVALUE, such as REXX programs, rules, GEM, and OPSLINK; however, for this function to be useful for OPSVALUE or ADDRESS SQL, you must use it in an OPS/REXX program.

For OPSVALUE requests, you must specify one of the following values for the *derivedname* variable:

GLOBAL.READONLY

GLOBAL.READWRITE

For ADDRESS SQL requests, you must specify one of the following values for the *derivedname* variable:

GLOBAL.SQLREADONLY

GLOBAL.SQLREADWRITE

- Returns a value describing the authorization status of the OPS/REXX program:

AUTH-The request is permitted.

NOTAUTH-The request is denied (no error message is issued).

When the AUTH value is returned, subsequent OPSVALUE or SQL calls from the OPS/REXX program will not create the type of OPSGLOBAL or SQL security events that correspond to the authority level that was obtained on the preceding high-level security call. For example:

- If you specify GLOBAL.READWRITE on the function, and the returned value is AUTH, no subsequent OPSVALUE calls will create security events.

- If you specify GLOBAL.READONLY on the function, and the returned value is AUTH, no subsequent OPSVALUE access-type calls will create security events.
- If you specify GLOBAL.SQLREADWRITE on the function, and the returned value is AUTH, no subsequent ADDRESS SQL operations will create security events.
- If you specify GLOBAL.SQLREADONLY on the function, and the returned value is AUTH, no subsequent ADDRESS SQL access-type operations will create security events.
 - Results in an OPSGLOBAL product security event (SEC rule). The SEC.AUGLOPCH is set to H, and the SEC.AUGLRQTY variable is set to one of the following:
 - A (Access)-If you specify GLOBAL.READONLY or GLOBAL.SQLREADONLY
 - U (Update)-If you specify GLOBAL.READWRITE or GLOBAL.SQLREADWRITE

Examples:

```
RTVL = OPSVALUE('GLOBAL.READONLY','H')
RTVL = OPSVALUE('GLOBAL.READWRITE','H')
RTVL = OPSVALUE('GLOBAL.SQLREADONLY','H')
RTVL = OPSVALUE('GLOBAL.SQLREADWRITE','H')
```

- I (Information)
 - Returns information about all of the immediate subnodes of the *derivedname* to the external data queue

The *derivedname* value must be a compound symbol node. The return value is the number of immediate subnodes that exist. The external data queue contains two lines per subnode: the first line contains the next segment of the derived name, and the second line contains statistics about the derived name. The second line returned for each derived name contains the following information. The first piece of information indicates the word number, the second indicates the length of the word, and the third describes the word.
- 1, 8-The number of subnodes under this subnode
- 2, 10-Creation date (in the form yyyy/mm/dd)
- 3, 8-Creation time (in the form hh:mm:ss)
- 4, 17-Creation ruleset.rule or program name
- 5, 8-Creation Jobname/Taskname/TSO ID
- 6, 10-Last modification date (in the form yyyy/mm/dd)
- 7, 8-Last modification time (in the form hh:mm:ss)
- 8, 17-Last modification ruleset.rule or program name
- 9, 8-Last modification Jobname/Taskname/TSO ID
- 10, 1-This word, which is reserved, always contains the value 0; provides compatibility with programs expecting a numeric value

- 11, 8-Number of updates to this node
- 12,10-Last access date (in the form yyyy/mm/dd) or the string NONE if the variable was created under an older release of the product; permits users to determine when global variables have not been used in a long time and thus may be eligible for deletion:
 - Returns the number of subnodes listed in the external data queue.
 - Places two lines per subnode in the external data queue
 - Returns no partially updated symbol names

Example:

```
RTVL = OPSVALUE(derivedname, 'I')
```

- J (Immediate subtree count)

Returns a count of all the immediate subnodes of *derivedname*.

This action returns the same result value as the I or L action codes. However, the external data queue is not modified.

Example:

```
RTVL = OPSVALUE(derivedname, 'J')
```

- K (subtree count)

Returns a count of all the subnodes of the *derivedname*

The result value returned by this action is the same value returned by the S or T action code. However, the external data queue is not modified.

Example:

```
RTVL = OPSVALUE(derivedname, 'K')
```

- L (List)

- Lists the derived names of all the immediate subnodes of *derivedname* by placing them on the external data queue

The results of this action illustrate the difference between dropped symbols (processed by action D) and removed symbols (processed by action R). Dropped symbols still exist, so the List action can find them. The List action does not return removed symbols.

- Returns the number of subnodes listed in the external data queue
- Places a list of subnodes of the specified nodes in the external data queue
- Returns no partially updated symbol names

Example:

```
RTVL = OPSVALUE(derivedname, 'L')
```

- N (Null)

- Obtains the value of a global variable. If the global variable does not exist, then OPS/REXX returns a null string.
- Does not change the external data queue.

Note: If you use this function, then you will not be able to differentiate between a global variable that has a null string as its value and a global variable that does not exist.

Example:

```
RTVL = OPSVALUE(derivedname, 'N')
```

- O (Obtain)

- Obtains the value of a global variable. If the global variable does not exist, OPS/REXX returns an error
- Does not change the external data queue

Example:

```
RTVL = OPSVALUE(derivedname, 'O')
```

- R (Remove)

- Removes the node specified by *derivedname* and all of its subnodes. Once a node is removed, it ceases to exist.
- Returns the number of subnodes removed.
- Does not change the external data queue.
- Does not allow other accessors of compound symbols to see partially updated symbols.

Example:

```
RTVL = OPSVALUE(derivedname, 'R')
```

- S (Subtree)

- Lists the derived names of all the subnodes of *derivedname* in the external data queue. Action code S is similar to code L with two differences:

- OPS/REXX places the entire global variable name in the external data queue.

- All subnodes of the derived name are listed.

- Returns the number of subnodes listed in the external data queue.
- Places the entire global variable name in the external data queue.
- Returns no partially updated symbol names.

Example:

```
RTVL = OPSVALUE(derivedname, 'S')
```

- T (subTree/info)

- Returns to the external data queue information on all the subnodes of the *derivedname*

The *derivedname* value parameter must be a compound symbol node. The return value is the number of subnodes that exist. The external data queue contains two lines per subnode: the first line contains the next segment of the derived name, and the second line contains statistics about the derived name. The second line contains the following information-the first piece of information indicates the word number, the second indicates the length of the word, and the third describes the word.

- 1, 8-The number of subnodes under this subnode (always contains a zero)
- 2, 10-Creation date (in the form yyyy/mm/dd)
- 3, 8-Creation time (in the form hh:mm:ss)
- 4, 17-Creation ruleset.rule or program name
- 5, 8-Creation Jobname/Taskname/TSO ID
- 6, 10-Last modification date (in the form yyyy/mm/dd)
- 7, 8-Last modification time (in the form hh:mm:ss)
- 8, 17-Last modification ruleset.rule or program name
- 9, 8-Last modification Jobname/Taskname/TSO ID
- 10, 8-This word, which is reserved, always contains the value 0; provides compatibility with programs expecting a numeric value
- 11, 8-Number of updates to this node
- 12, 10-Last access date (in the form yyyy/mm/dd) or the string NONE if the variable was created under an older release of the product; permits users to determine when global variables have not been used in a long time and thus may be eligible for deletion.

Action code T resembles code I with three differences:

- The entire global variable name goes into the external data queue.
- All subnodes of the derived name are listed.
- The Number of Subnodes field on the second line of the pair of messages in the external data queue for each node always contains zero.
- Returns the number of subnodes listed in the external data queue
- Places in the external data queue two lines per subnode and the entire Global Variable name
- Returns no partially updated symbol names

Example:

```
RTVL = OPSVALUE(derivedname, 'T')
```

■ U (Update)

- Assigns *newvalue* as the value of the compound symbol specified by *derivedname*. If the compound does not exist, OPS/REXX creates it then gives it the new value.
- Returns the value specified by *newvalue*
- Does not change the external data queue
- Prevents others accessing compound symbols from seeing partially updated symbols

Example:

```
RTVL = OPSVALUE(derivedname, 'U, newvalue)
```

■ V (Value)

- Returns the current value of the node specified by *derivedname*. If the node does not exist, OPS/REXX creates it but assigns it no value (giving the symbol the same value as its name).
- Does not change the external data queue.
- Prevents the issuer of OPSVALUE from seeing partially updated symbols.

Example:

```
RTVL = OPSVALUE(derivedname, 'V')
```


Examples: OPSVALUE

Review the following examples for a better understanding of OPSVALUE:

- Example 1: The following example shows how you can use a serialized update of global variables to implement an enqueue/dequeue strategy that is similar to the RDF table editor:

```

enqsys = OPSINFO('MSFID'); vrc=0 /* Get our sysid */
enqnew = 'HELD BY' userid 'ON' enqsys /* New value for enqueue */
if sysid <> " then do /* Remote resource? */
  Address 'OPSTCL' "MSF DEFAULT SYSTEM('sysid')" /* Xsys request*/
  enqsys = sysid /* Sysid for variable name */
end
enqvar = 'GLVTEMP0.ENQ.'enqsys'.resource' /* ENQ variable name */
do i=1 to 100 while vrc=0 /* Try 100 times or works */
  enqold = OPSVALUE(enqvar,V) /* Get current ENQ value */
  if enqold = " | enqold = enqvar then /* if ENQ is free */
    vrc = OPSVALUE(enqvar,C,enqnew,enqold) /* Grab ENQ */
  else
    rc = OPSWAIT(1) /* Wait a little and retry */
  end
if sysid <> " then /* Reset Xsys environment */
  Address 'OPSTCL' "MSF DEFAULT SYSTEM(*)"
if vrc = 0 then /* Test ENQ obtained */
  say 'Enqueue of 'enqsys'.resource' failed. enqold /* Failure */
else
  deqlst = deqlst enqvar /* Save var name for DEQ */

```

- Example 2: The following example deletes variables that are subnodes of GLOBALA.TEST. if they have not been referenced in one year. The last reference date is word 12 of the variable data record.

```

xrc = OPSQLEDQ() /* Clear EDQ */
vcnt = OPSVALUE('GLOBALA.TEST;',T) /* Get data on subnodes */
do i=1 to vcnt while QUEUED() > 0 /* Loop thru subnode data */
  pull varname /* First record is name */
  pull vardata /* Second is data record */
  if WORD(vardata,12) = NONE then /* Real old variable */
    vardate = 19900101 /* Set to ancient date */
  else
    /* Get data as yyyyymmdd */
    vardate = SPACE(TRANSLATE(WORD(vardata,12),' '),0)
  if DATE('B') - DATE('B',vardate,'S') > 365 then /* Not used? */
    rcnt = OPSVALUE(varname,'R') /* Delete var + subnodes */
  end
end

```

- Example 3: The following example DROPS (deletes) all variables that are subnodes of GLOBALA.TEST. and sets GLOBALA.TEST. to uninitialized:

```

val = OPSVALUE('GLOBALA.TEST;',D) /* Delete all subnodes */

```

- Example 4: The following example requests high-level security for the highest level of authority that is allowed. High-level security reduces overhead in applications that perform a large number of global variable operations.

```
sec = OPSVALUE('GLOBAL.READWRITE','H') /* Request update auth */
if sec <> 'AUTH' then
  sec = OPSVALUE('GLOBAL.READONLY','H') /* Try read only auth */
```

OPSVASRV() Function

The OPSVASRV() function creates and manipulates sysplex variables that are defined in the CA CCS Common Variable Service component (CAVARSRV). These variables are independent from the standard CA OPS/MVS global variables. Currently these sysplex variables are only stored in main storage of the CCS CAVARSRV component. This CCS CAVARSRV component is active on each system within the sysplex, providing a tool to any CA OPS/MVS automation that needs to store and manipulate data within the boundaries of a sysplex through this OPSVASRV() function. As long as one system in the sysplex remains active, the variables that are created through this OPSVASRV() function within any CA OPS/MVS automation on any system, continue to exist across system IPLs.

Utilization Environments

The OPSVASRV() function can be utilized within specific AOF rules, (ARM, API, and REQ), OPS/REXX programs, TSO/E programs, and OMVS scripts. Additionally, an OPSVASRV command processor version of this function exists, and it can be utilized from the TSO foreground or within a batch if needed. For more information, see the [POI Command Process](#) (see page 689) for details on invoking the OPSVASRV command processor.

Installation Requirements

The CA CCS Common Variable Service component must be installed on each system in which CA OPS/MVS automation will be creating and manipulating sysplex variables. Refer to CCS Common Variable Service (CAVARSRV) documentation for complete details on installing and utilizing this CCS component.

No special installation requirements or parameters are needed within CA OPS/MVS to use this function. However, the GLVNOTIFYRULES CA OPS/MVS parameter needs to be set to YES, if AOF GLV rules will be created to process specific sysplex variables.

Coding Syntax

somevarname = OPSVASRV("function [supporting keywords]")

Refer to the [Sample Coding Statements](#) (see page 665) for specific syntax details of each of these listed functions and keywords. Valid function options:

function	Function Description
CREATE	Creates a single variable with specific attributes and a value as defined within the supporting NAME, DATAVAR or DATAVAL keywords.
DECR	Decrement a single variable that was created as a DATATYPE(SIGNED32) type, by a numeric value that is specified in the supporting DATAVAL or DATAVAR keywords.
DELETE	Delete a single variable as specified within the supporting NAME keyword.
DELLIKE	Delete all variables matching a name mask as specified within the supporting NAME keyword.
GET	Retrieve the value of a single variable as specified within the supporting NAME keyword.
GETNEXT	Retrieve the value of the next variable matching a variable name mask as specified within the supporting NAME and TOKEN keywords.
GETLIKE	Read the variables that match a name mask as specified within the supporting NAME keyword and return the list of variable names, values, and attributes. The OPTIONS keyword can be used to select or limit the data that is returned.
INCR	Increment a single variable that was created as a DATATYPE(SIGNED32) type, by a numeric value that is specified in the supporting DATAVAL or DATAVAR keywords.

function	Function Description
PURGE	Purge/Delete all sysplex variables created by the same CA OPS/MVS sub system across all sysplex systems.
SWAP	Update a variable as a compare and swap instruction utilizing the new value as specified with the DATAVAL or DATAVAR keywords and current value as specified with the CURVAL or CURVAR keywords.
UPDATE	Updates an existing single variable as specified by the supporting NAME keyword with a value as defined by the supporting DATAVAL or DATAVAR keywords.

Valid [supporting keywords]:

Supporting keyword	Keyword Description
NAME(Variable name or mask)	Specify the name or name mask of the sysplex variable you want to access. The variable name that you specify must begin with the sysplex variable stem value GLVPLXTx, where x=A-Z or 0-9. Like OPSVALUE, A-Z can fire CA OPS/MVS AOF Global Variable Rules (GLV) rules for value changes. The 0-9 suffix never fires GLV rules. The maximum character length is 128 characters. Name lengths less than 80 characters are recommended. Required keyword for CREATE, DELLIKE, GETLIKE, and GETNEXT functions
DATAVAL('new data value')	Defines the new data literal value that is to be assigned to the specified variable. For the variable values containing non character or complex character strings such as mixed quotation marks, utilize the DATAVAR keyword instead of this DATAVAL keyword.

Supporting keyword	Keyword Description
DATAVAR(variable name)	Defines the new data literal value that is to be assigned to the specified variable using the name of a simple variable created within the application (supported AOF rule, OPS/REXX program, TSO/E REXX)
DATATYPE(data type)	Defines the type of the variable data. Valid data types are: ■ STRING A Text string ■ SIGNED32 32-bit signed integer ■ XML An XML document ■ UNDEF Undefined Default: UNDEF If DATATYPE is specified on the CREATE function it is required on the UPDATE function.
CURVAL(current data value)	Specify the current value of the variable. This keyword is only available when using the SWAP function.
CURVAR(variable name)	Simple variable name that contains the current value of the variable. This keyword is only available when using the SWAP function.

Supporting keyword	Keyword Description
OPTIONS(option1,option2,...)	Provide additional processing options for specific functions. Valid options are: ■ STATS Include variable name metadata that is associated with variable names returned by the GETLIKE, GET, and GETNEXT request. STATS is a default value for GETNEXT. ■ SUBNAME Perform z/OS system symbol substitution on the variable name specified within the NAME keyword. ■ NOVALUES When used with the GETLIKE, GET, and GETNEXT function, do not return the variable value data for each variable. ■ NOSYNCPDT Prevents waiting for the variable operation to complete. Treats the sysplex variable request as asynchronous and always returns RC=0, RSN=0 even if the request fails.
GROUP(group name)	Specify a name to associate variables by an application or component name. Group name must be 16 characters or less.
DELAY(number of secs)	The DELAY keyword indicates the number of seconds that your OPSVASRV request waits before it can be processed. The default is 0; you can specify any number of seconds from 1 to 300. Delay is ignored when executing in an AOF rule.
PREFIX(output variable prefix)	Optionally specify the stem name that is to be used for any returned output variables
SUBSYS(ssid)	The CA OPS/MVS subsystem name that is to execute the specified function. This keyword is only needed when multiple copies of CA OPS/MVS exist and the function is to occur on a specific CA OPS/MVS subsystem.

Supporting keyword	Keyword Description
TOKEN(token id number)	Identifies the next internally stored variable that is to be obtained via the GETNEXT function. This token id would have been obtained by a previously issued GET function.

Generated Output

The generated or returned output is dependent on the specified function. Every specified function will always result in the following default variables being created within the executing rule or program:

vasrv_rc

Contains the return code from the specified variable function

vasrv_rsn

Contains the Reason code from the specific variable function

Refer to [Return Codes](#) (see page 669) for a complete list of RCs and RSNs.

GET function variable output:

The GET function retrieves or obtains the value of the specified variable as defined in the NAME keyword, and assigns it to the calling variable of the function. For example:

```
payroll_status = OPSVASRV("GET NAME(GLVPLXT0.APPL.PAYROLL.STATUS)")
```

The value of the GLVPLXT0.APPL.PAYROLL.STATUS variable would be assigned to the payroll_status variable. within the executing program.

When the OPTION(STATS) keyword is specified for a GET request, the output variables produced by GETLIKE are also created for GET. The stem variable index is omitted from the variables, since only one instance of each variable is created.

GETNEXT function variable output:

The GETNEXT function obtains the value of the next variable that matches the variable name mask specified in the NAME keyword. The output TOKEN value of the prior request acts as the start location for the next matching variable search. For example:

```
Next_value=OPSVASRV("GETNEXT NAME(GLVPLXT0APPL.PAYROLL.*) TOKEN("vasrv_token")")
```

The OPTION(STATS) keyword is the default for a GETNEXT request. The output variables produced by GETLIKE are also created for GETNEXT. The stem variable index is omitted from the variables, since only one instance of each variable is created. The VASRV_NAME variable provides the complete variable name of the returned variable.

GETLIKE function variable output:

The GETLIKE function reads the variables that match a name mask as specified within the supporting NAME keyword and returns the list of variable names, values, and attributes in unique stemmed variables. The default stemmed variable names begins with VASRV_, or can optionally be named by using the PREFIX keyword. The OPTIONS keyword can be used to select or limit the data that is returned if needed.

The default stemmed variables that are created from a GETLIKE function call without the PREFIX keyword are:

VASRV_NAME.0

The count total of variables that match the specified stem mask.

VASRV_NAME.x (x = 1 to total count returned in vasrv_name.0)

Full variable name

Length: 1-128

Example: GLVPLXT1.STATUS.SYSTEM

VASRV_VALUE.x (x = 1 to total count returned in vasrv_name.0)

These VALUE variables contain multiple word values.

Note: These variables are not created if the OPTIONS(NOVALUES) keyword was specified on the GETLIKE function.

Word 1

Variable data type name

Length: 8

The Supported data types are: UNDEF, SIGNED32, STRING, or XML.

Words 2-n

Variable data value

Length: 1-30000

Example value for a VASRV_VALUE.x variable: UNDEF PROD CICS ACTIVE ON
SYSTEMX

VASRV_TOKEN.x (x = 1 to total count returned in vasrv_name.0)

The variable token value in printable hex characters.

Note: These variables are only created when the OPTIONS(STATS) keyword is specified on the GETLIKE function.

Length: 32

Example value for a VASRV_TOKEN.x variable: FFFFFFFF2F4F6F9FFFFFFFAFCF2F0F0

VASRV_INFO.x (x = 1 to total count returned in vasrv_name.0)

These INFO variables contain multiple word values.

Note: These variables are only created when the OPTIONS(STATS) keyword is specified on the GETLIKE function.

Word Number: 1

Variable product name

Length: 1-16

For CA OPS/MVS, the variable product name is always CA OPS/MVS.

Word Number: 2

Variable instance name

Length: 1a-16

For CA OPS/MVS, the variable instance name is always a CA OPS/MVS subsystem name such as OPSS.

Word Number: 3

Variable group name

Length: 1-16

Example: PRODPAYROLL

Word Number: 4

Variable data type name

Length: 5-8

Example: STRING

Word Number: 5

Variable scope

Length: 7

For CA OPS/MVS, the scope is always SYSPLEX.

Word Number: 6

Variable is eligible to trigger a)GLV rule.

Length: 1

Y=Yes or N=No

Word Number: 7

Persist attribute.

Length: 4

TEMP (temporary) is the only value supported.

Word Number: 8

Length: 1-8

Contains the system name that created the variable.

Word Number: 9

Creation user ID or job name.

Length: 1-8

Word Number: 10

Length: 1-8

Contains the program name or the memory address of the program that created the variable.

Word Number: 11

Length:10

Creation date in YYYY/MM/DD format

Word Number: 12

Length:8

Creation time is in HH:MM:SS (UTC time)

Word Number: 13

Length: 1-8

Contains the system name that updated the variable

Word Number: 14

Length: 1-8

Contains the User name or Job name that updated the variable

Word Number: 15

Length: 1-8

Contains the name or the memory address of the program that updated the variable

Word Number: 16

Length:10

Update date in YYYY/MM/DD format

Word Number: 17

Length:8

Update time is in HH:MM:SS format (UTC time)

Word Number: 18

Length:10

Last reference date in YYYY/MM/DD format

Word Number: 19

Length:8

Last reference time is in HH:MM:SS format (UTC time).

Refer to [Sample Coding Statements](#) (see page 665) for samples on using the GETLIKE function for more details.

Sample Coding Statements

The following generic syntax demonstrates the most common usage of the OPSVASRV() function in which the automation creates, updates, obtains, and deletes the CCS Sysplex Variable Service variables.

Creating basic sysplex variables using the CREATE function:

```
/*-----*/
/* Create basic variable named GLVPLXT0.VARTEST.1 with desired */
/* value using the NAME and DATAVAL keywords. This variable can then */
/* be accessed by any OPS/MVS system that is also running with the */

/* CCS CAVASRV component active. */
/*-----1-----2-----3-----4-----5-----6-----7*/
setvar = OPSVASRV("CREATE NAME(GLVPLXT0.VARTEST.1)",
                  "DATAVAL('Specify contents of the variable here')")

/*-----*
/* Use simple variables that will identify the name of the variable *
/* and its value, using the NAME and DATAVAR keywords. This is useful *
/* if the variable value is lengthy and/or contains mixed quotation *
/* marks or any other special characters. */
/*-----1-----2-----3-----4-----5-----6-----7*/
varname = 'GLVPLXT0.VARTEST.2'
varval = 'Specify the contents of the variable here'
setvar = OPSVASRV("CREATE NAME('varname') DATAVAL('varval')")
```

Update a variable, and create if it does not exist:

```
/*-----*/
/* Update a basic variable named GLVPLXT0.VARTEST.1 with desired */
/* value. Always manipulate the returned rc/rsn to determine if the */
/* variable does not exist (rc=4,rsn=8/16). In this sample case,the */
/* logic will simply CREATE the variable if it does not exist, but */
/* other logic can be created (send out alert, restart some process, */
/* etc) if the variable was not previously created. */
/*-----1-----2-----3-----4-----5-----6-----7*/
varname = 'GLVPLXT0.VARTEST.1'
varval = 'Contents of the variable value'
update = OPSVASRV("UPDATE NAME('varname') DATAVAR(varval)")
if vasrv_rc = 4 & (vasrv_rsn = 8 | vasrv_rsn = 16) then
do
createvar = OPSVASRV("CREATE NAME('varname')",
                    "DATAVAR(varval)")
end
```

Obtaining and manipulating basic sysplex variables using the GET and the GETLIKE functions:

```
/*-----*/
/* Use simple variables that will identify the name of the variable */
/* to be obtained. If desired, this can be hardcoded directly in the */
/* NAME keyword. These simple statements will obtain a previously */
/* created sysplex variable called GLVPLXT0.VARTEST.1 and its value */
/* will be returned to the variable that invoked the function (getvar)*/
/* Simply SAY the contents of this variable. */
/*-----1-----2-----3-----4-----5-----6-----7*/
varname = 'GLVPLXT0.VARTEST.1'
getvar = OPSVASRV("GET NAME("varname")")
say ****The value of sysplex variable varname=varname

/*-----*/
/* Obtain and manipulate a mask of variables. Assuming the sysplex */
/* variables exist named:GLVPLXT0.IPLDATA.SYSA, GLVPLXT0.IPLDATA.SYSB, */
/* GLVPLXT0.IPLDATA.SYSC, etc the following code would obtain and loop*/
/* through each variable. */
/*-----1-----2-----3-----4-----5-----6-----7*/
varnamemask = 'GLVPLXT0.IPLDATA*'
getvar = OPSVASRV("GETLIKE NAME("varnamemask")")
say ****Total number of variables with mask= varname.0
do v = 1 to varname.0
say Variable name=varname.v
say Value of var=varname.v
end

/*-----*/
/* Obtain and manipulate a mask of variables and request variable */
/* metadata so that the group name, create date, and create time can */
/* be manipulated in addition to the variable names and values. */
/*-----1-----2-----3-----4-----5-----6-----7*/
varnamemask = 'GLVPLXT0.VARX*'
getvar = OPSVASRV("GETLIKE NAME("varnamemask")",
"OPTIONS(STATS)")
do v = 1 to varname.0
groupname = WORD(varname.v,3)
createdate = WORD(varname.v,11)
createtime = WORD(varname.v,12)
say **Variable name=varname.v
say ' Value =varname.v
say ' Group =groupname
say ' CDATE =createdate
say ' CTIME =createtime
end
```

Obtaining and manipulating a large variable mask that exceeds the GETLIKE constraints:

```
/*-----*/
/* Assuming you have a very large mask of plex variables that you */
/* need to manipulate that exceed GETLIKE storage constraints, use */
/* the GETNEXT function to loop through and process each variable. */
/* The first GETNEXT will obtain the first variable in the mask, and */
/* also set the pointer or TOKEN that will be used in the subsequent */
/* GETNEXT that is done within a loop. At the end of the variable list */
/* a non vasrv_rc will be returned to end the loop. */
/*-----1-----2-----3-----4-----5-----6-----7*/
varcnt = 1
varnamemask = 'GLVPLXT0.VARDG*'
firstvarval = OPSVASRV("GETNEXT NAME(varnamemask)")
say 'Value of variable #1 in mask=firstvarval'
do while vasrv_rc = 0
  varcnt = varcnt + 1
  nextvarval = OPSVASRV("GETNEXT NAME(varnamemask)",
    "TOKEN(vasrv_token)")
  if vasrv_rc = 0 then,
    say 'Value of variable' vasrv_name='nextvarval'
end
```

Creating, incrementing, decrementing, and obtaining counter variables using the INCR and/or DECR functions:

```
/*-----*/
/* Create a counter type variable that will be used within the INCR */
/* and/or DECR functions to automatically increment/decrement the */
/* value. Set value to initial value (zero), and use required */
/* DATATYPE(SIGNED32) keyword. */
/*-----1-----2-----3-----4-----5-----6-----7*/
setvar = OPSVASRV("CREATE NAME(GLVPLXT0.ABENDCNTR)",
  "DATAVAL(0)",
  "DATATYPE(SIGNED32)")

/*-----*/
/* Add some value to the counter variable using the INCR function. */
/* This sample code will add 1 to a variable called GLVPLXT0.ABENDCNTR */
/* Simply change INCR to DECR if you are decrementing the value. */
/*-----1-----2-----3-----4-----5-----6-----7*/
varname = 'GLVPLXT0.ABENDCNTR'
add12var = OPSVASRV("INCR NAME(varname)",
  "DATAVAL(1)")

/*-----*/
/* Obtain a counter variable that was incremented/decremented with */
/* the INCR/DECR functions. Wrap the REXX C2D() function around the */
```

```
/* the internally stored binary value to view the decimal format */
/*-----1-----2-----3-----4-----5-----6-----7*/
varname = 'GLVPLXT0.ABENDCNTR'
getvar = OPSVASRV("GET NAME("varname")")
say ***The value of counter variable varname='C2D(getvar)
```

Deleting sysplex variables:

```
/*-----*
/* These simple statements will delete a single variable called *
/* GLVPLXT0.VARTEST.1, thus making it non-existent to any automation *
/* executing on any system in the sysplex. *
/*-----1-----2-----3-----4-----5-----6-----7*/
varname = 'GLVPLXT0.VARTEST.1'
delvar = OPSVASRV("DELETE NAME("varname")")

/*-----*
/* Delete a mask variables. *
/*-----1-----2-----3-----4-----5-----6-----7*/
varnamemask = 'GLVPLXT0.IPLINFO*'
delvars = OPSVASRV("DELLIKE NAME("varnamemask")")

/*-----*/
/* Purge all sysplex variables within the CCS CAVARSRV component */
/* !WARNING!! Use this function with caution as sysplex variables */
/* are not checkpointed in any copy of the CCS CAVARSRV component.Thus*/
/* a PURGE will cause all applications running on all OPS/MVS regions */
/* to no longer have access to any sysplex variable. */
/*-----1-----2-----3-----4-----5-----6-----7*/
purgeall = OPSVASRV("PURGE)
```


Debugging Guidelines

Incorporate a 'SIGNAL on SYNTAX' routine within the rule or program to obtain and display any error messages that may be returned during a function call failure. Additionally, utilize a 'TRACE I' REXX statement when needed before the function call to obtain the logical trace activity within the invoking rule or program. This debugging code resembles:

```

/*-----*
/* Create Signal on Syntax routine to obtain any failure messages.  *
/*-----1-----2-----3-----4-----5-----6-----7*
signal on syntax
/*-----*
/* Add TRACE I if needed to trace logic flow of rule/program      *
/*-----1-----2-----3-----4-----5-----6-----7*
trace I
varname = 'GLVPLXT0.VARTEST.1'
varval = 'TEST NUMBER 1'
setvar = OPSVASRV("CREATE NAME('varname') DATAVAL('varval')")
return
syntax:
do queued()
  pull err_info
  say 'Failure info from OPSVASRV=err_info'
end

```

Return and Reason Codes

The default vasrv_rc and vasrv_rsn variables contain the return and reason codes of the OPSVASRV function call. This table contains the common non-zero error code conditions. Refer to the CA CCS Sysplex Variable Service for a complete list of these codes if not documented in this table.

Return Code	Reason Code	Description
4	4	Duplicate variable
4	8	Variable not found
4	12	A retrieve type request was made but the user supplied area was not large enough to return all data
4	16	No variables exist within CCS CAVASRV from any CA OPS/MVS system.
4	20	No match on masks found.

Return Code	Reason Code	Description
4	40	A SWAP request was made, but the result of the comparison is unequal. No variable update takes place.
8	4	CCS CAVARSRV Component is unavailable.
12	4	A INCR/DECR function failed because the variable was not created as a DATATYPE(SIGNED32) type variable.

OPSVSAM Function

The OPSVSAM function enables OPS/REXX programs running in a non-AOF environment to read and write VSAM KSDS, ESDS, and RRDS files. Rules do not permit access to VSAM files. Supported VSAM functions include: open, close, point, read, read for update, update, insert, delete, and end request.

Note: The OPSVSAM function can be used in OPS/REXX or AOF rules.

This function has the following format:

```
var = OPSVSAM({function},{ddname},arg3,arg4,arg5,arg6)
```

Arguments for OPSVSAM

You must always specify the *function* and *ddname* arguments. You can specify up to five arguments; any inapplicable arguments are ignored.

Access Strategies

The following are the access strategies for each type of VSAM file:

KSDS

A key sequenced data set may be accessed sequentially without a key argument, or it may be accessed directly by specifying a full or partial key. To avoid a record not found condition-unless the KGE search type is specified-any partial keys that you specify for point and read functions must match a real key up to the length specified.

The key argument for OPSVSAM is a string value. The key length may be specified when a substring or a blank padded extension of the string is desired. The default key length is the length of the key argument string.

RRDS

A relative record data set may be accessed sequentially without a key argument, or it may be accessed directly by specifying the relative record number of the desired record. Record inserts must indicate the new relative record number. You may specify the key argument for OPSVSAM as a REXX numeric string or a four-byte binary value (for example, '0000001'x).

ESDS

An entry-sequenced data set may be accessed sequentially without a key argument, or it may be accessed directly by specifying the relative byte address of the desired record. Record inserts are always placed at the end of a data set. You may specify the key argument for OPSVSAM as a four-byte RBA binary value only (for example, '000001A0'x).

Output Variables

OPS/REXX variables are created for each VSAM operation. These variables indicate the result of the operation. The result returned from the function is null, except for the READ and READUP requests, which return the record that was read.

OPSRC

The VSAM operation return code (0/4/8/12, and so on). The code 78 is returned for dynamic allocation failures and abends.

OPSRE

The reason code indicated in the last byte of the RPL feedback area. For detailed information about the meaning of these codes, see the IBM VSAM macro reference guide.

OPSRC	OPSRE	Meaning
8	4	End of file on read
8	16	Record not found for keyed access

OPSKEY

The key of the last record that was read or written. Relative record numbers and relative byte addresses are stored as four-byte binary numbers. To convert to decimal number strings, use C2D(opskey).

OPSRBA

The RBA of the last record that was read or written in four-byte binary format.

The following variables are set only by the OPEN operation:

Variable	Definition
OPSDDD	The ddname of the file. This variable can be used for the ddname argument of all subsequent operations.
OPSDSN	The data set name of the file.
OPSLRECL	The maximum record size.
OPSKEYLN	The KSDS full key length.
OPSKEYOF	The KSDS key offset relative to one. You can use this offset as the begin value in a REXX SUBSTR function to extract the <i>key</i> value from a returned record.

Types of OPSVSAM Functions

The following lists and describes the available OPSVSAM functions:

- This function opens a VSAM file for input or output.

```
var = OPSVSAM('OPEN','ddname','dsname','I' or 'O','RLS' or 'RES')
```

One or both of these variables must be specified:

ddname

dsname

If only the *ddname* variable is specified, the file must be preallocated to that *ddname*. If only the *dsname* variable is specified, the file is dynamically allocated to a system-generated *ddname*. If both the *ddname* and *dsname* variables are specified, the *dsname* is allocated to the specified *ddname*.

Input ('I') is the default open mode. To update a file, the output mode ('O') must be specified.

RLS (Record Level Sharing) is a feature of z/OS that allows VSAM files to be shared across systems in a sysplex. Consult your systems programming group regarding the availability of this feature. To use RLS, the VSAM file must be SMS-managed and defined with the LOG(NONE) parameter.

RES specifies that a reserve should be issued for the data set to serialize access across systems. The major name for the reserve is the ddname used and the minor name is SVDB. Because a reserve can cause system performance problems, you should consult with your systems programming group before using this option.

The default is not to use RLS or RES.

- This function closes the VSAM file that is allocated to the specified *ddname*.

```
var = OPSVSAM('CLOSE','ddname')
```

If the file was dynamically allocated by an OPEN function, it will also be dynamically deallocated.

- This function temporarily closes the file and ensures the following:

- All modified VSAM records are written to DASD
- All catalog information is updated to reflect any changes to the high-used RBA and any additional extents that were allocated
- SMF usage data is generated

```
var = OPSVSAM('CLOSET','ddname')
```

Note: An OPEN function is not required to continue file processing.

- This function issues a VSAM ENDREQ, freeing any control interval locking that was caused by a prior READUP call.

```
var = OPSVSAM('ENDREQ','ddname')
```

- This function positions the file to the record specified by the *key* for subsequent sequential processing. *Key length* is relevant only for KSDS keys when specifying the substring length of the string specified in the *key* argument. It should not be specified for RRDS and ESDS data sets. KEQ (key equal) is the default key match criteria. If KGE (key greater than or equal to) is specified and the key is not matched, the next higher key is pointed to.

```
var = OPSVSAM('POINT','ddname','key',key length,'KGE' or 'KEQ')
```

- This function reads the next VSAM record or a specific VSAM record. If no key data is specified, sequential access is attempted. If key data is specified, direct access is attempted through the specified key. This function returns the retrieved record.

```
var = OPSVSAM('READ','ddname','key',key length,'KGE' or 'KEQ')
```

- This function is the same as the READ function, except that a write request to update the record may follow. A control interval lock is obtained to prevent concurrent access to the same record by another task using shared access to the same file. ENDREQ, DELETE, UPDATE, or another READ/READUP request frees this lock.

```
var = OPSVSAM('READUP','ddname','key',key length,'KGE' or 'KEQ')
```

- This function replaces the record data at the current position that was established by a prior READUP request. You must specify the *record* data.

```
var = OPSVSAM('UPDATE','ddname','record')
```

- This function deletes the record at the current position that was established by a prior READUP request.

```
var = OPSVSAM('DELETE','ddname')
```

- This function adds a new record with the specified *key*. For KSDS data sets, you do not have to specify the *key* argument; the *key* is obtained from the record. For RRDS data sets, you must specify the *key*. For ESDS data sets, you should *not* specify the *key*. All records are appended to the end of the data set.

```
var = OPSVSAM('INSERT','ddname','record','KEY',key length)
```

Example: OPSVSAM

This example illustrates performing VSAM file operations on a VSAM KSDS with a character key length of 10 that begins in position 1 of a maximum 100-byte record:

```
Signal On Syntax Name VSAM_ERROR
/* Dynamically allocate and open the VSAM file */
vrc = Opsvsam('OPEN','TEST.VSAM.KSDS','O')
if opsrc > 0 Then Signal VSAM_ERROR
/* Insert a new record with key = OPSMVS */
newrec = Substr('OPSMVS',1,opskeyln) || 'TEST DATA RECORD'
vrc = Opsvsam('INSERT',opsdd,newrec)
if opsrc > 0 Then
  if opsrc = 8 & opsre = 8 Then
    Say 'Duplicate record:' newrec
  Else
    Signal VSAM_ERROR
/* Retrieve and delete the record we just added */
vrc = Opsvsam('READUP',opsdd,Substr(newrec,1,opskeyln))
if opsrc = 0 Then
  vrc = Opsvsam('DELETE',opsdd)
Else
  Signal VSAM_ERROR
/* Clean up and exit */
Signal ALL_DONE
/* VSAM error procedure */
VSAM_ERROR:
  Say 'OPSVSAM error: RC=opsrc REASON=opsre DD=opsdd'
  Do While Queued() > 0
    Pull xdcmsg
    Say xdcmsg
  End
ALL_DONE:
  lrc = opsrc
  vrc = Opsvsam('CLOSE',opsdd)
  Exit lrc
```

OPSWAIT Function

The OPSWAIT function can be used in an OPS/REXX program or REQ rule to suspend processing of the program or rule for a specified period of time.

Note: The OPSWAIT function can be used in OPS/REXX, AOF rules, TSO/E REXX, or CLIST.

This function has the following two formats:

```
var = OPSWAIT(interval)
```

```
var = OPSWAIT("any valid OPSWAIT POI command keywords")
```

interval

Defines the amount of time OPSWAIT should suspend processing. You may specify a value in hh:mm:ss format, or the number of seconds. The number may contain a decimal point and up to two decimal digits (for indicating fractional seconds). If you specify 0 or a negative value, OPSWAIT does not suspend processing.

Values OPSWAIT Returns

OPSWAIT always returns a null string.

Usage Notes for OPSWAIT

Keep these notes in mind when using OPSWAIT:

- Do not use the OPSWAIT function in any type of rule other than an REQ rule. Using OPSWAIT in a TOD rule delays the execution of other TOD rules. The specification of OPSWAIT in any other type of rule is ignored.
- When a particular operation (such as waiting) needs to be performed from an OPS/REXX program or rule, and the operation can be performed by either using a TSO command processor or by using an OPS/REXX function or host environment command, the OPS/REXX function or host environment command is always more efficient.

When the same operation is performed repeatedly, such as in a loop, the difference in CPU resources that are consumed can be significant.

For these reasons, we recommend that whenever it is possible, you should use the OPSWAIT function in OPS/REXX programs and REQ rules rather than the OPSWAIT TSO command processor.

For more information, see Special Guidelines for POI Command Processors in the chapter "POI Command Processors."

OPSWLM Function

The OPSWLM OPS/REXX function performs one of the following operations:

- Retrieves information about the state of a WLM scheduling environment available on a specified system. A scheduling environment is a group of some number of resources, each having a required state. If all of the resources are in the required state, then the scheduling environment is available. If any of the resources is not in the required state, then the scheduling environment is not available.
- Sets the status of a WLM scheduling environment resource.

Note: The OPSWLM function can be used in OPS/REXX or AOF rules.

This function has the following format:

```
var = OPSWLM('{func}','{name}',[arg3])
```

func

Specifies the type of function you want the OPSWLM function to perform. You must always specify the *func* argument.

Currently, the only valid function codes for OPSWLM are:

I

Retrieves information about the state of a WLM scheduling environment. This function determines if a scheduling environment is available on a specified system. This function code may be used in any environment where OPS/REXX can be used. To get meaningful results, you should only specify the names of scheduling environments that are defined in the active service policy.

S

Sets the status of a WLM scheduling environment resource. This function code may only be used in real AOF rules. When used in AOF Test, no action is taken and a fixed result string is returned. To get meaningful results, you should only specify the names of scheduling resources that are defined in the active service policy on this system.

Q

Query. This function code is reserved for future use and must not be used.

name

The value you specify for the required name argument depends on the function code:

- For function code I, specify the name of a WLM scheduling environment.
- For function code S, specify the name of a WLM scheduling environment resource.

arg3

The value you specify for the *arg3* argument depends on function codes I or S.

- For function code I, specify the name of the system on which you want to check the status of the WLM scheduling environment whose name you specified in the *name* argument. If you do not specify this argument, then the local system name is used. Note that in this context, system name refers to the z/OS SYSNAME parameter value as specified in logical parmlib member IEASYSxx. You can specify the name of any system that is part of the same sysplex as the system on which this function is executing.

When the I function code is used, the following values may be returned:

AVAIL

The scheduling environment on the designated system is available.

NOTAVAIL

The scheduling environment on the designated system is not available.

NOTSUPPT

The system does not support scheduling environment services. This return code is set for unsupported versions of the operating system.

NOTEXIST

The named scheduling environment does not exist on the designated system.

ERROR RC=*nn* RS=xxxxxxxx

The IBM IWMSEDES service returned an unexpected return code and reason code combination. This string contains the return code (in decimal) and reason code (in hexadecimal) from the IWMSEDES service. In addition, message OPx4280E is issued and contains similar information. Check the IBM documentation for the WLM IWMSEDES service and attempt to diagnose and correct the problem. If you are unable to resolve the problem, then contact CA customer support and provide them with the AOF rule or OPS/REXX program and the exact contents of this value.

For assistance, contact Customer Support at <http://ca.com/support>

OPSDOWN

The CA OPS/MVS main address space is not active.

ABEND

An abend occurred either in the OPSWLM function or in the product WLM service routine. Check the OPSLOG for message OPx9500S. The message indicates the abend code as well as the module name and offset where the abend occurred. For assistance, contact Customer Support at <http://ca.com/support>.

- For function code S, specify the state to which you want the WLM scheduling environment resource to be set. The only allowable values are ON, OFF, and RESET. You must always specify this argument.

When the S function code is used, the following values may be returned:

OK

This string indicates that the state of the designated WLM scheduling environment resource was set as directed.

NOTSUPPT

The system does not support scheduling environment services. This return code is set for unsupported releases of the operating system.

NOTEXIST

The named resource does not exist. Check the *name* argument. If the specified resource name is correct, then check whether the resource is defined in the active WLM service policy.

ERROR RC=nn RS=xxxxxxxx

The IBM IWMSESET service returned an unexpected return code and reason code combination. This string contains the return code (in decimal) and reason code (in hexadecimal) from the IWMSESET service. In addition, message OPx4280E is issued and contains similar information. Check the IBM documentation for the WLM IWMSESET service and attempt to diagnose and correct the problem. If you are unable to resolve the problem, then contact CA customer support at <http://ca.com/support> and provide them with the AOF rule or OPS/REXX program and the exact contents of this value.

OPSDOWN

The CA OPS/MVS main address space is not active.

ABEND

An abend occurred either in the OPSWLM function or in the product WLM service routine. Check the OPSLOG for message OPx9500S. The message indicates the abend code as well as the module name and offset where the abend occurred. For assistance, contact customer support at <http://ca.com/support>

AOF_TEST

The Set function code was used in AOF Test.

Note: A syntax error (for example, using an invalid function code) in the function call statement causes an OPS/REXX error 40, Incorrect Call to Routine.

OPSWLM Scheduling Examples

- Example 1: This example shows the status of the WLM scheduling environment named IMSA on the current system:

```
temp = OPSWLM("I",IMSA)
```

- Example 2: This example shows the status of the WLM scheduling environment named IMSA on system SYSA:

```
temp = OPSWLM("I",IMSA,"SYSA")
```

- Example 3: This example sets the state of the WLM scheduling environment resource named Q51 to ON, on the current system:

```
temp = OPSWLM("S",Q51,"ON")
```

- Example 4: This example resets the state of the WLM scheduling environment resource named Q51 on the current system:

```
temp = OPSWLM("S",Q51,"RESET")
```

OPSWORD Function

The OPSWORD function provides compatibility for translated Automate rules that contain the &WORD nn and &MLWORD nn environmental variables.

Like the REXX WORD function, OPSWORD returns the n th word in *string*. Unlike the REXX WORD function, the OPSWORD function delimits words by both blanks and commas. This feature is useful when messages contain positional output whose absence is indicated by successive commas in the message text.

Note: The OPSWORD function can be used in OPS/REXX or AOF rules. The OPSWORD function is not dependent upon CA OPS/MVS being active.

This function has the following format:

```
var = OPSWORD(string, $n$ )
```

string

The string is the input text string and may contain up to 32,000 characters.

n

The n is the sequential placement, in *string*, of the word you want OPSWORD to return. If n is greater than the total number of words in *string*, then null is returned.

Examples: OPSWORD

```
OPSWORD('HELLO,,MOTHER,HELLO FATHER',2)  □ *
```

```
OPSWORD('HELLO,,MOTHER,HELLO FATHER',3)  □ 'MOTHER'
```

```
OPSWORD('HELLO,,MOTHER,HELLO FATHER',5)  □ 'FATHER'
```

OPSYSPLX Function

The OPSYSPLX function returns information about a system in the current sysplex.

When the main address space is active, OPSYSPLX can be used in both AOF rules and OPS/REXX programs.

Note: The OPSYSPLX function can be used in OPS/REXX, AOF rules, TSO/E REXX, or CLIST.

This function has the following format:

```
systems = OPSYSPLX(function,type,selector)
```

function

Currently, the only valid function argument is I, for inquiry about information retrieval.

type

Currently, the only valid type argument is S (for system).

selector

Use the optional *selector* argument as a filter to reduce the amount of information that is being returned. When the *type* argument is set to S, the *selector* argument can be used to filter the system values that are returned in the EDQ. The *selector* value may contain single character wildcards (%), a suffix wildcard (*), or both.

Example: *selector* arguments

```
SYSA
```

```
SYS*
```

```
S%S*
```

Example: OPSYSPLX

This example returns information about all the systems in the sysplex:

```
temp = OPSYSPLX("I","S")
do temp
  pull line
  say line
end
```

The following record was returned in a local sysplex:

```
XAEI  EI LOCAL  AEBE70DBFCB89206  20030602 16:51:16 1  5  8500  8800  ACTIVE+SINGLE_SYSTEM
```

The following two records were returned on a two-sysplex system:

```
XEI3  13 AORPLX01 AEC00ECC0E441A06  20030602 18:00:59 1  5  6000  6500  ACTIVE
XE03  03 AORPLX01 AEC00ECC0E441A06  20030602 18:01:04 2  5  6000  6500  ACTIVE
```

Record That OPSYSPLX Returns

Each line that is added to the external data queue contains a record of the following format:

Word Number: 1

Length: 8

Contains the system name.

Word Number: 2

Length: 2

Contains the system clone ID (SYSCLONE).

Word Number: 3

Length: 8

Contains the sysplex name.

Word Number: 4

Length: 16

Contains the sysplex identifier token.

Word Number: 5

Length: 8

Contains the last date of status update, in the form DATE('S').

Word Number: 6

Length: 8

Contains the last status update time, in the form TIME('N').

Word Number: 7

Length: 4

Contains the system slot number.

Word Number: 8

Length: 4

Contains the system release.

Word Number: 9

Length: 8

Contains the monitor interval, in units of 1/100 seconds.

Word Number: 10

Length: 8

Contains the operator interval, in units of 1/100 seconds.

Word Number: 11

Length: 74

Contains the system status. The status consists of text substrings separated by plus (+) signs, indicating to REXX that this is a single word. Possible text substrings include:

- ACTIVE
- STATUS_UPDATE_MISSING
- IN_SYSPLEX_PARTITIONING
- SINGLE_SYSTEM
- CLEANUP

Note: The length of the above words may change in the future, but their relative position (word number) will not change.

OPSYSSYM Function

The OPSYSSYM function can be used in any AOF rules or OPS/REXX programs environment to return information about the z/OS system symbols. The System symbols are defined in the appropriate IEASYMxx member of the Logical Parmlib Concatenation.

This function has the following format:

```
var = OPSYSSYM(function[,prefix][,system])
```

function

Specifies the type of function you want the OPSYSSYM function to perform. The *function* argument must be specified.

Currently, the only valid function codes for OPSYSSYM are:

I

Returns one or more matching system symbol names and values in the external data queue.

R

Returns the value of the specified system symbol as the value of the function. If the symbol does not exist, an empty string is returned. No changes are made to the external data queue.

prefix

(Optional) Specifies one to ten characters that are compared against the defined system symbols. For the function 'I', the prefix matches and returns any defined system symbol up to the length of the prefix. If the prefix argument is omitted, then all of the system symbols are returned. For the function 'R' only a symbol that matches the prefix exactly is returned as the value of the function.

Note: If you omit the & character as the first character, it is automatically inserted.

system

(Optional) Provides CA OPS/MVS with the capability to execute the OPSYSSYM function on external systems that have been defined to the MSF. The MSF establishes VTAM or CCI sessions between copies of CA OPS/MVS, permitting any copy to issue a command on any other copy and to receive its response.

You can specify any of these values:

- MSF name of an individual system on which the function is to execute
- MSF alias of an individual system on which the function is to execute
- *DEFAULT - This value tells CA OPS/MVS to use the default system name set by the ADDRESS OPSCTL MSF DEFAULT host command

If the local MSF system name is used or implied, the system argument is ignored.

The number of seconds that the function waits for a response is determined by the ADDRESS OPSCTL MSF DEFAULT syswait value when it is greater than zero or the value of the MSFSYSWAIT parameter. When the system argument is specified for a remote system within a no-wait AOF rule, no output for the function is returned.

An invalid or inactive MSF system name causes the REXX program to be interrupted for a SYNTAX error. Use the ADDRESS OPSCTL MSF LIST host command to validate the MSF system name before using it as an argument for any REXX function.

Examples: OPSYSSYM Function

Reviewing the following examples gives you a better understanding of the OPSYSSYM function:

- Example 1: Consider the following sample OPSYSSYM statement:

```
temp = OPSYSSYM("I")
do temp
  pull line
  say line
end
```

This statement displays all of the defined system symbols on this system. For example:

```
&SYSOLONE DI
&SYSNAME XADI
&SYSFLEX LOCAL
```

- Example 2: The following sample OPSYSSYM statement returns the number of system symbols that begin with the string "&SYS":

```
temp = OPSYSSYM("I",&SYS)
```

- Example 3: The following sample OPSYSSYM statement returns the value of the &SYSNAME system symbol:

```
value = OPSYSSYM("R","SYSNAME")
```

- Example 4: This example shows how to use the OPSYSSYM function to display all the system symbols beginning with the prefix '&SYS' on a remote system with an MSF name of SYSB.

```
ADDRESS OPSCTL "MSF DEFAULT SYSTEM(SYSB) SYSWAIT(30)"
temp = OPSYSSYM("I","SYS","DEFAULT")
do temp
  pull line
  say line
end
```

Records that OPSYSSYM Returns

Each line that is added to the external data queue contains a record of the following format:

Word Number	Length	Description
1	10	System symbol string
2	50	System symbol value

Note: The lengths of these words may change in the future, but their relative position (word number) will not.

TIME Function—Returns Current or Elapsed Time

When executed in a stand-alone OPS/REXX program, the TIME function operates identically to standard REXX and returns the current or elapsed time in the format specified by *option*.

Following is a list of various AOF rule types and a description of how the TIME function operates when executed in them:

- In a REQ rule, this function operates identically to standard REXX and returns the current or elapsed time in the format specified by *option*.
- In a TOD rule, the current time returned is always the time when the rule was scheduled to execute.
- In any other type of rule, the current time returned is always the time when the rule actually executed.
- In any rule other than a REQ rule, the elapsed time returned is always zero.

The TIME function has this format:

```
var = TIME(option)
```

option

Supports all values for *option* that are supported by standard REXX.

TRACE Function

Like standard REXX, the TRACE function returns the current value of the TRACE setting and optionally changes it to the value specified for *setting*.

Unlike standard REXX, program lines consisting solely of comments do not appear in OPS/REXX TRACE output.

For details on the OPS/REXX TRACE instruction, see the chapter “Using OPS/REXX” in the *User Guide*.

This function has the following format:

`var = TRACE(setting)`

setting

Supports all values for *setting* that are supported by standard REXX, *except*:

Setting	Function
!	Inhibit host command execution
?	Control interactive tracing
<i>n</i> or <i>-n</i>	Set interactive trace counts
S	Scan TRACE (without execution)

Chapter 5: POI Command Processors

This section contains the following topics:

[List of POI Command Processors](#) (see page 690)

[Benefits of POI](#) (see page 692)

[OPSBROW Command Processor—View Automation Events](#) (see page 701)

[OPSCMD Command Processor—Issue z/OS, JES, VM, and IMS Commands](#) (see page 702)

[OPSDELV Command Processor—Delete Global Variables](#) (see page 717)

[OPSDOM Command Processor—Delete Operator Messages](#) (see page 720)

[OPSEXEC Command Processor—Run a Specified Program](#) (see page 722)

[OPSGETV Command Processor—Retrieve Global Variable Value](#) (see page 722)

[OPSGETVL Command Processor—Get Global Variable Names](#) (see page 726)

[OPSHFI Command Processor—Store Global Variable Values](#) (see page 732)

[OPSIMEX Command Processor—Executes a Member as a REXX Program](#) (see page 739)

[OPSPARM Command Processor](#) (see page 740)

[OPSQSQL Command Processor—Invoke SQL Statements](#) (see page 745)

[OPSREPLY Command Processor—Compare WTOR Messages](#) (see page 746)

[OPSREQ Command Processor—Invoke AOF Request Rules](#) (see page 755)

[OPSRMT Command Processor—Issue TSO Commands to Other Systems](#) (see page 758)

[OPSSMTBL Command Processor—Maintain the Directory Table](#) (see page 766)

[OPSVASRV Command Processor](#) (see page 773)

[OPSVIEW Command Processor—Access OPSVIEW](#) (see page 776)

[OPSWAIT Command Processor—Suspend CLIST or REXX Processing](#) (see page 777)

[OPSWTO Command Processor—Issue WTO and WTOR Messages](#) (see page 780)

[STATESET Program—Set State Values](#) (see page 788)

List of POI Command Processors

This section briefly describes each of the CA OPS/MVS command processors. It also includes a description of the STATESET REXX program, which is also explained in this chapter.

OPSBRW

Enables you to view CA OPS/MVS events

OPSCMD

Issues z/OS, JES, VM, and IMS commands from a CLIST, an OPS/REXX program, or a TSO terminal and receives output

OPSDLV

Deletes global status variables that match a variable name mask you specify

OPSDOM

Deletes a message from the console of the operator

OPSEEXEC

Enables you to run an OPS/REXX program explicitly

OPSGETV

Retrieves the value of a global variable, and either displays it at the terminal or stores it in a CLIST or REXX variable

OPSGETVL

Retrieves the names of global variables that match a variable name mask you specify, and either returns them to your terminal or stores them in CLIST or REXX variables

OPSHFI

Provides the capability to store values for global variables on an external VSAM key-sequenced data set; using OP SHFI, you may WRITE global variables to the VSAM file, READ variable records from the VSAM file, or DELETE variable records from the VSAM file

OPSIMEX

Enables you to run an OPS/REXX program implicitly.

OPSPARM

Displays, and enables you to change, the values of the CA OPS/MVS operating parameters

OPSQL

Invokes SQL statements from a TSO terminal, a TSO CLIST, or a TSO/E REXX program

OPSREPLY

Compares WTOR messages to a specified set of values and issues replies to messages containing matching values

OPSREQ

Invokes AOF request rules

OPSRMT

Enables you to issue TSO commands to other systems from either a TSO terminal or from a console logged onto the ECF component

OPSETV

Creates a global status variable or updates the value of a global status variable

OPSSMTBL

Enables you to maintain the directory table of the System State Manager

OPSVASRV

Manipulate CA CCS Sysplex variables

OPSVIEW

Provides access to OPSVIEW, which is the CA OPS/MVS full-screen, interactive application for controlling CA OPS/MVS itself, z/OS, and JES

OPSWAIT

Enables you to suspend processing of a CLIST or REXX program until a specified amount of time passes or a specific time is reached

OPSWTO

Issues one-line and multiline WTO and WTOR messages from an OPS/REXX program, a TSO/E REXX program or CLIST, or a TSO READY prompt

STATESET REXX program

Enables you to change the current state, desired state, and mode values specified for a resource; display the current state or desired state value of a resource; indicate whether CA OPS/MVS waits for the current state of a resource to equal its desired state

Benefits of POI

The Programmable Operations Interface (POI) is a set of TSO command processors that give you control of the CA OPS/MVS product and access to z/OS and subsystem functions.

Command processors can do the following:

- Issue the commands required to coordinate processing for batch jobs and the online systems they support.

For example, if you have a batch job requiring exclusive use of a CICS file, that job can include a job step that runs the TSO Terminal Monitor Program (TMP) in batch mode. This job step can use the OPSCMD command processor of the POI to issue CICS commands (through the z/OS MODIFY command) that take control of the needed file from CICS. After the job step executes, a subsequent job step can use the batch TMP again to return the file to CICS.

- Capture, in TSO CLISTs, command sequences that your system operators issue manually.

When you use the CA OPS/MVS Enhanced Console Facility to invoke the CLISTs, these command sequences run automatically.

- Be used in custom ISPF dialogs that guide help desk personnel or end users through complex operations procedures that might otherwise require help from senior programmers or systems analysts.

POI command processors can automate steps of these procedures that extract information from z/OS, issuing the commands needed to get the information and processing the command output in the ISPF dialog. People using these dialogs need to do nothing except respond to prompts for information that only they can supply.

Security Considerations for POI

The POI controls many areas of system operation, so you might want to restrict who can use it and what those users can do. You can maintain security for POI in any of these ways:

- Use RACF or CA ACF2 to restrict the use of the OPSRMT and OPSCMD command processors by user ID.
- Read-protect the library where the OPSCMD load modules reside.
- Use the OPUSEX exit routine to check the authority of users when they issue any POI command processor. All of the POI command processors call this exit, which is described in the chapter “Technical Notes” in the *Administrator Guide*.
- Write AOF security rules to check the authority of the users instead of using the OPUSEX exit routine.

Note: Security rules execute for all POI commands that are issued from an unsecured environment (for example, the TSO address space of a user).

Specify a Subsystem ID on a POI Command Processor

You can use the SUBSYS keyword available on almost all POI command processors to specify the subsystem ID of a particular copy of CA OPS/MVS on the local system, from where the command was issued, that should process the command. The SUBSYS keyword overrides the default subsystem ID, which is usually OPSS but may be changed as described below.

The only exception is the OPSWAIT command processor. Since execution of OPSWAIT does not require communication with a CA OPS/MVS subsystem, it does not support the SUBSYS keyword.

Change the Default Subsystem ID

The default subsystem ID for all CA OPS/MVS command processors is OPSS. Users can modify their own default subsystem IDs.

To change the default subsystem ID, allocate a ddname of the form OPSS\$OPSx (where x is any alphabetic character). The CA OPS/MVS product uses the last four characters of this ddname as the default subsystem ID for all CA OPS/MVS command processors (including the OPS/REXX command processors OI and OX).

Example: Change The Default Subsystem ID

TSO users might issue the following command to set the default subsystem ID for their session to OPSA:

```
ALLOC F(OPSS$OPSA) DUMMY
```

When CA OPS/MVS finds multiple DDs of this form, it uses the first one found in the TIOT.

Note: The order of DDs in the TIOT has no connection to the order in which the TSO ALLOCATE command allocated them.

Specify an MSF System ID on a POI Command Processor

You can use the SYSID or SYSTEM keywords available on many (but not all) POI command processors to specify the MSF system IDs of those copies of CA OPS/MVS to which the command should be routed. Routing is done by the subsystem on the local system, from where the command was issued, that was specified on the SUBSYS keyword (or OPSS by default). If not specified (or if the SYSID or SYSTEM keywords are not supported), then the command is routed to only the local system, from where the command was issued.

Except where noted for each command processor, the SYSID or SYSTEM keywords generally support these values:

Value	Description
*	Indicates that only the local system should receive the command. This is the default, and the same as not specifying the SYSID or SYSTEM keywords. When you are not using MSF, specify this value or no SYSID or SYSTEM keywords.
<i>msfids</i>	Indicates a list of from one to eight MSF system IDs (remote or local) that should receive the command.
ALL	Indicates that all active MSF-connected systems should receive the command, including the local system.

Value	Description
EXT	Indicates that all active MSF-connected systems other than the local system should receive the command.

How to Invoke POI Command Processors

You can invoke POI command processors as follows:

- By including them in a TSO CLIST or TSO/E REXX program.
- By issuing them in ADDRESS TSO host environment instructions in an OPS/REXX program. The TSO host environment must be present to execute a POI command in an OPS/REXX program.
- By including them in a UNIX REXX program running under UNIX System Services (USS).
- By issuing them as NetView commands or NetView REXX functions.
- By issuing them interactively in a user TSO session.
- By specifying them in a z/OS parameter string as input to a TSO/E CALL command outside of the TSO host environment.
- By specifying them as the PGM value on the EXEC statement in a batch job step with parameters specified as the PARM value. For an example, see Issuing POI Commands in Batch Mode in this chapter.
- By including them in the //SYSTSIN DD in a batch job step that is running the TSO TMP in batch mode. For an example, see Issuing POI Commands in Batch Mode in this chapter.

Issue POI Commands in Batch Mode

Use the following JCL to run a single POI command as a batch job step:

```
// EXEC PGM=command,PARM='command parameter'
```

Use the following JCL to issue multiple POI commands running under the TSO TMP as a batch job step:

```
// EXEC PGM=IKJEFT01,PARM='first TSO or POI command or CLIST to be executed'
// SYSPROC DD DISP=SHR,DSN=user.clistlib
// SYSEXEC DD DISP=SHR,DSN=user.rexxlib
// SYSTSPT DD SYSOUT=*
// SYSTSIN DD *
(TSO commands or CLISTS to be executed,
  *including *
POI commands to be executed,
  *including *
OX commands to invoke OPS/REXX programs from data sets,
OI commands to invoke OPS/REXX programs from //SYSEXEC)
/*
```

Abbreviations in Command Processor Text

In coded automation procedures, such as CLISTS or OPS/REXX programs, always specify the complete text of command processors and their keywords and operands. Doing so avoids syntax ambiguity in the future as we add new command processors to the CA OPS/MVS product.

If you are invoking a command processor online, as you would during an interactive TSO session, you can (in most cases) use the shortest unique abbreviations. Be aware that the shortest unique abbreviation of a command processor, keyword, or operand can change from release to release.

Using OPS/REXX Efficiently

When a particular operation (such as waiting) needs to be performed from an OPS/REXX program or rule, and the operation can be performed either by using a TSO command processor or by using an OPS/REXX function or host environment command, the OPS/REXX function or host environment command is always more efficient.

When the same operation is performed repeatedly, such as in a loop, the difference in CPU resources that are consumed can be significant.

For these reasons, we recommend that whenever possible the OPS/REXX function or host environment command (as shown in the following table) be used in place of the TSO command processor in an OPS/REXX program or REQ rule:

REXX Function/Host Command	TSO Command Processor
OPSDOM()	OPSDOM
OPSPRM()	OPSPARM
OPSVALUE()	OPSGETV, OPSGETVL, OPSSETV, OPSDELV
OPSWAIT()	OPSWAIT
ADDRESS OPER	OPSCMD, OC
ADDRESS SQL	OPSQL
ADDRESS WTO	OPSWTO

How to Use POI Command Processors

Use the following table to determine where and how to use POI command processors. Keep these guidelines in mind:

- All POI commands can run as stand-alone TSO command processors or TSO/E CLIST commands.
- In TSO/E REXX and OPS/REXX, all commands can run as ADDRESS TSO host commands. The TSO/E column in the table indicates what commands can also be used as REXX functions.
- The OPS/REXX Preferred Method column in the table shows the recommended host environment for invocation of the command, or the equivalent OPS/REXX built-in function.
- The NetView column in the table indicates what POI commands can execute as NetView commands or NetView REXX functions.
- The Batch column in the table indicates the ability of a command to run outside of TSO using a z/OS parameter string as input to the TSO/E CALL command or the following z/OS JCL statement:

```
// EXEC PGM=command,PARM='command parameter'
```

POI Command	TSO/E	OPS/REXX Preferred Method	NetView	Batch
OPSBRW	Command processor requires ISPF (works with ISPF only)	OPSLOG(...) function: Note that the OPSLOG function lets you retrieve data from the OPSLOG, it is not equivalent to the OPSLOG command processor.	No	No
OPSCMD	Command processor	ADDRESS OPER host environment	No	Yes
OPSDLV	Command processor or REXX function	OPSVALUE(...) function	Yes	Yes
OPSDOM	Command processor	OPSDOM(...) function	No	Yes
OPSEEXEC	Command processor	N/A	No	Yes
OPSGETV	Command processor or REXX function	OPSVALUE(...) function	Yes	Yes
OPSGETVL	Command processor or REXX function	OPSVALUE(...) function	Yes	Yes
OPSHFI	Command processor or REXX function	OPSHFI(...) function	No	Yes
OPSIMEX	Command processor	N/A	No	Yes
OPSPARM	Command processor	OPSPRM(...) function	No	Yes

POI Command	TSO/E	OPS/REXX Preferred Method	NetView	Batch
OPSQL	Command processor or REXX function	ADDRESS SQL host environment	Yes	Yes
OPSREPLY	Command processor	N/A	No	Yes
OPSREQ	Command processor	N/A	No	Yes
OPSRMT	Command processor	N/A	No	Yes
OPSETV	Command processor or REXX function	OPVALUE(...) function	Yes	Yes
OPSSMTBL	Command processor or REXX function	OPSSMTBL(...) function	Yes	Yes
OPSVIEW	Command processor requires ISPF (works with ISPF only)	N/A	No	No
OPSWAIT	Command processor or REXX function	OPSWAIT(...) function	No	Yes
OPSWTO	Command processor	ADDRESS WTO host environment	No	Yes

Run POI Commands in UNIX

UNIX System Services (USS) allows the TSO/E REXX language to be used in the UNIX environment to execute shell commands and other USS callable services. The function of such a REXX program is similar to that of a UNIX shell script. For a complete description of the capabilities of UNIX REXX, see the IBM documentation.

In UNIX REXX, a limited number of CA OPS/MVS POI commands can be used. Only those POI commands that are documented as also being callable as TSO/E REXX functions will function properly in UNIX REXX. These POI commands include OPSDELV, OPSGETV, OPSGETVL, OPSHFI, OPSQL, OPSETV, OPSSMTBL, and OPSWAIT. The CMDRESP(TERMINAL) option available with some of these commands results in WTO message output and consequently should not be used. Since the UNIX REXX environment is not a TSO environment, security rules for protected resources such as SQL and OPSGLOBAL must be enabled to allow access to these resources, as in z/OS batch mode.

The UNIX shell provides a 'tso' command for executing TSO commands from the UNIX environment. From an OMVS TSO session, the command is executed in the address space of a TSO user. From a non-TSO UNIX environment, a temporary TSO TMP is created to execute the command.

If you want to execute a POI command that does not run as a REXX function or if you want the OI command to execute an OPS/REXX program, use the TSO shell command. The following UNIX REXX statement invokes the STATESET OPS/REXX program:

```
Address SH "tso 'OI STATESET STCTBL.CICSUSS CURRENT(DOWN)'"
```

The same OPS/REXX program could be invoked directly from the shell outside of a UNIX REXX program with the command:

```
tso 'OI STATESET STCTBL.CICSUSS CURRENT(DOWN)'
```

Note: For running the OI command as a shell command without TSO, see the chapter “Using OPS/REXX” in the *User Guide*.

When a TSO TMP is created to execute the OI command, the SYSEXEC ddname must be allocated to the appropriate data sets. This can be accomplished by setting the environment variable SYSEXEC to the list of data set names required before the tso shell command is issued. The following command concatenates two data sets to the SYSEXEC ddname:

```
export SYSEXEC=MYOWNREXX:SYS1.OPS.CCLXEXEC
```

A generalized mechanism for allocating any required data sets for the created TSO TMP is described in the IBM documentation.

In all the above scenarios, the assumption was made that the CA OPS/MVS commands were in the system linklist or LPA. If this is not the case, the environment variable STEPLIB must include the CA OPS/MVS load library in the steplib concatenation. The following shell command updates or creates the STEPLIB variable:

```
export STEPLIB=$STEPLIB:SYS1.OPS.CCLXLOAD
```


OPSBRW Command Processor—View Automation Events

The OPSBRW command processor enables you to view automation events. To display these events, OPSBRW invokes the same program as OPSVIEW option 1 (OPSLOG).

Abbreviation: OB

This command processor has the following format:

```
OPSBRW  
[LOGNAME(logname)]  
[SUBSYS(ssid)]  
[SYSTEM|SYSID(msfid)]
```

LOGNAME

(Optional) Defines the one- to sixteen-character log name of the OPSLOG that you wish to browse.

SUBSYS

(Optional) For more information on the SUBSYS keyword, see Specifying a Subsystem ID on a POI Command Processor in this chapter.

SYSTEM or SYSID

(Optional) You may specify a single system only. For more information on the SYSTEM or SYSID keywords, see Specifying an MSF System ID on a POI Command Processor in this chapter.

OPSCMD Command Processor—Issue z/OS, JES, VM, and IMS Commands

The OPSCMD command processor issues z/OS, JES, VM, and IMS commands from a CLIST, an OPS/REXX program, a TSO/E REXX program, or a TSO terminal and receives the command output.

Note: You can use OPSCMD to issue IMS commands only if your site has licensed the CA OPS/MVS optional IMS Operations Facility.

Use either of the following formats to issue the OPSCMD command processor.

Note: Format 2 for the OPSCMD command processor supports most of the same keywords that the ADDRESS OPER host environment supports. If you are unsure of the meaning of any of these keywords, see the description of the ADDRESS OPER command keywords in the chapter "[Host Environment Commands](#)" (see page 23)."

1. Use this format for the OPSCMD command processor when you do not want to specify any additional keywords:

`OPSCMD commandtext`

2. Use the following format to specify additional keywords for the OPSCMD command processor:

Note: The REPLY keyword on the CMDRESP keyword is optional.

```
OPSCMD COMMAND(text) BMFCMDOUT(OPSLOG|WTO|NONE)
      CAPTURE(msgtextlist)
      CMDECHO(Y|ES|NO)
      CMDLOG(Y|ES|NO)
      CMDRESP(destination,[REPLY])
      CMDWAIT(seconds)
      CONTYPE(A|N|Y|EXTCONS|SSCONS|XTRA CONS)
      DELAY(seconds)
      MSID(name["|"list])
      MSPLEX(name["|"list])
      MSREPLY
      INTERVAL(ocinterval)
      LOG(Y|ES|NO|OFF)
      MAXCMDOUT(number)
      MFORM(M|J|S)
      NAME[CONNAME(consolename)]
      NOCLIST
      OUTDELIM(delimstring,Y|ES|NO)
      OUTPUT|NOOUTPUT
      PREFIX(CMDOUT,prefix)
      STOPEND(Y|ES|NO)
      STOPMSG(msgtextlist)
      STOPRESP(msgtextlist)
```

```
SUBSYS(OPSS,ssid)
SYSQSYSTEM(msids|ALL|EXT)
SYSWAIT(seconds)
WAIT(seconds)
XML(YES|NO)
```

commandtext

The *commandtext* operand is the text of the command for OPSCMD. This text is totally freeform, and CA OPS/MVS does not convert it to uppercase characters.

COMMAND

The **COMMAND** keyword supplies the text of the system command you are issuing.

The **COMMAND** keyword has no default value. If you omit this keyword but specify other keywords, CA OPS/MVS places you in OPSCMD subcommand mode.

If the command text contains unmatched parentheses, you must follow conventional TSO syntax and enclose the text in quotation marks.

If you use non-alphabetic characters as command control characters, you might need to enclose the command in quotation marks.

You can invoke OPSCMD without keywords so long as the first word of the command string does not conflict with any keywords or their operands.

Example: Specify command text. In the example, quotes are used to delimit command text.

```
OPSCMD COMMAND(')testrex')
```

CMDRESP

This keyword determines where the output from any variation of OPSCMD goes. Specify any of the following as the destination of output:

TERMINAL

Displays the command output as lines on a TSO terminal. CMDRESP(TERMINAL) is the default.

CLIST

Generates an array of TSO CLIST variables. The value of the PREFIX keyword provides the prefix for these variables.

REXX

Generates a series of REXX stem variables. The value of the PREFIX keyword is the stem for these variables.

NOWHERE

Suppresses the return of any command output, either as variables or as display lines.

Optionally, you can specify a second positional parameter, called REPLY, for the CMDRESP keyword. Specifying REPLY causes the response from the REPLY command to be captured. Typically, the response is the REPLY echo and z/OS message IEE600I, but the response can include more if the issuer of the WTOR message requests it (for example, if IMS is the issuer).

The POI ignores invalid values for CMDRESP. Even usually valid values such as REXX and CLIST can be invalid in certain contexts. For instance, if you specify CMDRESP(CLIST) on an OPSCMD that is not issued in a CLIST.

When OPSCMD executes in an OPS/REXX program, command output lines are written to the external data queue only; no stem variables are generated even if you specify CMDRESP(REXX).

When OPSCMD executes in a TSO/E REXX program, command output lines are never written to the external data queue.

When you specify either CMDRESP(CLIST) or CMDRESP(REXX) for OPSCMD, no &SYSOUTLINE variables are generated.

INTERVAL

(Optional) This keyword specifies, in centiseconds, how frequently OPSCMD tests for command response lines to see if the response has ended. The INTERVAL value temporarily overrides the value of the CA OPS/MVS OCINTERVAL parameter.

The default for INTERVAL is the value of the OCINTERVAL parameter, but you can specify any number between 10 and 300, or a value of 0 to bypass interval testing.

OPSCMD Security With Audit Ability

Because you can use OPSCMD to enter commands that can crash JES, z/OS, or VM, we have built security measures into OPSCMD. The default security authorization exit for OPSCMD issues an error message and terminates if you do not have OPER authority. OPSCMD also terminates if you specify any of the following commands:

Type of Command	Command
IMS	/CHE
JES3	*DUMP, *RETURN
MVS	QUIESCE
VM	CP, #CP, I, IPL, LOG, LOGOFF, LOGOUT, SHUTDOWN, SYS, SYSTEM

All commands issued through OPSCMD go into the system log, prefaced with message ID OPS1181H.

Abbreviation: OC is the abbreviation of OPSCMD.

Enter Subcommand Mode

From an interactive TSO session only, you can enter OPSCMD subcommand mode by entering the text OPSCMD by itself as shown here:

```
OPSCMD
```

To receive the command output, OPSCMD subcommand mode prompts you to enter either system commands or the END command. For more information, see Using OPSCMD Subcommand Mode in this chapter.

Guidelines for Issuing OPSCMD

When you are issuing OPSCMD in automation procedures, follow these guidelines:

- In coded automation procedures, such as CLISTs or OPS/REXX programs, always specify the complete text of command processors and their keywords and operands. Doing so avoids syntax ambiguity in the future as we add new command processors to CA OPS/MVS.
- The length of all operands for the CAPTURE, STOPMSG, and STOPRESP keywords, added together, cannot exceed 1240 bytes.

- Since the CA OPS/MVS MSF component sends only 320 bytes of OPSCMD text to a remote system, you should limit the length of CAPTURE, STOPMSG, and STOPRESP operands accordingly.
- OPSCMD does not support the command stacking delimiter.

Output Variables Generated by OPSCMD

When you use OPSCMD in a CLIST or REXX program, you can capture the response to the issued command in REXX or CLIST variables. The program can then analyze the response and take the appropriate action.

The following list shows the output variables that OPSCMD creates. The value *prefix* is the value specified with the PREFIX keyword.

- Contains the number of lines returned in the command response.
Variable Name if CMDRESP(CLIST): *prefix* and *prefix0*
Variable Name if CMDRESP(REXX): *prefix.0*
- Contains the text of line *n* of the command response.
Variable Name if CMDRESP(CLIST): *prefixn*
Variable Name if CMDRESP(REXX): *prefix.n*
The *n* value is an integer corresponding to the output line number.
- Contains the number of words in line *n* of the command response.
Variable Name if CMDRESP(CLIST): *prefixnW*
Variable Name if CMDRESP(REXX): *prefix.n.0*
- Contains the *m*th word in line *n* of the command response.
Variable Name if CMDRESP(CLIST): *prefixnWm*
Variable Name if CMDRESP(REXX): *prefix.n.m*
The *m* is a number indicating the position of the word in output line *n*.
- Contains the return code from the command.
Variable Name if CMDRESP(CLIST): *prefixRC*
Variable Name if CMDRESP(REXX): *prefixRC*
- Contains the name of the console associated with the command.
Variable Name if CMDRESP(CLIST): *prefixCONNAME*
Variable Name if CMDRESP(REXX): *prefixCONNAME*

IMS Type 2 Message Considerations

A subset of the IMS Type 2 message facility implemented in the ADDRESS OPER host environment is available to the OPSCMD command processor. The keywords associated with issuing IMS Type 2 commands, IMSID, IMSPLEX, WAIT, and XML, are all described under that section, as well as return and reason code interpretation. You can direct output from a Type 2 command to the terminal or the EDQ.

In order to issue IMS Type 2 commands, access must be available to IBM modules CSLSRG00 and CSLSDR00. Those are shipped in IMS product libraries at release 9.1 or greater, and are usually named *h/q.SDFSRESL*. The entire IMS product library can be allocated, or a private data set with just those isolated modules, and may be an explicit allocation, or a LNKLIST entry.

Command Characters for Subsystem Commands

z/OS supports the use of special command characters to identify the subsystem to which a command entered from the console is directed. The following is a list of commonly used command characters. When you use OPSCMD to issue the command, you must prefix the command with the appropriate subsystem command character to address a command to one of the subsystems in this list.

\$

JES2 command.

*** or 8**

JES3 command. While you can omit the command character when issuing a JES3 command from a JES3 console, you must include the character when issuing a JES3 command through OPSCMD.

-

DB2 command. You can change the DB2 command character by modifying its entry in the appropriate IEFSSNxx member of the Logical ParmLib Concatenation.

OPSCMD Command Characters for IMS Operation

The CA OPS/MVS IMS Operation Facility supports a table of command characters that imply the specific control region for which a command is intended. Defining and using these command characters enables you to omit the IMSID keyword on the OPSCMD, and thus use the simpler form of OPSCMD syntax.

For information on defining IMS command characters, see the description of the `IMSnCHAR` parameter in the *Parameter Reference*.

A single copy of CA OPS/MVS can manage up to 32 separate copies of IMS. For a list of the versions of IMS that the CA OPS/MVS product supports, see the *Installation Guide*. If you run more than one IMS under the control of CA OPS/MVS, they can be at any combination of supported IMS levels.

Issue VM Commands Through OPSCMD

The prefix `#CP` must precede the text of any VM command. OPSCMD strips the `#CP` prefix out and passes the rest of the command text to VM for execution. OPSCMD can issue only one VM command at a time.

Most VM, JES, or z/OS commands that OPSCMD issues return output, except for a few commands, for example, z/OS `REPLY`. Even if a command returns no output, it is always executed by z/OS, JES, or VM.

Issue the OPSCMD from CLISTs

In most cases, you issue OPSCMD from a CLIST to capture the output of system commands in variables so the CLIST can then act on that output. However, you can also direct the command output to a TSO terminal.

Note: If you are using OPSCMD from a CLIST and you turn on CLIST debugging options, you will probably receive a bogus first line of output (reflecting the command you entered), with all the correct output lines starting with the second line returned through the `SYSOUTTRAP` feature. If your CLIST uses debugging options, your code must allow for this extra output line. OPS/REXX `TRACE` options do not introduce extra lines into the external data queue.

Examples: Issue the OPSCMD from CLISTs

The following OPSCMD examples show how to issue the same command with different abbreviations for OPSCMD:

- To execute a z/OS display command, issue one of the following commands:

```
OPSCMD d a,l
```

```
OC d a,l
```

- To execute a JES3 display command, issue the following command:

```
OPSCMD *i a
```

- To execute a VM query command, issue the following command:

```
OPSCMD #cp q n
```

- To execute a z/OS command, issue one of the following commands:

```
OPSCMD COMMAND(d r,l)
```

```
OPSCMD command(D r,l)
```

Use OPSCMD Subcommand Mode

The CA OPS/MVS OPSCMD subcommand mode provides an easy way to enter several operating system commands.

To enter OPSCMD subcommand mode

1. Issue the OPSCMD command without supplying a system command or any keywords as follows:

```
OPSCMD
```

In response, you see prompts of OC, which indicate that the system is ready.

2. You can either enter system commands or enter the END command to exit OPSCMD subcommand mode.

You can enter one or more system commands after each prompt. If you enter system commands, they execute and you receive their output. Entering END terminates OPSCMD subcommand mode.

You can also enter OPSCMD subcommand mode by issuing the OPSCMD command with operands, if you omit the COMMAND keyword. For example, all of the following commands start OPSCMD subcommand mode. The last two commands invoke OPSCMD subcommand mode because they specify no command text.

```
OPSCMD
```

```
OC
```

```
OPSCMD NOO
```

```
OC WAIT(15)
```

Example: Use OPSCMD Subcommand Mode

The following illustration shows you sample interaction between the CA OPS/MVS product and a user in OPSCMD subcommand mode. Items shown in bold are what appear on the screen; the other text describes what each item is.

```
READY          - READY prompt from TSO
OC            - You enter OC to invoke OPSCMD subcommand mode
OC            - Prompt from OPSCMD
D R,L         - z/OS command you enter
(output from D R,L command...)
OC            - Prompt from OPSCMD
#CP Q N       - VM command you enter
(output from Q N command...)
OC            - Prompt from OPSCMD
*I B         - JES3 command you enter
(output from *I B command...)
OC            - Prompt from OPSCMD
(blank line you enter)
OC            - Prompt from OPSCMD
END          - You enter END to cancel OPSCMD subcommand mode
READY        - READY prompt from TSO
```

Use OPSCMD with z/OS REPLY Command

You can use OPSCMD to reply to any WTOR message. OPSCMD checks each verb in the system command you specify to see if that verb is R or REPLY; if so, OPSCMD issues the command using the z/OS master console. The only output from such commands is a message stating that there is no z/OS response to a reply.

The length of the reply text specified through OPSCMD must be less than or equal to the maximum reply length. You receive no error message if you specify reply text that is too long. For example, consider the following:

```
opscmd command(reply 29,n)
opscmd r 35,n
oc reply 99,y'
```

You can obtain output from REPLY commands by using the CMDRESP(REPLY) keyword to override the usual NOOUTPUT default. A subsystem or extended console is used to issue the REPLY command.

How OPSCMD Retrieves Command Output

Where you receive output from a command issued by OPSCMD depends on how you issued OPSCMD:

If OPSCMD is issued from...	The command output...
A TSO terminal	Is displayed on the terminal
A CLIST	Goes into CLIST variables if SYSOUTTRAP is active
An OPS/REXX program (through an ADDRESS TSO instruction)	Goes to the OPS/REXX external data queue

OPSCMD internally determines when it has received the last line of command output. If you do not receive all of the output you expect, specify the WAIT keyword with OPSCMD or increase the specified wait time.

OPSCMD receives z/OS or JES2 command output only if the CA OPS/MVS address space is active. If CA OPS/MVS is not running, the system command you specified executes, but OPSCMD generates a non-zero return code and issues the following message:

```
COMMAND OUTPUT RETRIEVAL NOT OPERATIONAL
```

You can always receive VM command output if z/OS runs under VM, even if the CA OPS/MVS address space is not running. The CA OPS/MVS product can retrieve up to 4 KB of output from VM, the maximum amount of contiguous real storage that VM can obtain from z/OS. If a VM command produces more than 4 KB of output, OPSCMD returns the first 4 KB of output and this message:

```
VM COMMAND OUTPUT BUFFER OVERFLOW
```

z/OS and JES2 commands can return up to 2000 lines of output. You can increase this value either by using the OPSPRM REXX function to alter the OCMAXMSG parameter value, or by using the MAXCMDOUT keyword.

You receive the following error message if the CA OPS/MVS address space is active but a command issued through OPSCMD produces no output:

```
{IMS | VM | JES} COMMAND GENERATED NO OUTPUT
```

CLIST Variables That TSO/E Creates

Under TSO, if the variable &SYSOUTTRAP has a value greater than zero, the first *n* lines of command output (where *n* is the value of &SYSOUTTRAP) go into CLIST variables called &SYSOUTLINE*x* (*x* is the line number).

Note: These variables are *not* created if the OPSCMD command processor includes the CMDRESP(REXX) or CMDRESP(CLIST) keyword. For more information, see CMDRESP Keyword in this chapter.

Typical &SYSOUTTRAP values range from 1 to several thousand. When TSO/E captures command output, it also sets the variable &SYSOUTLINE to the number of &SYSOUTLINE*x* variables it created. You must do the following:

- Copy the value of &SYSOUTLINE into a separate CLIST variable before your CLIST does any further processing. You *must* copy the &SYSOUTLINE value because the next TSO command in the CLIST will reset it.
- Set &SYSOUTTRAP to zero immediately so that output from a subsequent command does not overlay the CLIST variables just created.

CLIST variables that OPSCMD creates exist until you delete them. Thus, issuing an OPSCMD that generates 100 lines of output creates variables &SYSOUTLINE1 through &SYSOUTLINE100. If you issue another OPSCMD that produces 10 output lines, &SYSOUTLINE1 through &SYSOUTLINE10 contain the new output lines, while &SYSOUTLINE11 through &SYSOUTLINE100 contain the rest of the output of the previous command.

CLIST Variables That OPSCMD Creates

If you issue OPSCMD without the CMDRESP(REXX), CMDRESP(CLIST), or NOCLIST keywords *and* &SYSOUTTRAP has a value other than zero, OPSCMD creates additional CLIST variables from system command output. These variables contain the number of tokens found in each line of system command output and the value of each token.

Note: Tokens are strings enclosed in delimiters. The CA OPS/MVS product recognizes blanks and the following characters as delimiters: () / = , . All delimiters except the blank are also tokens and have CLIST variables created for them. Each use of any delimiter, except the blank, counts as one delimiter, but any number of blanks in one row counts as a single delimiter.

The CLIST variables created have these names:

Name	Description
&SYSOUTLINE <i>n</i> W	Contains the number of tokens in line <i>n</i> . <i>n</i> is the current output line number without any leading zeroes or blanks.

Name	Description
&SYSOUTLINE <i>nWm</i>	Contains the value of the <i>m</i> th token on line <i>n</i> . <i>m</i> is a sequential token number (starting with 1 and increased incrementally by 1) and <i>n</i> is the current output line number.

OPSCMD Return Codes

OPSCMD generates the following return codes:

Return Code	Message Number	Message Text
0	OPS1100	COMMAND EXECUTION SUCCESSFUL
0	n/a	NO REPLY COMMAND OUTPUT
4	OPS1101	COMMAND GENERATED NO OUTPUT
8	OPS1102	OUTPUT RETRIEVAL MECHANISM DOWN
12	OPS1103	NO MVS/JES/VM COMMAND ENTERED
16	OPS1104	MVS/JES/VM CMD LENGTH EXCEEDS MAXIMUM
20	OPS1105	MVS/JES/VM CMD OUTPUT BUF OVERFLOW
24	OPS1106	VM COMMAND INVALID - MVS NOT UNDER VM
28	OPS1107	AUTHORIZATION CHECK FAILED
32	OPS1108	NO JES3 COMMAND OUTPUT ON JES3 LOCALS
36	OPS1109	INVALID KEYWORD COMBINATION ENTERED
40	OPS1110	INVALID LOCAL/REMOTE SYSTEM ID
44	OPS1111	SYSTEM ID IS NOT ACTIVE
48	OPS1112	COMMANDS DISABLED AT THIS TIME
52	n/a	SOME TYPE OF SYSTEM NOT ACTIVE
56	n/a	SOME TYPE OF THING NOT FOUND
60	OPS1115	SOME TYPE OF ERROR OCCURRED
64	OPS1116	PRODUCT MUST BE ACTIVE TO USE COMMAND
88	OPS1122	TSO/E IS NOT INSTALLED
92	OPS1123	COMMAND NOT ISSUED WITH AUTHORIZATION
96	OPS1124	NO CNSLS AVAILABLE FOR CMD OUT RETRIEVAL
100	OPS1125	NO CONSOLES OF THE CORRECT TYPE EXIST

Return Code	Message Number	Message Text
104	OPS1126	MAIN PRODUCT ADDRESS SPACE TERMINATED
108	OPS1127	MULTI-SYSTEM FACILITY NOT INSTALLED
112	OPS1128	MULTI-SYSTEM FACILITY NOT ACTIVE
120	OPS1130	MULTIPLE IMS IDS MISSING IMSPLEX KEYWORD
124	OPS1131	HOST SYSTEM INACTIVE
128	OPS1132	CONSOLE CONTROL BLOCK TAG CHECK
132	OPS1133	MESSAGE QUEUE SEND ERROR
136	OPS1134	MESSAGE QUEUE RECEIVE ERROR
160	OPS1140	CONSOLE CONTROL BLOCK ENABLE FAILED
164	OPS1141	CONSOLE BLOCK RELEASE LOGIC ERROR
172	OPS1143	COMMAND FUNCTION AREA GET/FREE FAILED
176	OPS1144	COMMAND OUTPUT BUFFER GET/FREE FAILED
180	OPS1145	COMMAND OUTPUT QUEUE GET/FREE FAILED
184	OPS1146	ABEND FAILURE IN USER EXIT
186	OPS1146	ABEND FAILURE IN FUNCTION ROUTINE
188	OPS1147	MGCR (SVC 34) FAILURE
189	OPS1147	MGCRC (SVC 34) FAILURE
190	OPS1153	CONVCON FAILURE
200	OPS1150	COMMAND SCAN AREA ALLOCATION FAILED
204	OPS1151	COMMAND SCAN AREA RELEASE FAILED
216	OPS1154	INVALID COMMAND BUFFER FORMAT
220	OPS1155	MASTER CONTROL BLOCK ERROR
224	OPS1156	IKJSCAN ERROR
240	OPS1160	CLIST VARIABLE ACCESS ERROR
244	OPS1161	WORD TOKENIZATION ERROR
248	OPS1162	SYSOUTTRAP CONVERSION ERROR
252	OPS1164	COMMAND BUFFER PARSE ERROR
253	OPS1183	KEYWORD NOT SUPPORTED IN CURRENT MVS LEVEL
254	OPS1184	INVALID CONSOLE ID SPECIFIED

Return Code	Message Number	Message Text
255	OPS1153	NAME AND ID ARE MUTUALLY EXCLUSIVE
256	OPS4363	KEYWORD NOT VALID IN THIS ENVIRONMENT

Type 2 messages and commands utilize a specialized API to communicate with the IMSPLEX manager, and therefore are exposed to a new series of return and reason codes unique to conditions within the API.

Return and reason codes generated by this API have a distinctive format, where both return and reason codes are represented as:

Xnnnnnnnn'

For example, X'01000010'/X'00004004' is defined in the IBM documentation as CSLSRG00 could not be loaded, which indicates that the two required IMS modules are not available to CA OPS/MVS processing.

Note: The new series of type 2 return and reason codes are produced by IBM code, and documented in the IBM manual *IMS V10 System Programming API Reference* manual number SC18-9967-00, which is applicable to IMS version 10. Errors associated with API registration can be found under the section CSLSCREG Return and Reason Codes, and those with API dialog under CSLOMI Return and Reason Codes.

OPSDELV Command Processor—Delete Global Variables

The OPSDELV command processor deletes global variables whose names match a name mask that you specify and returns the count of the deleted variables.

Note: As a result of OPSDELV, both the global variable values and the names of the global variables are deleted.

Unlike the OPSVALUE function used with the R (remove) action code, the OPSDELV command processor does not remove subnodes of the variable names. Only those variables whose names match the value of *namemask* are removed.

In addition to invoking OPSDELV as a TSO command, you can invoke it as an OPS/REXX or TSO/E REXX function. The function will have a single argument equal to the OPSDELV TSO command syntax. The value returned by the function will be the count of deleted variables. The CA OPS/MVS product will set the OPSRC variable to the same return code that would have been generated if you had invoked OPSDELV as a TSO command.

This command has the following format:

```
OPSDELV {namemask}  
[CMDRESP(TERMINAL|REXX|CLIST|NOWHERE|XDQ)]  
[DELAY(0,seconds)]  
[SUBSYS(OPSS,ssid)]  
[SYSTEM(msfids|ALL|EXT)]  
[SYSWAIT(seconds)]  
[TOKEN]
```

namemask

For the value of *namemask*, specify either a complete variable name or a variable name that includes one or both of the special characters described here. The CA OPS/MVS product searches through the global variables on the specified system and tries to find any names that match *namemask*.

The *namemask* can be up to the maximum global variable size in length, and is automatically converted to uppercase. You can use two special characters in the *namemask*:

- + - This wildcard character matches any single character in a variable name.
- * - Used as the last character in the *namemask*, this wildcard character matches any number of trailing characters in a variable name.

The *namemask* that you specify can be either of the following:

- A valid CA OPS/MVS global variable name beginning with a valid global variable stem prefix (GLOBAL, GLOBALx, GLVTEMPx, or GLVJOBID).
- An Automate-format variable name up to 32 characters in length. CA OPS/MVS will look at the values of the ATMLOCALSCOPE nn and ATMSHAREDSCOPE nn parameters to determine the appropriate global variable stem prefix and will add it internally.

Guidelines for specifying *namemask*:

- If your name mask is for CA OPS/MVS global variable names, do not include wildcard characters in the first stem of the *namemask* value.
- Name masks for Automate-format variable names can include wildcard characters anywhere in the mask, since CA OPS/MVS adds the appropriate global or shared stem prefix to the mask internally.

Examples: *namemask*

GLOBAL0.ABCD*

Matches all CA OPS/MVS global variable names with twelve or more characters that begin with GLOBAL0.ABCD, such as GLOBAL0.ABCDEF.

TAPE_+++_STATUS

Matches all Automate-format variable names with fifteen characters, where the sixth through eighth characters can be any character and the other characters are equal to those in the mask, such as TAPE_280_STATUS.

+++

Matches all Automate-format variable names composed of three characters.

+++*

Matches all Automate-format variable names composed of three or more characters.

GLVTEMP1.*

Matches all CA OPS/MVS global variable names that begin with the stem GLVTEMP1.

CMDRESP

(Optional) Indicates how CA OPS/MVS should handle the response from OPSDELV. Values are:

TERMINAL

Displays the response at the terminal.

REXX

Sets the variable OPSDLCNT to the number of variables deleted by the command.

CLIST

Sets the variable OPSDLCNT to the number of variables deleted by the command.

NOWHERE

Tells CA OPS/MVS that it should not issue any messages except error messages. This option is provided to eliminate messages in REXX programs.

XDQ

Places the messages that are normally written to the terminal into the REXX external data queue.

When you invoke your OPSDELV request as a TSO command, the default of CMDRESP is CMDRESP(TERMINAL).

When you invoke your OPSDELV request as an OPS/REXX or TSO/E REXX function, the default of CMDRESP is CMDRESP(NOWHERE).

DELAY

(Optional) Indicates the number of seconds that your OPSDELV request waits before it can be processed. The default is 0. You can specify any number of seconds from 1 to 300.

SUBSYS

(Optional) For more information on the SUBSYS keyword, see Specifying a Subsystem ID on a POI Command Processor in this chapter.

SYSTEM

(Optional) For more information on the SYSTEM keyword, see Specifying an MSF System ID on a POI Command Processor in this chapter.

Note: If more than one system is specified or implied, and the local system is one of the systems that will receive the request, then CA OPS/MVS returns only the count for the local system.

SYSWAIT

(Optional) For a cross-system request, the SYSWAIT keyword specifies the number of seconds that CA OPS/MVS should wait for the remote system to receive and respond to the OPSDELV request. The default is the value of the MSFSYSWAIT parameter; you can specify any number of seconds from 1 to 600.

TOKEN

(Optional) Enables you to delete a single variable if the variable has not been updated since the last time it was retrieved with an OPSGETV command and the TOKEN keyword. The value specified for the TOKEN keyword is compared to the current update count for the variable. If the values match, the variable is deleted. If the values do not match, the requested deletion is rejected.

Examples: OPSDELV

This example finds all variables with names that match the mask, deletes them, and displays the number of deleted variables at the terminal:

```
OPSDELV GLOBAL0.XYZ+ABC*
```

This REXX function finds all global variables with names that match the mask and deletes them. Since `SYSTEM(ALL)` is specified, CA OPS/MVS sends the request to all systems that are connected through the MSF or CCI (CAICCI) communications protocols. The *cnt* variable is set to the number of variables deleted on the local system. The `OPSRC` variable is set to the same return code that would have been set if you had invoked `OPSDELV` as a TSO command:

```
cnt = OPSDELV ('GLOBAL0.XYZ* SYSTEM(ALL)')
```

OPSDELV Return Codes

OPSDELV produces these return codes:

Return Code	Description
0	The command executed successfully.
4	The command failed due to errors.
12	The CA OPS/MVS product is inactive.
40	You invoked the OPSDELV request as a REXX function, and it included an invalid REXX function argument.

OPSDOM Command Processor—Delete Operator Messages

The OPSDOM command processor deletes a message from the console of the operator.

This command has the following format:

```
OPSDOM  
{AMRFID(amrfid)}[DOMID(domid)][TOKEN(tokenid)]  
[DELAY(seconds)]  
[SUBSYS(ssid)]
```

AMRFID(*amrfid*)

Specifies the Action Message Retention Facility (AMRF) ID of the message, as displayed on the z/OS command D R,L. The value of *amrfid* must be a numeric value from 1 to 100000.

DOMID(*domid*)

Specifies the WTO IDs that z/OS returned when the original message was issued. You can specify a list of up to 10 DOM IDs.

TOKEN(*tokenid*)

Specifies the value of the TOKEN operand used on a previous WTO or WTOR. If you mark a group of messages using the TOKEN operand, you can delete all messages at once through this keyword. Specify *tokenid* either as a four-byte character string or as a hexadecimal string in the form: X'nnnnnnnn'.

Note: This keyword must be issued under the same address space as the message that is being deleted. Therefore, the utility of this keyword is limited.

DELAY

(Optional) Specifies a number of seconds, from 1 to 300, for CA OPS/MVS to wait before issuing the DOM. You can use this keyword to avoid a problem in situations in which a WTO message has not been issued or has not finished processing before a corresponding DOM request is issued.

SUBSYS

(Optional) For more information on the SUBSYS keyword, see Specifying a Subsystem ID on a POI Command Processor in this chapter.

Return Codes

OPSDOM produces these return codes:

Return Code	Message Number	Message Text
0	OPS1229	DOM COMPLETED NORMALLY
12	OPS1225	NO CORRESPONDING MESSAGE FOUND
16	OPS1226	AUTHORIZATION CHECK FAILED
84	n/a	COMMAND BUFFER PARSE ERROR
88	OPS1231	COMMAND PROCESSOR NOT AUTHORIZED
92	OPS1235	AUTHORIZATION USER EXIT ABENDED
100	OPS1237	SERIOUS SYSTEM MACRO ERROR
104	OPS1238	SERIOUS CONTROL BLOCK ERROR
108	OPS1230	INVALID COMMAND BUFFER FORMAT

OPSEEXEC Command Processor—Run a Specified Program

The OPSEEXEC (OX) command processor causes an explicitly-specified program to run. The function of OPSEEXEC is similar to that of OPSIMEX, except that OPSEEXEC requires that you supply the full command path. OPSEEXEC can execute a command that is outside the command concatenation.

OPSGETV Command Processor—Retrieve Global Variable Value

The OPSGETV command processor retrieves the value of a global variable that you specify. Depending on the keywords that you specify for your OPSGETV request, CA OPS/MVS either displays the value of the variable on your terminal or stores it in a CLIST or REXX variable.

You can retrieve the entire value of the variable, or only a substring portion of the value.

Note: If the variable that you specify does not exist, CA OPS/MVS returns a null value and no CLIST or REXX variable is created.

In addition to invoking OPSGETV as a TSO command, you can invoke it as an OPS/REXX or TSO/E REXX function. The function will have a single argument equal to the OPSGETV TSO command syntax. The value returned by the function will be the specified value of the variable. The CA OPS/MVS product sets the OPSRC variable to the same return code that would have been generated if you had invoked OPSGETV as a TSO command.

To invoke OPSGETV as a REXX function, use the following format:

```
OPSGETV
{varname}
[outputvar(ATMSV)]
[CMDRESP(TERMINAL|REXX|CLIST|NOWHERE)]
[DELAY(0,seconds)]
[OFFSET(displacement,length)]
[SUBSYS(OPSS,ssid)]
[SYSTEM(msfids)]
[SYSWAIT(seconds)]
[TOKEN(ATMSVTK,tkvname)]
```

varname

For the value of *varname*, specify the complete name of the global variable whose value you want to retrieve. The *varname* that you specify can be either of the following:

- A valid CA OPS/MVS global variable name beginning with a valid global variable stem prefix (GLOBAL, GLOBALx, GLVTEMPx, or GLVJOBID).
- An Automate-format variable name up to 32 characters in length. CA OPS/MVS looks at the values of the ATMLOCALSCOPEnn and ATMSHAREDSCOPEnn parameters to determine the appropriate global variable stem prefix and adds it internally.

outputvar

(Optional) When the OPSGETV command processor is running in a CLIST or REXX environment, use the *outputvar* operand to specify the name of the CLIST or REXX variable into which the retrieved variable value should be stored.

Note: If you specify a value for *outputvar*, it must precede any other optional keywords that you specify.

Default: ATMSV

CMDRESP

(Optional) The CMDRESP keyword specifies how the output from the OPSGETV command processor is handled. Values are:

TERMINAL

Displays the first 100 characters of the retrieved variable value and the update counter token value at the terminal. A plus sign (+) at the end of the retrieved variable value indicates that the value is longer than 100 characters.

REXX

Creates the REXX output variable specified by *outputvar* and the token variable specified by TOKEN(tkname). Stores the retrieved variable value in *outputvar* and the retrieved update counter token value in tkname.

CLIST

Creates the CLIST output variable specified by *outputvar* and the token variable specified by TOKEN(tkname). Stores the retrieved variable value in *outputvar* and the retrieved update counter token value in tkname.

NOWHERE

Tells CA OPS/MVS that it should not issue any messages except error messages. This option is provided to eliminate messages in REXX programs.

Default of CMDRESP when you invoke your OPSGETV request as a TSO command: both CMDRESP(TERMINAL) and CMDRESP(CLIST)

Default of CMDRESP when you invoke your OPSGETV request as an OPS/REXX or TSO/E REXX function: CMDRESP(REXX)

Note: Values of CMDRESP(CLIST) and CMDRESP(REXX) are honored only when the appropriate variable creation environment is possible.

DELAY

(Optional) The DELAY keyword indicates the number of seconds that your OPSGETV request waits before it can be processed. The default is 0; you can specify any number of seconds from 1 to 300.

OFFSET

(Optional) To retrieve only a substring portion of the value of a variable, use the OFFSET keyword to indicate two things:

- The substring displacement of the value to be retrieved, relative to 1
- The length of the value to be retrieved

CA OPS/MVS retrieves only the requested portion of the value of the variable. If displacement is outside the current value of the variable, CA OPS/MVS returns a null value. The range for both displacement and length is from 1 to 32,000. The default of displacement is 1; the default of length is the length of the current value of the variable.

SUBSYS

(Optional) For more information on the SUBSYS keyword, see Specifying a Subsystem ID on a POI Command Processor in this chapter.

SYSTEM

(Optional) For more information on the SYSTEM keyword, see Specifying an MSF System ID on a POI Command Processor in this chapter.

Note: You may specify a single system only.

SYSWAIT

(Optional) For a cross-system request, the SYSWAIT keyword specifies the number of seconds that CA OPS/MVS should wait for the remote system to receive and respond to the OPSGETV request. The default is the value of the MSFSYSWAIT parameter; you can specify any number of seconds from 1 to 600.

TOKEN

(Optional) Use the TOKEN keyword to specify the name of the CLIST or REXX variable that is to be set to the value of the update counter token. You can use this token value to perform a serialized update in a subsequent OPSSETV request. CA OPS/MVS sets *tkvname* only when the OPSGETV command processor is running in a CLIST or REXX program. The token value is only displayed or assigned to a variable when the TOKEN keyword is explicitly specified.

Default: ATMSVTK

Examples: OPSGETV

- This example retrieves the value and update token of the GLOBAL0.XYZ variable. If the command is issued from a CLIST or REXX program, CA OPS/MVS stores the retrieved value of the variable in the OUTVAR variable and stores the retrieved update counter token value in the OUTTK variable. In addition, CA OPS/MVS issues messages indicating the retrieved variable value and update token value:

```
OPSGETV GLOBAL0.XYZ OUTVAR TOKEN(OUTTK)
```

- This REXX function finds the current value of the GLOBAL0.XY variable and retrieves the substring portion of the variable beginning at the fifth character of the value of the variable and continuing for a length of eight characters (or until the end of the variable value, whichever comes first). The resulting substring will be stored in the *val* variable. If the GLOBAL0.XY variable does not exist or an error occurs, the *val* variable is null. No messages except error messages are produced. The OPSRC variable is set to the same return code that would have been set if you had invoked OPSGETV as a TSO command:

```
val = OPSGETV('GLOBAL0.XY OFFSET(5,8) CMDRESP(NOWHERE)')
```

Return Codes

OPSGETV produces these return codes:

Return Code	Description
0	The command executed successfully.
4	The command failed due to errors.
12	The CA OPS/MVS product is inactive.
40	You invoked the OPSGETV request as a REXX function, and it included an invalid REXX function argument.

OPSGETVL Command Processor—Get Global Variable Names

The OPSGETVL command processor retrieves the names of global variables that match a *variablename* mask that you specify. Depending on the keywords that you specify for your OPSGETVL request, CA OPS/MVS either displays the list of variables on your terminal or stores them in CLIST or REXX variables. Typically, you will use the retrieved variable names in subsequent variable manipulation operations.

In addition to invoking OPSGETVL as a TSO command, you can invoke it as an OPS/REXX or TSO/E REXX function. The function has a single argument equal to the OPSGETVL TSO command syntax. The value returned by the function is the count of retrieved variable names. The CA OPS/MVS product sets the OPSRC variable to the same return code that would have been generated if you had invoked OPSGETVL as a TSO command.

This command has the following format:

```
OPSGETVL
{namemask}
[prefix]
[CMDRESP(TERMINAL|REXX|CLIST|NOWHERE)]
[DELAY(0,seconds)]
[MAX(99,count)]
[SORT(ASCEND|DESCEND)]
[SUBSYS(OPSS,ssid)]
[SYSTEM(msfids)]
[SYSWAIT(seconds)]
[TOKEN(nn)]
```

namemask

For the value of *namemask*, specify either a complete variable name or a variable name that includes one or both of the special characters described here. The CA OPS/MVS product searches through the global variables on the specified system and tries to find any names that match *namemask*.

The *namemask* can be up to the maximum global variable size in length, and is automatically converted to uppercase.

You can use two special characters in the *namemask*:

- + - Matches any single character in a variable name.
- * - Used as the last character in the *namemask*, this wildcard character matches any number of trailing characters in a variable name.

The *namemask* that you specify can be either of the following:

- A valid CA OPS/MVS global variable name beginning with a valid global variable stem prefix (GLOBAL, GLOBALx, GLVTEMPx, or GLVJOBID).
- An Automate-format variable name up to 32 characters in length. CA OPS/MVS looks at the values of the ATMLOCALSCOPE*nn* and ATMSHAREDSCOPE*nn* parameters to determine the appropriate global variable stem prefix and adds it internally.

Guidelines for specifying *namemask*:

- If your name mask is for CA OPS/MVS global variable names, do not include wildcard characters in the first stem of the *namemask* value. Name masks for Automate-format variable names can include wildcard characters anywhere in the mask, since CA OPS/MVS adds the appropriate global or shared stem prefix to the mask internally.
- When you specify an Automate-format name mask that contains wildcard characters, and the name mask would usually match an Automate built-in variable name, the OPSGETVL command processor does not return the matching built-in variable names. For example, if you specify OPSGETVL ATMSYST*, the built-in variable ATMSYSTEM is not returned. However, if you specify OPSGETVL ATMSYSTEM, ATMSYSTEM is returned because no wildcard characters are present in your *namemask* value.

namemask examples:

GLOBAL0.ABCD*

Matches all CA OPS/MVS global variable names with 12 or more characters that begin with GLOBAL0.ABCD, such as GLOBAL0.ABCDEF.

TAPE_+++_STATUS

Matches all Automate-format variable names with 15 characters, where the sixth through eighth characters can be any character and the other characters are equal to those in the mask, such as TAPE_280_STATUS.

+++

Matches all Automate-format variable names composed of three characters.

+++*

Matches all Automate-format variable names composed of three or more characters.

GLVTEMP1.*

Matches all CA OPS/MVS global variable names that begin with the stem GLVTEMP1.

prefix

Note: If you specify a value for *prefix*, it must precede any other optional keywords that you specify.

(Optional) When you invoke OPSGETVL from a CLIST or REXX EXEC, you can assign the retrieved variable names to a variable array. Use the *prefix* operand to indicate the prefix of the CLIST or REXX variable names into which the retrieved variable names are to be stored.

Default: GETVL

The value of *prefix* affects the following:

- If you specify CMDRESP(REXX), the value of *prefix* becomes the stem name of the REXX variables in the array.
- If you specify CMDRESP(CLIST), the index value of the array is concatenated to the value of *prefix*.
- The token value is returned in the variable *prefix*TK.

For more information, see CLIST or REXX Variables Generated by OPSGETVL in this chapter.

CMDRESP

(Optional) The CMDRESP keyword specifies where the output goes when OPSGETVL executes. Values are:

TERMINAL

Displays the response (number of variable names returned, the token, and all variable names selected) at the terminal.

REXX

Generates an output variable array using the REXX stem variable format, which is *prefix.index*.

CLIST

Generates an output variable array using the CLIST variable name format, which is *prefix||index*.

NOWHERE

Tells CA OPS/MVS that it should not issue any messages except error messages. This option is provided to eliminate messages in REXX programs.

When you invoke your OPSGETVL request as a TSO command, the default of CMDRESP is both CMDRESP(TERMINAL) and CMDRESP(CLIST).

When you invoke your OPSGETVL request as an OPS/REXX or TSO/E REXX function, the default of CMDRESP is CMDRESP(REXX).

Note: Values of CMDRESP(CLIST) and CMDRESP(REXX) are honored only when the appropriate variable creation environment is possible.

For more information, see CLIST or REXX Variables Generated by OPSGETVL in this chapter.

DELAY

(Optional) The DELAY keyword indicates the number of seconds that your OPSGETVL request waits before it can be processed. The default is 0; you can specify any number of seconds from 1 to 300.

MAX

(Optional) The MAX keyword indicates the maximum number of variable names that CA OPS/MVS is to return for this OPSGETVL request. If more matching variable names exist than the number specified by count, CA OPS/MVS generates a resume token. The default is 99; you can specify any number from 1 to 2048.

SORT

(Optional) The SORT keyword tells CA OPS/MVS how to sort the returned variable names before displaying them or assigning them to CLIST or REXX variables.

Values are:

ASCEND

(Default) Sort the variable names in ascending order.

DESCEND

Sort the variable names in descending order.

SUBSYS

(Optional) For more information on the SUBSYS keyword, see Specifying a Subsystem ID on a POI Command Processor in this chapter.

SYSTEM

(Optional) For more information on the SYSTEM keyword, see Specifying an MSF System ID on a POI Command Processor in this chapter.

Note: You may specify a single system only.

SYSWAIT

(Optional) For a cross-system request, the SYSWAIT keyword specifies the number of seconds that CA OPS/MVS should wait for the remote system to receive and respond to the OPSGETVL request. The default is the value of the MSFSYSWAIT parameter; you can specify any number of seconds from 1 to 600.

TOKEN

(Optional) Use the TOKEN keyword to specify the numeric token number that was returned by a previous OPSGETVL request. By specifying a value for this keyword, you indicate that CA OPS/MVS should start the *current* search for matching variable names at the variable name following the last name that was returned by the previous OPSGETVL request. The TOKEN keyword is useful when large numbers of variable names are being retrieved. There is no default.

Examples: OPSGETVL

- This example finds all variable names that match the mask and displays them at the terminal. A maximum of 200 names is returned:

```
OPSGETVL GLOBAL0.XYZ+ABC* MAX(200)
```

- This REXX function finds all global variable names that match the mask and generates a REXX variable array where VNAME.0 contains the variable name count and VNAME.*n* is the *n*th variable name returned. The *cnt* variable contains the same value as VNAME.0. The OPSRC variable is set to the same return code that would have been set if you had invoked OPSGETVL as a TSO command:

```
cnt = OPSGETVL('GLOBAL0.XYZ* VNAME)
```

CLIST or REXX Variables Generated by OPSGETVL

OPSGETVL generates these CLIST or REXX variables:

CLIST	REXX	Value
GETVLTK	GETVLTK	Resume token number
GETVL, GETVL0	GETVL.0	Returned name count
GETVL1 to GETVL <i>n</i>	GETVL.1 to GETVL. <i>n</i>	First returned name to the last returned name

Return Codes

OPSGETVL produces these return codes:

0

The command executed successfully.

4

No matching variable names were found.

8

The maximum number of matching variable names was returned, but more matching names exist.

10

The value you specified for the TOKEN keyword is invalid. Reissue the original OPSGETVL request.

12

The CA OPS/MVS product is inactive.

40

You invoked the OPSGETVL request as a REXX function, and it included an invalid REXX function argument.

OPSHFI Command Processor—Store Global Variable Values

The OPSHFI command processor provides the capability to store values of global variables on an external VSAM key-sequenced data set that can be shared among systems. Using OPSHFI, you can WRITE global variables to the VSAM file, READ variable records from the VSAM file, or DELETE variable records from the VSAM file.

Since global variables are usually checkpointed as their values change over time, you can use the OPSHFI command processor to reset variables to specific initial values at CA OPS/MVS initialization (or as required).

When the VSAM file is shared across systems, values can be propagated to or synchronized with other copies of CA OPS/MVS. This ability is useful when cross-system OPSVALUE function calls can be too voluminous.

In addition to invoking OPSHFI as a TSO command, you can invoke it as an OPS/REXX or TSO/E REXX function. The function has a single argument equal to the OPSHFI TSO command syntax. The value returned by the function is the count of retrieved VSAM records that were selected for processing. The CA OPS/MVS product sets the OPSRC variable to the same return code that would have been generated if you had invoked OPSHFI as a TSO command. When OPSHFI is invoked as a function in an AOF rule in which waits are not allowed, the NOOUTPUT keyword is implied.

Use this format for the OPSHFI command processor:

```
OPSHFI
{READ|WRITE|DELETE}
{namemask}
[QDRESP(TERMINAL|NOWHERE)]
[DELAY(0,seconds)]
[GLVRULES(YES|NO)]
[OUTPUT|NOOUTPUT]
[SHARED|LOCAL]
[SMFD(smfid)]
[SUBSYS(OPSS,ssid)]
[SVRULES(YES|NO)]
[SYSTEM(msfids|ALL|EXT)]
[SYSWAIT(120,seconds)]
```


READ, WRITE, and DELETE

The READ, WRITE, and DELETE keywords indicate the function that you want to perform with this OPSHFI request. Specify any of the following:

READ

Reads every variable record from the VSAM file that meets the following criteria:

- The variable name matches the value of *namemask*.
- The scope of the variable matches the specified scope criteria.

For each variable record that is selected, the OPSHFI READ command creates a global variable using the variable name and value from the VSAM record. If a global variable was written as a shared variable, then it can be read into memory on any system that shares the VSAM file.

WRITE

Finds all global variables that match the value of *namemask* and writes their names and values to the VSAM file with the specified scope. The global variables can be written as shared variables or as local variables. Shared variables are made available to any system that shares the VSAM data set; however, local variables are available only to the system that wrote them. Writing global variables to VSAM does not change their values in memory. Likewise, subsequent updates to global variables in memory are not automatically sent to VSAM.

DELETE

Finds all matching variable records on the VSAM file and deletes them from the file. No global variables are altered.

Important! The DELETE keyword does not remove the variables from memory.

namemask

For the value of *namemask*, specify either a complete variable name or a variable name that includes one or both of the special characters described here. The CA OPS/MVS product searches through the global variables on the specified system and tries to find any names that match *namemask*.

The *namemask* can be up to the maximum global variable size in length, and is automatically converted to uppercase.

You can use two special characters in the *namemask*:

- **+** - Matches any single character in a variable name.
- ***** - Used as the last character in the *namemask*, this wildcard character matches any number of trailing characters in a variable name.

The *namemask* that you specify can be either of the following:

- A valid CA OPS/MVS global variable name beginning with a valid global variable stem prefix (GLOBAL, GLOBALx, or GLVTEMPx). Global stem prefix GLVJOBID is not supported by the OPSHFI function.
- An Automate-format variable name up to 32 characters in length. CA OPS/MVS looks at the values of the ATMSHAREDScope nn parameters to determine the appropriate global variable stem prefix and adds it internally.

Guidelines for specifying *namemask*:

- If your name mask is for CA OPS/MVS global variable names, do not include wildcard characters in the first stem of the *namemask* value. Name masks for Automate-format variable names can include wildcard characters anywhere in the mask, since CA OPS/MVS adds the appropriate global or shared stem prefix to the mask internally.
- You can specify only global and shared variable names when you specify an Automate-format name mask. Automate built-in and environmental variable names are ignored.

Namemask examples:

GLOBAL0.ABCD*

Matches all CA OPS/MVS global variable names with 12 or more characters that begin with GLOBAL0.ABCD, such as GLOBAL0.ABCDEF.

TAPE_+++_STATUS

Matches all Automate-format variable names with 15 characters, where the sixth through eighth characters can be any character and the other characters are equal to those in the mask, such as TAPE_280_STATUS.

+++

Matches all Automate-format variable names composed of three characters.

+++*

Matches all Automate-format variable names composed of three or more characters.

GLVTEMP1.*

Matches all CA OPS/MVS global variable names that begin with the stem GLVTEMP1.

CMDRESP

(Optional) The CMDRESP keyword specifies how output is to be handled when OPSHFI executes.

Values are:

TERMINAL

Displays the response at the terminal.

NOWHERE

Tells CA OPS/MVS that it should not issue any messages except error messages. This option is provided to eliminate messages in REXX programs.

When you invoke your OPSHFI request as a TSO command, the default of CMDRESP is CMDRESP(TERMINAL).

When you invoke your OPSHFI request as an OPS/REXX or TSO/E REXX function, the default of CMDRESP is CMDRESP(NOWHERE).

DELAY

(Optional) The DELAY keyword indicates the number of seconds that your OPSHFI request waits before it can be processed. The default is 0; you can specify any number of seconds from 1 to 300.

GLVRULES

(Optional) The GLVRULES keyword is an alias of the SVRULES keyword. Both keywords perform the same function. For details, see SVRULES Keyword in this chapter.

OUTPUT and NOOUTPUT

(Optional) If you specify OUTPUT, the following occur:

- The request is queued to the shared file server task in the CA OPS/MVS main product address space
- The command waits for the request to complete
- The command returns the count of variable records that were selected for processing

If you specify NOOUTPUT, no value is returned and return code 2 is set.

SHARED and LOCAL

(Optional) If you specify LOCAL, CA OPS/MVS sets the SMF ID to the SMF ID of the system on which the OPSHFI request executes.

If you specify SHARED, the SMF ID is blank. Only records with a blank SMF ID suffix are processed.

When the value of namemask begins with a stem (in other words, it is a CA OPS/MVS-format name mask), the default is LOCAL. When the value of namemask is an Automate-format name mask, the default is LOCAL and SHARED because the Automate prefix tables are used to determine the correct scope.

SMFID

(Optional) Use the SMFID keyword to specify the SMF ID suffix that CA OPS/MVS should append to the VSAM variable name record key. If your OPSHFI request specifies the READ or DELETE keyword, CA OPS/MVS processes only those records having this SMF ID suffix. If your request specifies the WRITE keyword, CA OPS/MVS appends the specified SMF ID to each variable record that is written. There is no default.

SUBSYS

(Optional) For more information on the SUBSYS keyword, see Specifying a Subsystem ID on a POI Command Processor in this chapter.

SVRULES

(Optional) The SVRULES keyword indicates whether you want the variables that OPSHFI changes to be processed by global variable rules. Values are:

- YES - Indicates that you want rules to process the variables.
- NO - (Default) Indicates that you do not want rules to process the variables.

Note: The SVRULES keyword applies only when you are requesting a READ of the VSAM file.

SYSTEM

(Optional) For more information on the SYSTEM keyword, see Specifying an MSF System ID on a POI Command Processor in this chapter.

Note: If more than one system is specified or implied, then the NOOUTPUT keyword is implied.

SYSWAIT

(Optional) The SYSWAIT keyword specifies the number of seconds that CA OPS/MVS should wait for a response to the OPSHFI request on either the local system or a remote system. The default is 120; you can specify any number of seconds from 1 to 600. The default is high due to the possibility of reserve contention for control of the VSAM file among multiple systems.

Examples: OPSHFI

- This example reads every variable record from the VSAM file that meets these criteria:

- The variable name matches the mask.
- The variable record specifies the SMF ID of the local system.

For each variable record that is selected, CA OPS/MVS creates or updates the corresponding global variable with the value in the record:

```
OPSHFI READ GLOBAL0.XYZ+ABC* LOCAL
```

- This REXX function searches the global variable pool for matching names. For any matches it finds, it adds a record to the VSAM file (or updates a record already in the VSAM file) with the current value of the global variable. A blank SMF ID is appended to the variable name key. The number of VSAM records added or updated is returned in the *cnt* variable. The OPSRC variable is set to the same return code that would have been set if you had invoked OPSHFI as a TSO command:

```
cnt = OPSHFI('WRITE GLOBAL0.SHARED.XYZ* SHARED')
```

Variable Characteristics

Review these variable characteristics:

- A global variable written to the shared VSAM file can have a local or global scope.
- A local variable has the SMF ID of the local system appended to its variable name to form the complete key of the variable record.
- A shared variable has no SMF ID appended to its name.
- Local variables are intended for a particular system, while shared variables are intended for multiple systems. However, you can use the SMFID keyword to manipulate the variables from any system using the VSAM file.

OPSHFI Return Codes

OPSHFI produces these return codes:

0

The command executed successfully.

2

The command executed successfully, but no variable count was returned since NOOUTPUT was specified or implied.

4

For read operations, no matching VSAM records were found.

For write operations, there is no space in the VSAM data set or the data set is not allocated.

8

An error was encountered during processing.

12

The CA OPS/MVS product is inactive.

40

You invoked the OPSHFI request as a REXX function, and it included an invalid REXX function argument.

OPSIMEX Command Processor—Executes a Member as a REXX Program

The OPSIMEX (OI) command processor causes a member found in the OPSEEXEC concatenation or the SYSEEXEC concatenation to be executed as a REXX program.

Use the following syntax for the OPSIMEX or OI command:

```
OPSIMEX[OI  
{PROGRAM(member)}  
[ARG(arguments)]  
[ITRACE(x)]  
[MAXEDQ(lines)]  
[SUBSYS(ssid)]  
[WORKSPACE(size)]
```

Note: The *member* keyword refers to the name of a member that is to be found by searching first the OPSEEXEC concatenation, and then the SYSEEXEC concatenation.

You can also use these abbreviated formats for OPSIMEX:

```
OPSIMEX member argument  
OI member argument
```

OPSIMEX Keywords and Arguments

The OPSIMEX command uses exactly the same keywords and arguments as the OPSEEXEC command, except that the data set name containing the member to be executed is implicit and cannot be specified in OPSIMEX.

Note: When using the form of the OPSIMEX or OI command that uses PROGRAM or other keywords, enclose the argument string in single quotation marks.

The record formats supported for the SYSEEXEC libraries are the same as those supported for REXX data sets used to invoke OPS/REXX explicitly.

OPSPARM Command Processor

This command processor displays, and enables you to change, the CA OPS/MVS operating parameters.

You can issue the OPSPARM command:

- When initializing CA OPS/MVS, through the OPSTART1 CLIST or the appropriate OPSSPA00 member of the Logical Parmlib Concatenation that OPSTART1 invokes. This is the *only* way to set the parameters described as parameters to set at initialization.
- Using option 6 of the ISPF main menu to display or change parameter values.
- Anywhere a TSO command processor or OPS/REXX program can execute.

OPSPARM Format—Change Parameters

To change parameters, use this format of the OPSPARM command:

```
OPSPARM
{SET(parmname)}
{VALUE(parmvalue)}
[SUBSYS(ssid)]
[SYSTEM(msfids)|ALL|EXT]
[SYSWAIT(seconds)]
```

SET and VALUE

Identifies the name of the parameter to be changed and the new value for the parameter.

Always specify the SET and VALUE keywords together, for example:

```
OPSPARM SET(OSFMAX) VALUE(2)
OPSPARM SET(SYSID) VALUE(FRODSYS)
```

If you specify an invalid parmname, the OPSPARM command processor issues an error message and a return code of 60.

SUBSYS

(Optional) For more information on the SUBSYS keyword, see Specifying a Subsystem ID on a POI Command Processor in this chapter.

SYSTEM (Used With SET)

(Optional) For more information on the SYSTEM keyword, see Specifying an MSF System ID on a POI Command Processor in this chapter.

Examples: SYSTEM keyword

1. In this OPSPARM command, the SYSTEM keyword specifies that the OCWAIT parameter will be set to 7 on system OPSY:
OPSPARM SET(OCWAIT) VALUE(7) SYSTEM(OPSY)
2. In this example, the SYSTEM keyword specifies that the OCWAIT parameter is to be set to 9 on all systems except the local system:
OPSPARM SET(OCWAIT) VALUE(9) SYSTEM(EXT)
3. Finally, this example indicates that the OCWAIT parameter is to be set to 8 on all systems including the local one:
OPSPARM SET(OCWAIT) VALUE(8) SYSTEM(ALL)

SYSWAIT

(Optional) Specifies the number of seconds that CA OPS/MVS waits for a response from a remote system. You can specify a value from 1 to 300; the default is the value of MSFSYSWAIT.

OPSPARM Format—Display Parameters

To display parameters, use this format for the OPSPARM command:

```
OPSPARM
{SHOW(name|GROUPS|ALL)}
[INFO|NAMES|CLIST]
[SYSTEM(msfid|ALL|EXT)]
[SYSWAIT(seconds)]
```

SHOW

The SHOW keyword, used when you display parameter values, specifies which parameters to display.

Specify one of these options:

name

Displays the value of the named parameter or the values of all parameters in the named group.

GROUPS

Displays a list of the parameter groups.

ALL

Displays all parameter values.

The following examples use the SHOW keyword.

- This example displays the value of the ECFSECURITY parameter:
- `OPSPARM SHOW(ECFSECURITY)`
- This example displays the values of all parameters in the PRODACTIVITY parameter group:
`OPSPARM SHOW(PRODACTIVITY)`

INFO

(Optional) Use the INFO keyword with the SHOW keyword to display the possible values the parameter can have. For example, to see all possible values of the ECFSECURITY parameter, issue this command:

```
OPSPARM SHOW(ECFSECURITY) INFO
```

NAMES

(Optional) Use the NAMES keyword with the SHOW keyword to display the names and modifiability indicators of individual parameters. You cannot use NAMES with the SHOW(GROUPS) keyword.

For example, to see the names of the parameters in the PRODACTIVITY group, issue this command:

```
OPSPARM SHOW(PRODACTIVITY) NAMES
```

CLIST

(Optional) Creates CLIST variables based on CA OPS/MVS parameters. These variables have the same names and corresponding values as the parameters.

The CLIST keyword is especially useful for obtaining the values of fields such as SYSID, which you can use in CLISTs for conditional processing based on the system where the CLIST runs.

SYSTEM(Used With SHOW)

(Optional) For more information on the SYSTEM keyword, see Specifying an MSF System ID on a POI Command Processor in this chapter.

Note: If more than one system is specified or implied, and the local system is one of the systems that will receive the request, then CA OPS/MVS returns only the value on the local system.

In the following example, the SYSTEM keyword in this command indicates that SYSA is the name of the system on which the command is to execute and from which the response is to be returned:

```
OPSPARM SHOW(OCWAIT) SYSTEM(SYSA)
```

SYSWAIT

(Optional) Specifies the number of seconds CA OPS/MVS waits for a response from a remote system. You can specify a value from 1 to 300.

Default: MSFSYSWAIT

Using OPSPARM to Display and Set Message Severity Codes

While initializing CA OPS/MVS or after it becomes active, you can use the OPSPARM command processor to display and set severity codes for CA OPS/MVS messages.

Display a Message Severity Code

To display the severity code *c* for message OPS*nnnn*c, use this form of the OPSPARM command:

```
OPSPARM SHOW(OPSnnnn)
```

Change a Message Severity Code

To change the severity code *c* to *d* for message OPS*nnnn*c, use this form of the OPSPARM command:

```
OPSPARM SET(OPSnnnn) VALUE('d')
```

Important! Before changing the severity code of any CA OPS/MVS message, see the chapter “Understanding CA OPS/MVS Messages” in the *User Guide*.

OPSPARM Return Codes

The OPSPARM command produces the following return codes and messages:

Return Code	Message Number	Message Text
0	OPS1250	IS THE ACTUAL VALUE OF THE PARAMETER ITSELF
4	n/a	CLIST KEYWORD USED OUTSIDE OF A CLIST
8	OPS1252	CANNOT BE USED BEFORE PRODUCT ACTIVE
12	OPS1253	SET/SHOW KEYWORDS MISSING
16	OPS1254	NO VALUE KEY ENTERED WITH SET
20	OPS1255	PARM CANNOT BE CHANGED AFTER INIT
24	OPS1256	OUTPUT ONLY FIELD

Return Code	Message Number	Message Text
28	OPS1257	INPUT ONLY FIELD
32	OPS1258	SET DATA VALUE IS INVALID
36	OPS1259	AUTHORIZATION CHECK FAILED
40	OPS1260	'ALL' OR GROUP NAME CANNOT BE USED WITH SET
44	OPS1261	KEYWORD CANNOT BE USED WITH SET/GROUPS
None for this message	OPS1262	USED INTERNALLY TO CREATE CLIST VARIABLES
56	n/a	NO PARAMETER INFORMATION AVAILABLE
60	n/a	INVALID PARAMETER NAME
80	OPS1270	TSO/E IS NOT INSTALLED
84	OPS1271	COMMAND BUFFER PARSE ERROR
88	OPS1272	COMMAND NOT ISSUED WITH AUTHORIZATION
92	OPS1273	MASTER CONTROL BLOCK ERROR
96	OPS1274	MASTER CONTROL BLOCK NOT FOUND
100	OPS1275	AUTHORIZATION EXIT ABEND FAILURE
104	OPS1276	SUBSYSTEM DOES NOT EXIST
108	OPS1277	CONTROL BLOCK NOT FOUND
112	n/a	MASTER/LOCAL VERSION CODE MISMATCH
116	OPS1279	CLIST VARIABLE UPDATE ERROR
None for this message	OPS1280	WRITE CURRENT PARM SET/SHOW REQUEST ONTO SYSLOG
120	n/a	SYSTEM MOVE DATA ROUTINE FAILED
124	n/a	SERVICE ROUTINE FAILED
128	n/a	CROSS SYSTEM NOT ACTIVE
132	n/a	MESSAGE QUEUE ALLOCATION FAILED
136	n/a	LAST MESSAGE NEVER RECEIVED
140	n/a	COMMAND OUTPUT ERROR
144	n/a	CROSS SYSTEM SEND ERROR
148	OPS3533	SECURE/NOSECURE ACTION COMMAND

OPSQL Command Processor—Invoke SQL Statements

Use the OPSQL command processor to invoke SQL statements from a TSO terminal, a TSO CLIST, or a TSO/E REXX program. These SQL statements enable you to collect information about your system, organize it into tables containing rows and columns of related data, and retrieve data from those tables for use in OPS/REXX programs and AOF rules.

This command has the following format:

```
OPSQL  
{statement}
```

Note: The chapter “Using the Relational Data Framework” in the *User Guide* describes each of the statements you can invoke through the OPSQL command processor and explains the syntax of each statement and how to use it.

OPSREPLY Command Processor—Compare WTOR Messages

The OPSREPLY command processor compares WTOR messages to a specified set of values and issues replies to messages containing matching values.

This command has the following syntax:

OPSREPLY

You must choose *at least one* of the following keywords:

{MSID(*msid*)} {JOBNAME(*jobname*)} {JOBNUMBER(*jobnum*)}

{MSGID(*msgid*)|MSGTEXT(*msgtxt*)} {REPLYID(*replyid*)} {STEPNAME(*stepname*)}

In addition, you must specify the following required keyword:

{TEXT(*msgtxt*)}

[CAPTURE(*msgtextlist*)]

[CMDECHO(YES|NO)]

[CMDLOG(YES|NO)]

[CMDRESP(*destination*)]

[CMDWAIT(*seconds*)]

[DELAY(*delaytime*)]

[INTERVAL(*centiseconds*)]

[LOG(YES|NO|OFF)]

[MAXCMDOUT(*number*)]

[MAXREPLY(*replies*)|UNIQUE|ALL|ANY]

[MFORM(M|J)]

[NAME|CONNAME(*consolename*)]

[NOCLIST]

[NOSYSLEX|SYSLEX]

[OUTDELIM(*delimstring*,YES|NO)]

[OUTPUT|NOOUTPUT]

[PREFIX(*prefix*)]

[STOPEND(YES|NO)]

[STOPMSG(*msgtextlist*)]

[STOPRESP(*msgtextlist*)]

[SUBSYS(*ssid*)]

[SYSID|SYSTEM(*msids*)|ALL|EXT]

[SYSNAME(*sysname*)]

[SYSWAIT(*seconds*)]

[TRACE]

[WAIT(0,*waittime*)]

The following describes most of the keywords shown in the preceding syntax diagram. For information about the other keywords, see OPSCMD Command Processor in this chapter.

ALL

(Optional) Specifying the ALL keyword is the same as specifying MAXREPLY(9999). The MAXREPLY, ANY, ALL, and UNIQUE keywords are mutually exclusive.

ANY

(Optional) Specifies that OPSREPLY replies to the first WTOR message that matches the selection criteria, regardless of other WTOR messages that can match it. The ANY keyword is the opposite of the UNIQUE keyword. The MAXREPLY, ANY, ALL, and UNIQUE keywords are mutually exclusive. Specifying the MAXREPLY keyword implies the ANY keyword.

CMDWAIT

(Optional) Defines how long the CA OPS/MVS product waits for response collection to complete. You can specify a value from 0 to 300 seconds.

DELAY

(Optional) Defines, in seconds, how long CA OPS/MVS suspends processing of the OPSREPLY command processor. You can specify between 1 and 300 seconds; there is no default.

Here are examples of using the DELAY keyword:

```
OPSREPLY MSGD($HASPxxx) TEXT(WARM) DELAY(10)
OPSREPLY MSGD(JEC107A) TEXT(U) DELAY(10) SUBSYS(OPSF)
```

IMSID

Identifies the IMS control region that issued the WTOR. For imsid, specify a region ID of up to four characters. For example, you can specify the following values:

```
OPSREPLY IMSID(imsa)
OPSREPLY IMSID(inst)
```

If you do not also specify the MSGID keyword, OPSREPLY makes sure that the ID of the WTOR message is DFS996I, DFS972A, or DFS810A before determining whether to reply to it.

The IMS ID strings are assumed to be located at offsets of 25, 39, and 79.

JOBNAME

Specifies the name of the batch job, started task, or TSO user that issued the WTOR. A jobname can contain as many as eight characters.

Unless you specify the ANY keyword, which is described later in this section, OPSREPLY sets a failure return code if several WTORs have the same job name and there is no other way to distinguish between them.

If the WTOR message to be replied to came from an IMS region, use the IMSID keyword instead of JOBNAME.

JOBNUMBER

Specifies the specific JES job number of the issuer of the WTOR message. Specify only the numeric portion of the job ID; do not include the prefix (such as JOB, STC, or TSU). The value of jobnum can contain from 1 to 5 digits.

MAXREPLY

(Optional) Specifies the maximum number of replies that OPSREPLY performs. The value of replies can range from 1 to 9999. OPSREPLY replies to all WTOR messages that meet the selection criteria you specify, up to the limit indicated by the MAXREPLY value. The MAXREPLY, ANY, ALL, and UNIQUE keywords are mutually exclusive. Specifying the MAXREPLY keyword implies the ANY keyword.

MSGID

This keyword identifies a specific message ID, which is usually (but not always) the first token in the message. A message ID can contain up to 10 characters. For information about message IDs that exceed 10 characters in length, see below. For example, all of the following are valid message IDs:

```
OPSREPLY MSGID(DSI802A)
OPSREPLY MSGID(LUK0090A)
OPSREPLY MSGID(DFS996)
```

Unless you also specify the ANY, ALL, or MAXREPLY keyword, OPSREPLY sets a failure return code if several WTORs have the same message ID and there is no other way to distinguish between them.

If the length of the message ID text is 10 characters or less, the text refers to the formal message ID of the WTOR, as it does in AOF message rules. If the message ID text is more than 10 characters, the text is treated as if it was entered using the MSGTEXT keyword, which allows the selection of WTORs based on a leading substring of the message text. You can specify up to 124 characters.

OPSREPLY MSGID('DSI803A A44IM REPLY') is the same
as OPSREPLY MSGTEXT('DSI803A A44IM REPLY').

The MSGID and MSGTEXT keywords are mutually exclusive.

MSGTEXT

This keyword allows you to select WTORs based on a leading substring of the message text. You can specify up to 124 characters. The first character of the substring comparison is always the message ID portion of the WTOR—the reply ID and special characters are skipped. The comparison text must match exactly, including spaces and delimiters, up to the length of the substring specified for this keyword. The MSGTEXT keyword cannot be abbreviated.

The MSGID and MSGTEXT keywords are mutually exclusive.

Examples: MSGTEXT keyword:

- To reply to a particular NetView VTAM APPLID WTOR, specify:
OPSREPLY MSGTEXT('DSI803A NEIV1') TEXT('CLOSE IMMED')
- To reply to either DSI802 or DSI803 NetView WTORs, specify:
OPSREPLY MSGTEXT('DSI80') TEXT('CLOSE IMMED')

NOSYSPLEX and SYSPLEX

(Optional) Use these keywords if your site runs in a sysplex environment:

NOSYSPLEX

(Default) The OPSREPLY command processor limits its search for matching WTOR messages to those WTOR messages that originated on the current system in the sysplex. OPSREPLY ignores WTOR messages that originated from other systems. NOSYSPLEX is the default.

SYSPLEX

The search of the OPSREPLY command processor includes WTOR messages from all systems in the sysplex. Since cross-system REPLY commands can be issued from a sysplex console, specifying SYSPLEX permits WTOR messages that did not originate from the current system to be selected for a cross-system reply. Specifying the SYSNAME keyword implies the SYSPLEX keyword.

REPLYID

Identifies a specific WTOR message by its z/OS reply number. Depending upon the release of z/OS at your site, the value of replyid ranges from 0 to 9999.

STEPNAME

Supplies the z/OS job step name for the address space that issued the WTOR message. For started tasks, this value is the ID from either the z/OS START command or the PROC name.

You can specify as many as eight characters for stepname. For instance, these are valid step names:

```
OPSREPLY STEPNAME(UPDATE)
```

```
OPSREPLY STEPNAME(DFHINIT)
```

If several WTORs have the same job step name and no other criteria distinguish between these messages, OPSREPLY sets a failure return code.

SUBSYS

(Optional) For more information on the SUBSYS keyword, see Specifying a Subsystem ID on a POI Command Processor in this chapter.

SYSID/SYSTEM

(Optional) For more information on the SYSTEM keyword, see Specifying an MSF System ID on a POI Command Processor in this chapter.

Note: If more than one system is specified or implied, then the NOOUTPUT keyword is implied. Since specifying more than one system results in multiple systems replying to the same reply number, it is unlikely that you will ever specify more than one system. The capability is only included for compatibility with OPSCMD, whose services are used to issue the REPLY command on the target system.

The SYSID/SYSTEM keywords do not imply that the OPSREPLY command is sent to another system. The checking for a matching WTOR is performed only on the local system. WTORs from other systems in the sysplex are only considered for matching if the SYSPLEX or SYSNAME keywords are also specified. The purpose of the SYSID/SYSTEM keywords is to issue the REPLY command on another system in the MSF complex.

The SYSID/SYSTEM specified must be part of the same sysplex from where the OPSREPLY command is issued. If there is a need to go to a system outside the sysplex, then use the OPSRMT command processor.

Here is an example of how to use OPSRMT with OPSREPLY:

```
OPSRMT SYSA Command('OPSREPLY MSGID(ABC123) text(u)')
```

SYSNAME

(Optional) Indicates the name of the z/OS system that issued the WTOR message. The value of sysname can be from one to eight characters, or * (asterisk) to indicate the current system. The SYSNAME refers to the SYSNAME keyword specified in the z/OS IEASYSnn parmlib member.

Typically, you use this keyword to reply to:

- (In a sysplex environment) WTOR messages issued by one system and received by another
- WTOR messages issued by JES3 running on a local processor and received by the JES3 global processor

If you specify a value for the SYSNAME keyword, the SYSPLEX keyword is implied. This is true unless you explicitly specify the NOSYSPLEX keyword.

Note: If the matched system name on a JES3 system is not the current system and the current system is the JES3 global processor, OPSREPLY prefixes the reply text with the JES3 SEND command so that it reaches the correct JES3 local processor.

TEXT

Supplies the text of the reply to the WTOR. Use single quotation marks to enclose the text if it contains delimiters and embedded blanks; otherwise, you can omit the quotes. To indicate a single quote in the text string, use a pair of single quotes as shown in the following example:

```
OPSREPLY MSGID(RY0002A) TEXT('IN "QUOTES"')
```

Reply text can contain up to 119 characters. If the reply text exceeds the maximum reply length of the current WTOR, OPSREPLY sets a failure return code.

Here are more examples of using the TEXT keyword:

```
OPSREPLY MSGID(IEC107A) TEXT (U)
OPSREPLY MSID(IMSA) TEXT('/DIS ACTIVE)
OPSREPLY JOBNAME(RYOSYS) TEXT('SHUTDOWN)
```

TRACE

Specify the TRACE keyword if you want messages indicating the WTOR selection criteria to appear in OPSLOG. If a reply ID for the WTOR was found, it appears along with the selection criteria.

If the OPSREPLY command processor finds no WTOR messages that match the selection criteria, OPSLOG contains the selection data for each WTOR message that the OPSREPLY command processor examined in its search.

UNIQUE

(Optional) Specifies that OPSREPLY replies to the WTOR message if and only if one matching WTOR message is found. If more than one WTOR message matches the selection criteria, CA OPS/MVS issues an error message. The MAXREPLY, ANY, ALL, and UNIQUE keywords are mutually exclusive. UNIQUE is the default.

WAIT

(Optional) Specifies how many seconds pass before OPSREPLY replies to a WTOR. WAIT enables you to reply to a WTOR that is usually (but not always) outstanding (such as a WTOR to which someone else is replying).

You can specify a wait time between 1 and 3600 seconds; the default is zero. If the wait time is greater than zero, OPSREPLY looks once each second for a WTOR that matches the values specified with the JOBNAME, STEPNAME, MSGID, or MSID keywords. It continues trying to respond to the WTOR until the wait time expires.

Examples: Using the WAIT keyword:

```
OPSREPLY MSGID(IEC107A) TEXT (U) WAIT(1)
OPSREPLY MSID(IMSA) TEXT('/DIS ACTIVE) WAIT(30)
```

How OPSREPLY Determines Which WTORs Receive Replies

OPSREPLY evaluates these message characteristics when selecting the WTOR to which it will reply:

- The message origin, specified using these keywords: JOBNAME, MSGID, or IMSID
- The job step of the program that issued the WTOR, specified using the STEPNAME keyword

You can specify more than one keyword. If you do, CA OPS/MVS evaluates all of their values as if you had specified them with a Boolean AND function.

If you specify the UNIQUE keyword or allow it to default, OPSREPLY compares the selection criteria to all WTOR messages. If more than one WTOR message meets the selection criteria, OPSREPLY issues an error message and does not perform the reply. If you specify the ANY keyword, OPSREPLY replies to only the first WTOR message that meets the selection criteria. If you specify the ALL keyword, OPSREPLY replies to all the WTOR messages that meet the criteria.

You can use the MAXREPLY keyword to specify the maximum number of WTOR messages OPSREPLY performs.

Considerations for Issuing OPSREPLY

Keep this information in mind when issuing the OPSREPLY command processor:

- The OPSREPLY command processor internally calls the OPSCMD command processor to issue the z/OS reply command when:
 - You specify one or more OPSCMD command processor keywords in the text of the OPSREPLY command processor.
 - You specify a cross-system keyword (for example, SYSNAME or SYSPLEX) in the text of the OPSREPLY command processor.
 - OPSREPLY works across systems that are part of the same sysplex.

The internal call to OPSCMD provides these benefits:

- It enables OPSREPLY to take advantage of the cross-system capabilities of OPSCMD
 - It permits users to specify the CMDRESP(*destination*,REPLY) keyword of OPSCMD to obtain the response to the reply command that will be issued. Variable output is also available; for example, you can specify CMDRESP(REXX,REPLY) to specify the generation of a series of REXX stem variables to hold the output.
- CA OPS/MVS converts the values of all keywords, including reply text, to uppercase characters.
 - In a JES3 environment, local WTOR messages appear on the global ORE chain as foreign OREs. Specifying the SYSPLEX or SYSNAME keyword can cause these OREs to be selected. When the reply must be routed to the local processor, the REPLY command is prefixed with the following:

**T sysname*

- Whenever you specify an OPSCMD command processor keyword in the text of the OPSREPLY command processor, OPSREPLY links to OPSOSCMD. For a description of the OPSCMD command processor, see OPSCMD Command Processor in this chapter.
- In coded automation procedures, such as CLISTs or OPS/REXX programs, always specify the complete text of the OPSREPLY command processor and its keywords and operands. Doing so avoids syntax ambiguity in the future as we add new command processors to the CA OPS/MVS product.

If you are invoking the OPSREPLY command processor online, as you would during an interactive TSO session, you can use the shortest unique abbreviations. Be aware that the shortest unique abbreviation of a command processor, keyword, or operand can change from release to release.

OPSREPLY Return Codes

OPSREPLY produces these return codes:

Return Code	Message Number	Message Text
0	OPS1300	REPLY COMPLETED NORMALLY
4	OPS1301	JOBNAME, MSGID, AND IMSID MISSING
8	OPS1302	REPLY TEXT KEYWORD MISSING
12	OPS1303	WTOR NOT FOUND
16	OPS1304	AUTHORIZATION CHECK FAILED
20	OPS1305	MATCHING WTOR COUNT GREATER THAN MAX
24	OPS1306	REPLY TEXT LENGTH GREATER THAN MAX
80	OPS1320	TSO/E NOT INSTALLED
84	OPS1321	COMMAND BUFFER PARSE ERROR
88	OPS1322	COMMAND PROCESSOR NOT AUTHORIZED
92	OPS1323	AUTHORIZATION USER EXIT ABENDED
96	OPS1324	REPLY CONTROL BLOCK LOOP ERROR
100	OPS1325	SERIOUS SYSTEM MACRO ERROR
104	OPS1326	SERIOUS CONTROL BLOCK ERROR
n/a	OPS1350	WRITE CURRENT REPLY REQUEST ONTO THE SYSTEM LOG

OPSREQ Command Processor—Invoke AOF Request Rules

The OPSREQ command processor invokes AOF request (REQ) rules. When OPSREQ executes, it scans the production rule set of the AOF for a REQ rule with an event specification that matches the request code. If it finds one or more matches, OPSREQ executes the processing section of that rule or rules.

For more information about REQ rules, see the *AOF Rules User Guide*.

Use one of the following formats for the OPSREQ command processor.

- This is a valid format for OPSREQ:

```
OPSREQ
{requestcode}
[text]
```

- This is also a valid format for OPSREQ:

```
OPSREQ
{CODE(requestcode)}
[TEXT(text)]
[SUBSYS(ssid)]
```

requestcode

Specifies the one- to ten-character name of the request event, which OPSREQ compares to the request event identifiers of any enabled request rules. The requestcode must begin with an alphabetic or national character.

TEXT

(Optional) Specifies text that can be passed to request rules.

Note: When specifying the TEXT string, the sum of the lengths of the request code, the text string, and 1 cannot exceed 128 characters.

SUBSYS

(Optional) For more information on the SUBSYS keyword, see Specify a Subsystem ID on a POI Command Processor in this chapter.

Security and the OPSREQ Command Processor

The security rules, security exit, or both execute whenever the OPSREQ command processor is used.

To successfully use the OPSREQ command processor, one of these requirements must be met:

- You must have TSO OPER authority.
- You must be allowed to issue OPSREQ through an OPSREQ-type security rule (which uses the)SEC OPSREQ* format) or through the security user exit (which is called OPUSEX).

OPSREQ Return Codes

OPSREQ produces these return codes:

0

Request completed normally

4

Required keyword missing or invalid

8

Current request did not execute rules

20

Request length greater than maximum

24

Some type of execution error occurred

84

Command buffer parse error

88

Main product address space not active

92

Parameter list error

96

Invalid command buffer format

100

Serious product control block error

104

Main product address space terminated

108

Serious system or macro error

184

Abend failure

WARNING! Request (REQ) rules are allowed to return arbitrary numeric values using the REXX RETURN statement. To avoid confusion, you should code your REQ rules to avoid the use of return values of less than 500. This prevents any future changes to the internal return codes of the product from conflicting with your automation.

OPSRMT Command Processor—Issue TSO Commands to Other Systems

The OPSRMT command enables you to issue a TSO command to other systems from an OPS/REXX program, a TSO/E REXX program, a CLIST, a TSO terminal, or from a console logged onto the CA OPS/MVS ECF component. The command can contain up to 255 characters.

Abbreviation: OR is the abbreviation of OPSRMT.

OPSRMT provides a *positional* format and an *operand* format for issuing a command.

- The positional format for the OPSRMT command processor:

OPSRMT

You must specify both keywords in this order:

{sysid*}

{command}

When using the positional format, OPSRMT considers all text beyond *sysid* to be part of the command text.

Example: The TSO command text is LISTBC:

OPSRMT SYS2 LISTBC

Example: The TSO command text is LISTCAT LEVEL(SYS1):

OR SYSTEM3 LISTCAT LEVEL(SYS1)

- The operand format for the OPSRMT command processor:

OPSRMT

{sysid*}

{COMMAND('cmdtext')}

[DELAY(seconds)]

[MAXLINES(1024,lines)]

[NOCLIST]

[OUTPUT|NOOUTPUT]

[SUBSYS(OPSS,ssid)]

[SYSTEM(msfids|ALL|EXT)]

[SYSWAIT(seconds)]

[WAIT(seconds)]

When using the operand format, the *cmdtext* specified on the COMMAND keyword may or may not be enclosed in single quotes.

Example: The TSO command text is OI REXXPGM1:

OPSRMT SYS2 COMMAND('OI REXXPGM1')

Example: The TSO command text is also OI REXXPGM1:

OPSRMT SYS2 COMMAND(OI REXXPGM1)

sysid

The *sysid* operand specifies the MSF system IDs of those copies of CA OPS/MVS to which the command should be routed. Routing is done by the subsystem on the local system, from where the command was issued, which was specified on the SUBSYS keyword (or OPSS by default).

Values for the *sysid* operand are:

Indicates that only the local system should receive the command. Specify this when you are not using the CA OPS/MVS MSF.

sysname

Indicates the MSF system ID of a single MSF-connected system (remote or local) that should receive the command.

ALL

Indicates that all active MSF-connected systems should receive the command, including the local system.

EXT

Indicates that all active MSF-connected systems other than the local system should receive the command.

Default: *sysid*. It must be explicitly specified.

Note: If more than one system is specified, then the NOOUTPUT keyword is forced.

DELAY

(Optional) Specifies the number of seconds CA OPS/MVS waits before issuing the command. The delay is applied before each command that is to be issued, even if OPSRMT executes in subcommand mode. You can specify a value from 1 to 300 seconds.

MAXLINES

(Optional) Specifies the maximum number of output lines that can be returned by OPSRMT. The default value is 1024. You can specify a value from 100 to 32767 lines.

If the NOOUTPUT keyword is also specified, this keyword will be ignored.

NOCLIST

(Optional) Suppresses the generation of the &SYSOUTLINE*n*W variables that are usually produced when &SYSOUTTRAP is set. &SYSOUTLINE*n* variables are still produced by TSO.

OUTPUT and NOOUTPUT

(Optional) Determines whether command output is returned. If NOOUTPUT is specified, then no command output is returned. If OUTPUT is specified, then command output is returned in the manner appropriate for the caller. OUTPUT is the default.

SUBSYS

(Optional) For more information on the SUBSYS keyword, see Specifying a Subsystem ID on a POI Command Processor in this chapter.

SYSTEM

(Optional) For more information on the SYSTEM keyword, see Specifying an MSF System ID on a POI Command Processor in this chapter.

Note: If more than one system is specified, then the NOOUTPUT keyword is implied.

SYSWAIT

(Optional) Specifies how many seconds OPSRMT waits for output from the command it issues, due to MSF network delay time between the local (source) system and the remote (target) system. You can specify a wait time between 0 and 600 seconds. The default is the value that was specified on the DELAY keyword for the ADDRESS OPSCCTL MSF DEFINE command issued on the local (source) system to define the remote (target) system, or the CA OPS/MVS MSFSYSWAIT parameter if the DELAY keyword was not specified.

Unless the WAIT keyword is also specified, the wait time specified on the SYSWAIT keyword is conditional; that is, the response may be terminated before the number of seconds specified if the actual command response has terminated. Contrast with the WAIT keyword, which is used to specify unconditional wait time.

WAIT

(Optional) Specifies how many seconds OPSRMT waits for output from the command it issues. You can specify a value from 1 to 60. The default is the value of the CA OPS/MVS OCWAIT parameter. After the specified number of *seconds* elapse, OPSRMT considers the command to have completed and ignores any additional command output.

The wait time specified on the WAIT keyword is unconditional; that is, the response is not terminated until the complete number of *seconds* specified has elapsed. Contrast with the SYSWAIT keyword. If both the WAIT and SYSWAIT keywords are specified, then the SYSWAIT time is added to the WAIT time for a cumulative unconditional wait time.

The WAIT and NOOUTPUT keywords are mutually exclusive.

Using OPSRMT in Subcommand Mode

Using the OPSRMT command processor without specifying TSO command text causes OPSRMT to execute in subcommand mode, which operates similar to the OPSCMD command processor in subcommand mode. The only difference is that OPSRMT processes TSO commands and OPSCMD processes system commands. For a description of OPSCMD subcommand mode, see OPSCMD Command Processor in this chapter. When executing in subcommand mode, OPSRMT routes all TSO commands you enter to the systems specified on the *sysid* operand or the SYSTEM keyword.

OPSRMT Return Codes

OPSRMT produces these return codes:

Return Code	Message Number	Message Text/Description
0	n/a	Command execution successful
1	n/a	Command output terminated due to MAXLINES keyword (WAIT keyword not specified)
2	n/a	Command output terminated due to MAXLINES keyword (WAIT keyword specified)
4	OPS3601	COMMAND GENERATED NO OUTPUT
8	OPS3602	OUTPUT RETRIEVAL MECHANISM DOWN
12	OPS3603	NO MVS/JES/VM COMMAND ENTERED
16	OPS3604	MVS/JES/VM CMD LENGTH EXCEEDS MAXIMUM
28	OPS3607	AUTHORIZATION CHECK FAILED
36	OPS3609	WAIT CANNOT BE ENTERED WITH NOOUTPUT
88	OPS3622	TSO/E IS NOT INSTALLED
104	OPS1131	MAIN PRODUCT ADDRESS SPACE TERMINATED
168	OPS3640	COMMAND INPUT BUF ALLOCATION FAILED
184	OPS3646	USER EXIT ABEND FAILURE
200	OPS3650	COMMAND SCAN AREA ALLOCATION FAILED
204	OPS3651	COMMAND SCAN AREA RELEASE FAILED
216	OPS3654	INVALID COMMAND BUFFER FORMAT
224	OPS3656	IKJSCAN ERROR
240	OPS3660	CLIST VARIABLE ACCESS ERROR
244	OPS3661	WORD TOKENIZATION ERROR
248	OPS3662	SYSOUTTRAP CONVERSION ERROR
252	OPS3663	COMMAND BUFFER PARSE ERROR
256	OPS1127	MSF IS NOT INSTALLED
258	OPS1128	MSF NOT ACTIVE
260	OPS1110	SYSTEM ID INVALID
262	OPS1111	SYSTEM ID NOT ACTIVE

Return Code	Message Number	Message Text/Description
264	OPS3613	SYSTEM IS CONSERVE/VM

OPSSSETV Command Processor—Create or Update Global Variable Values

The OPSSSETV command processor creates a global variable or updates the value of a global variable.

You can either update the whole variable or only a substring portion of the variable.

The TOKEN keyword provides for a serialized fetch and update sequence in which an intervening update is detected.

This command has the following format:

OPSSSETV

Specify the following two required keywords in the order shown:

{*varname*}

{{*varvalue*}}

[CMDRESP(TERMINAL|NOWHERE)]

[DELAY(0,*seconds*)]

[OFFSET(*displacement*)]

[SUBSYS(OPSS,*ssid*)]

[SVRULES(YES|NO)]

[SYSTEM(*msfids*|ALL|EXT)]

[SYSWAIT(*seconds*)]

[TOKEN(*n*)]

[UPPERCASE(YES|NO)]

varname

Specify the complete name of the global variable that you want to create or update with a new value. This keyword must precede the *varvalue* keyword.

You can specify an unprefixed (Automate-format) variable name as the value of *varname*. The CA OPS/MVS product determines the appropriate stem prefix internally using the ATMLOCALSCOPEnn and ATMSHAREDSCOPEnn parameters.

varvalue

Specify the value that you want to assign to *varname*. This keyword must follow the *varname* keyword.

CMDRESP

(Optional) Indicates how CA OPS/MVS should handle the response from OPSSETV.
Values are:

TERMINAL

(Default) Displays the response at the terminal.

NOWHERE

Tells CA OPS/MVS that it should not issue any messages except error messages.
This option is provided to eliminate messages in REXX programs.

When you invoke your OPSSETV request as a TSO command, the default of CMDRESP is CMDRESP(TERMINAL).

When you invoke your OPSSETV request as an OPS/REXX or TSO/E REXX function, the default of CMDRESP is CMDRESP(NOWHERE).

DELAY

(Optional) Indicates the number of seconds that your OPSSETV request waits before it can be processed. You can specify any number of seconds from 1 to 300.

Default: 0

OFFSET

(Optional) Indicates the substring displacement, relative to 1, at which the variable value is to be placed. The length of the substring is the length of the new variable value. Undefined areas of a new or updated variable are padded with blanks.

Default: 1

SUBSYS

(Optional) For more information on the SUBSYS keyword, see Specifying a Subsystem ID on a POI Command Processor in this chapter.

SVRULES

(Optional) Indicates whether you want the status variable that OPSSETV sets to be processed by global variable rules. Values are:

YES

(Default) Indicates that you want rules to process the status variable.

NO

Indicates that you do not want rules to process the status variable.

SYSTEM

(Optional) For more information on the SYSTEM keyword, see Specifying an MSF System ID on a POI Command Processor in this chapter.

Note: If more than one system is specified or implied, and the local system is one of the systems that receives the request, CA OPS/MVS returns only the result for the local system.

SYSWAIT

(Optional) For a cross-system request, the SYSWAIT keyword specifies the number of seconds that CA OPS/MVS should wait for the remote system to receive and respond to the OPSSETV request. You can specify any number of seconds from 1 to 600.

Default: The value of the MSFSYSWAIT parameter.

TOKEN

(Optional) Use the TOKEN keyword if you want to update a variable only after verifying its current value.

For the value of *nn*, specify a token that is equal to the current value of the update counter of the variable. You can use the OPSGETV command processor to obtain a current value of the variable as well as the value of its update counter. The update counter of a new variable has a value of 0. The update counter value that OPSGETV returns is the value you should specify for TOKEN(*nn*).

Before CA OPS/MVS performs the update, it tries to match the value you specify for TOKEN(*nn*) against the current update count of the variable to ensure that the value of the variable has not changed since the time you obtained it using OPSGETV. If the values do not match, CA OPS/MVS does not perform the update.

The TOKEN keyword provides a serialization facility that is similar to the one provided by the compare and update facility of the OPSVALUE function.

There is no default.

UPPERCASE

(Optional) Determines whether the variable value specified is converted to uppercase characters.

Default: YES

OPSSSETV Invoked as a REXX Function

In addition to invoking OPSSSETV as a TSO command, you can invoke it as an OPS/REXX or TSO/E REXX function. The function will have a single argument equal to the OPSSSETV TSO command syntax. The value returned by the function will be either 1 (for REXX true) or 0 (for REXX false). The CA OPS/MVS product sets the OPSRC variable to the same return code that would have been generated if you had invoked OPSSSETV as a TSO command.

Examples: **OPSSSETV**

- This example sets the variable to the specified value if the current update counter is 2. If the counter is not 2, the update fails:

```
OPSSSETV GLOBAL0.XYZ (THIS IS A TEST) TOKEN(2)
```

- This REXX function updates a substring of the variable beginning at position 10 in the current value of the variable. If the current value is ABCD, positions 5 through 9 of the variable become blanks. The vrc variable is set to 1 (for REXX true). The OPSRC variable is set to the same return code that would have been set if you had invoked OPSSSETV as a TSO command:

```
vrc = OPSSSETV('GLOBAL0.XYZ (WORKS IN REXX) OFFSET(10)')
```

OPSSSETV Return Codes

OPSSSETV produces these return codes:

0

The command executed successfully.

4

The command failed due to errors.

12

The CA OPS/MVS product is not active.

40

You invoked the OPSSSETV request as a REXX function, and it included an invalid REXX function argument.

OPSSMTBL Command Processor—Maintain the Directory Table

The OPSSMTBL command processor enables you to maintain the directory table that the CA OPS/MVS System State Manager feature uses to manage tables containing information about system resources. With OPSSMTBL, you can modify the directory table, change some of the attributes of the resource information tables in the directory, and return the names of resource information tables into variables or as messages on your TSO terminal.

Alias: STATETBL is currently an accepted alias of OPSSMTBL. This alias can be removed in future releases of CA OPS/MVS.

For more information on the OPSSMTBL command processor and using it with System State Manager, see the *User Guide*.

This command has the following format:

```
OPSSMTBL
[ADD(tablenames)]
[CREATE]
[CHANGE(tablenames)]
[MODE(ACTIVE,mode)]
[ACTION(actiontablename)]
[CMDRESP(destination)]
[DELETE(tablenames)]
[LIST(tablenames)]
[NOOUTPUT]
[POST(tasknames)]
[PREFIX(SSMTBL,prefix)]
[UP(newstatename)]
[DOWN(newstatename)]
[UNKNOWN(newstatename)]
[STATETBL(dirtablename)]
SUBSYS(OPSS,ssid)
SYSTEM(msfid)
SYSWAIT(seconds)
TNGELIGIBLE(YES|NO)
```

Note: Specifying a *tablenames* value for the LIST keyword is optional.

ADD

(Optional) The *tablenames* you specify are 1- to 18-character names of resource information tables to be added to the directory table of the System State Manager. If you specify multiple resource information tables, separate them with commas. When you specify a table, CA OPS/MVS checks to see whether the column names and attributes of that table conform to the System State Manager table requirements; if the table does not meet the requirements, CA OPS/MVS issues an error message.

CREATE

(Optional) If any of the resource information tables specified with the ADD keyword do not exist and you specify the CREATE keyword, CA OPS/MVS creates a model table with the name of the nonexistent table. The model table contains the columns required by the System State Manager. You can then use the SQL INSERT command or the Relational Data Framework (RDF) table editor to insert rows for each resource that you want the System State Manager to manage. The structure of any SSM table created by this keyword always matches the column requirements for the highest supported version of SSM.

CHANGE

(Optional) This keyword specifies the 1- to 18-character names of resource information tables whose processing modes you want to change. If you specify multiple table names, then separate them with commas.

MODE

(Optional) Specified with the ADD or CHANGE keyword, the MODE keyword defines how the System State Manager processes the resource information table that you are adding. Valid mode values are:

ACTIVE

The System State Manager is monitoring this table and acts when a resource changes state. ACTIVE is the default mode.

NOPREREQ

Same as ACTIVE mode, but the System State Manager does not perform prerequisite resource checking.

PASSIVE

The System State Manager is monitoring this table but does nothing when the state of a resource changes.

INACTIVE

The System State Manager is not monitoring this table.

ACTION

(Optional) Used with the ADD or CHANGE keyword, the ACTION keyword specifies the action table that you want to associate with a resource information table.

For the ADD table operation with a specified action table, the column structure of the action table is checked to verify that it conforms to the required standards of the version of the SSM task that is currently running.

CMDRESP

(Optional) This keyword specifies how CA OPS/MVS returns its response to the OPSSMTBL LIST command.

Possible values for destination are **TERMINAL**, **CLIST**, **REXX**, **XDQ**, and **NOWHERE**.

- If you specify the default **CMDRESP(TERMINAL)**, CA OPS/MVS displays the output from the OPSSMTBL LIST command on your terminal.
- If you issue the OPSSMTBL LIST command from a TSO/E REXX program or from a CLIST, specify **CMDRESP(CLIST)** or **CMDRESP(REXX)** if you want the response to go only into CLIST or REXX variables.
- If you issue OPSSMTBL as an OPS/REXX function, you can specify **CMDRESP(XDQ)** to route all error messages to the external data queue.

Note: If you issue the OPSSMTBL command from in an OPS/REXX program through the OI or OX commands, the result is always returned to the REXX external data queue in the terminal message format.

- Specifying **CMDRESP(NOWHERE)** is the same as specifying the **NOOUTPUT** keyword. No response is produced.

The optional CMDRESP keyword determines whether the System State Manager creates the response variables described here:

- Variable contains the number of resource information tables listed in the directory table

CMDRESP(CLIST) variable name: prefix

CMDRESP(REXX) variable name: prefix.0

- Variable contains the resource information table name

CMDRESP(CLIST) variable name: prefixn

CMDRESP(REXX) variable name: prefix.n

- Variable contains the resource table mode

CMDRESP(CLIST) variable name: prefixnMODE

CMDRESP(REXX) variable name: prefix_MODE.n

- Variable contains the up-state name

CMDRESP(CLIST) variable name: prefixnUP

CMDRESP(REXX) variable name: prefix_UP.n

- Variable contains the down-state name

CMDRESP(CLIST) variable name: prefixnDOWN

CMDRESP(REXX) variable name: prefix_DOWN.n

- Variable contains the unknown-state name

CMDRESP(CLIST) variable name: prefixnUNKNOWN

CMDRESP(REXX) variable name: prefix_UNKNOWN.n

- Variable contains the action table name

CMDRESP(CLIST) variable name: prefixnACTION

CMDRESP(REXX) variable name: prefix_ACTION.n

- Variable contains the TNGELIGIBLE value

CMDRESP(CLIST) variable name: prefixnTNG

CMDRESP(REXX) variable name: prefix_TNG.n

Note: The default value for prefix is SSMTBL.

DELETE

(Optional) This keyword specifies the 1- to 18-character names of resource information tables to be deleted from the list of managed tables of the System State Manager. If you specify multiple table names, separate them with commas.

LIST

(Optional) The LIST keyword lists the resource information tables that the System State Manager is currently managing. The optional tablenames value allows you to list the resource information for a specific table or tables.

NOOUTPUT

(Optional) Specify the NOOUTPUT keyword if you do not want a response from the command.

POST

(Optional) The tasknames provided are predefined names of SSM-related subtasks that typically enter a wait state until a recognized event occurs, such as the update of an SSM-monitored column in an SSM resource table.

The POST keyword activates the specified tasks without creating a recognized event. If you specify multiple task names, then separate them with commas. Valid task names are:

- SSM-Post the SSM main task to initiate an SSM resource scan.
- SSMSUB-Post the SSM subtask to execute the associated AOF request rule specified in the parameter SSMSUBRULE.

PREFIX

(Optional) If you issue the OPSSMTBL LIST command with the CMDRESP keyword, CA OPS/MVS generates arrays of CLIST or REXX variables. The variables contain values corresponding to the names and attributes of the resource information tables managed by System State Manager. The PREFIX keyword supplies the prefix for these variables. When CA OPS/MVS generates the variable array, it creates a variable named prefix (SSMTBL by default), which stores a count of resource information table names (as shown in the table in the CMDRESP keyword description).

UP

(Optional) This keyword specifies the new one- to eight-character value that you have chosen to replace the default up-state value of UP.

DOWN

(Optional) This keyword specifies the new one- to eight-character value that you have chosen to replace the default down-state value of DOWN.

UNKNOWN

(Optional) This keyword specifies the new one- to eight-character value that you have chosen to replace the default unknown-state value of UNKNOWN.

STATETBL

(Optional) Use the STATETBL keyword to specify the name of the System State Manager resource directory table. The default is the value of the STATETBL parameter on the system that is the target of the command.

SUBSYS

(Optional) For more information on the SUBSYS keyword, see Specifying a Subsystem ID on a POI Command Processor in this chapter.

SYSTEM

(Optional) For more information on the SYSTEM keyword, see Specifying an MSF ID on a POI Command Processor in this chapter.

Note: You may specify a single system only.

SYSWAIT

(Optional) Used in conjunction with the SYSTEM keyword, the SYSWAIT keyword specifies the number of seconds that the CA OPS/MVS product waits for remote SQL responses. The range is 1 to 300; the default is the MSFSYSWAIT value.

TNGELIGIBLE

(Optional) Allows the TNGELIGIBLE column used by the CA Event Manager to be set to YES or NO when it exists in the resource directory table. If the column does not exist in the table, this keyword is ignored. The default is NO.

OPSSMTBL Invoked as a REXX Function

In addition to invoking OPSSMTBL as a TSO command, you can invoke it as an OPS/REXX function. The function will have a single argument equal to the OPSSMTBL TSO command syntax. The value returned by the function will be the same as the return code that would have been generated if you had invoked OPSSMTBL as a TSO command.

Example: **OPSSMTBL Invoked as a REXX Function**

```
trc = OPSSMTBL("LIST CMDRESP(REXX)")
```

OPSSMTBL Return Codes

The OPSSMTBL command processor generates the following return codes:

0

COMMAND COMPLETED SUCCESSFULLY

4

A COMMAND FUNCTION FAILED FOR ONE OR MORE TABLES

8

USER NOT AUTHORIZED TO USE THIS COMMAND

12

AUTOMATE IS NOT ACTIVE FOR CROSS SYSTEM USE

20

LOAD OF THE AUTOMATE STATETBL COMMAND FAILED

44

SYSTEM ID SPECIFIED IS INVALID OR UNUSABLE

48

THE OPSSMTBL OPS/REXX FUNCTION MAY NOT BE USED FOR CROSS SYSTEM
FUNCTIONS IN AN AOF RULE

80

TSO/E IS NOT INSTALLED. IT IS REQUIRED

84

COMMAND BUFFER PARSE ERROR. BAD SYNTAX

88

CA OPS/MVS IS NOT ACTIVE

92

AUTHORIZATION USER EXIT ABENDED

96

THE FORMAT OF THE COMMAND BUFFER IS NOT DECIPHERABLE

100

SERIOUS ERROR ENCOUNTERED IN A SYSTEM CONTROL BLOCK

104

CA OPS/MVS TERMINATED DURING THE COMMAND FUNCTION

108

SERIOUS ERROR IN A SYSTEM SERVICE ROUTINE

116

FAILURE OCCURRED IN A CLIST/REXX VARIABLE ROUTINE

184

ABEND OCCURRED IN THE OPSSMTBL FUNCTION ROUTINE

OPSVASRV Command Processor

Definition and Usage

The OPSVASRV command processor creates and manipulates sysplex variables that are defined in the CA CCS Common VariableService (CAVARSRV) component. These variables are independent from the standard CA OPS/MVS global variables. Currently these CCS CAVARSRV sysplex variables are only stored in main storage of the CCS CAVARSRV component. The CCS CAVARSRV component is active on each system within the sysplex, providing a tool to any CA OPS/MVS automation that needs to store and manipulate data within the boundaries of a sysplex through the OPSVASRV command processor. Since the variables exist on multiple systems, they will be available within the sysplex as long as one system in the sysplex remains active. As a system re-initializes after an IPL, the CCS CAVARSRV address space on that system will synchronize variables from an active CCS CAVARSRV on another sysplex system.

Installation Requirements

The CA CCS Common VariableService component must be installed on each system in which CA OPS/MVS automation will be creating and manipulating sysplex variables. Refer to CCS Common VariableService (CAVARSRV) documentation for complete details on installing and utilizing this CCS component.

No special installation requirements or parameters are needed within CA OPS/MVS to use this function. However, the GLVNOTIFYRULES CA OPS/MVS parameter needs to be set to YES, if AOF GLV rules will be created to process specific sysplex variables.

Utilization Environments

The OPSVASRV() function can be utilized within specific AOF rules, (ARM, API, and REQ), OPS/REXX programs, TSO/E programs, and OMVS scripts. Additionally, an OPSVASRV command processor version of this function exists, and it can be utilized from the TSO foreground or within a batch if needed. For more information, see the [POI Command Process](#) (see page 689) for details on invoking the OPSVASRV command processor.

Coding Syntax

OPSVASRV function supporting keyword

Example from the TSO foreground:

```
OPSVASRV GET NAME(GLVFLXTA.PROD.PLEXCICS.STATUS)
```

For more information, see the [Sample coding statements](#) (see page 665) section for more coding statements.

NOTE: With the exception of the CMDRESP() supporting keyword as described here, all functions and supporting keywords of the OPSVASRV command processor, work as documented in the OPSVASRV() function. For more information, see the [OPSVASRV\(\) function](#) (see page 654) for a complete list and description of the available functions and supporting keywords.

Supporting keyword.

CMDRESP(option)

The CMDRESP keyword specifies how the output from the OPSVASRV command processor is handled. Values are:

TERMINAL

Displays the first 100 characters of a message line back to the issuing terminal. A plus sign (+) at the end of the output line indicates the truncation of the line.

CLIST

Creates CLIST variable output with the same names as the REXX variable output created from the OPSVASRV() except for REXX stem variables. The period character in the REXX stem variables is omitted in the CLIST variables.

NOWHERE

Tells CA OPS/MVS not to issue any messages except error messages.

Sample Coding Statements

From TSO command line to create a variable called GLVPLXTA.PLEXVAR and set to some value:

```
OPSVASRV CREATE NAME(GLVPLXTA.PLEXVAR) DATAVAL('THIS IS VAR VALUE')
```

From TSO command line to Update a variable called GLVPLXTA.PLEXVAR to some value:

```
OPSVASRV UPDATE NAME(GLVPLXTA.PLEXVAR) DATAVAL('New value')
```

From TSO command line to obtain a variable called GLVPLXTA.PLEXVAR:

```
OPSVASRV GET NAME(GLVPLXTA.*)
```

From TSO command line to delete a variable called GLVPLXTA.PLEXVAR:

```
OPSVASRV DELETE NAME(GLVPLXTA.PLEXVAR)
```

From TSO command line to obtain a mask list of all GLVPLXTA.* variables:

```
OPSVASRV GETLIKE NAME(GLVPLXTA.*)
```

From batch job to create a variable called GLVPLXTB.V1 with a value of 0:

```
//USERIDJ JOB (123300000),'OPSVASRV',
//      CLASS=A,MSGCLASS=1,NOTIFY=&SYSUID
//*JOBPARM SYSAFF=SYSA
//  SET P="CREATE,NAME(GLVPLXTB.V1) DATAVAL(0)"
//RUN  EXEC PGM=OPSVASRV,PARM=&P
//STEPLIB DD DISP=SHR,DSN=OPSNDVR.MOTM.OPS.BASE.P2.LOADLIB
```

From batch job to update a variable called GLVPLXTC.PLEXCICSA to a value of MOVE:

```
//USERIDJ JOB (123300000),'OPSVASRV',
//      CLASS=A,MSGCLASS=1,NOTIFY=&SYSUID
//*JOBPARM SYSAFF=SYSA
//  SET P="UPDATE,NAME(GLVPLXTC.PLEXCICSA) DATAVAL(MOVE)"
//RUN  EXEC PGM=OPSVASRV,PARM=&P
//STEPLIB DD DISP=SHR,DSN=OPSNDVR.MOTM.OPS.BASE.P2.LOADLIB
```

Return and Reason Codes

For more information, see the [OPSVASRV\(\) function](#) (see page 654) documentation for a complete list of these codes.

OPSVIEW Command Processor—Access OPSVIEW

The OPSVIEW command processor provides access to OPSVIEW, the CA OPS/MVS full-screen, interactive application for controlling CA OPS/MVS itself, z/OS, and JES. For information about OPSVIEW and its options, see the *OPSVIEW User Guide*.

Note: Most OPSVIEW applications work only if the CA OPS/MVS started task is active. For a description of how to set up CA OPS/MVS, see the *Administration Guide*.

The OPSVIEW command processor has the following format:

```
OPSVIEW  
[option]  
[SYSID(msfid)]  
[SUBSYS(OPSS,ssid)]  
[COMMAND(commandtext)]
```

option

(Optional) Specifies the OPSVIEW option that executes first. Using this operand causes the CA OPS/MVS product to bypass the OPSVIEW Primary Options Menu.

To display the Operator Server Facility Panel, issue this command:

```
OPSVIEW 4.3
```

SYSID

(Optional) Names the system to execute all OPSCMD command processors issued by this OPSVIEW session. You may specify a single MSF system ID only. If you specify no msfid, then the default is the local system.

SUBSYS

(Optional) For more information on the SUBSYS keyword, see Specify a Subsystem ID on a POI Command Processor in this chapter.

COMMAND

(Optional) The *commandtext* value is the text of a z/OS or JES command to be executed. If you specify *commandtext*, it overrides the value of the *option* operand, because option 6 is implied.

OPSWAIT Command Processor—Suspend CLIST or REXX Processing

The OPSWAIT command processor enables you to suspend processing of a CLIST or REXX program until one of the following time events occurs:

- A time interval
- A specific time of day
- An event time interval

OW is the abbreviation of OPSWAIT.

The OPSWAIT command processor has the following format:

```
OPSWAIT  
{FOR(interval)|UNTIL(hh:mm:ss)|SYNCH(hh:mm:ss)}  
[MAXTIME(hh:mm:ss)]
```

FOR

Use this keyword to delay CLIST or REXX program processing for a specific amount of time, either a number of seconds or an *hh:mm:ss* (hours, minutes, seconds) value. The following examples show how to specify these intervals:

- To wait for half a second, specify the following value:
`FOR(.5)`
- To wait for 23 seconds, specify the following value:
`FOR(23)`
- To wait for two hundred seconds, specify the following value:
`FOR(200)`
- To wait for five minutes and 25 seconds, specify the following value:
`FOR(5:25)`
- To wait for two hours and 15 minutes, specify the following value:
`FOR(2:15:00)`

Note: When specifying an interval as seconds without colons, you can use up to two decimal places to indicate fractions of seconds.

UNTIL

Use this keyword to delay CLIST or REXX program processing until a specific time of day. If the specified time has already passed, the OPSWAIT command either terminates or waits until the correct time of day tomorrow, depending on the value of the MAXTIME keyword. For example, suppose that the current time is 13:23:00:

- To wait for six minutes, specify the following value:

`UNTIL(13:29:00)`

- To wait for 23 hours and 52 minutes, specify the following value:

`UNTIL(13:15) MAXTIME(23:59:59)`

- To not wait at all, specify the following value:

`UNTIL(13:15) MAXTIME(2:00:00)`

SYNCH

Use this keyword to delay CLIST or REXX program processing until the start of a specified time interval. This interval can be the start of the next minute, next five-minute period, next hour, and so on. For example, suppose that the current time is 13:23:00:

- To wait until the next 30-second interval starts (at 13:23:30), specify the following value:

`SYNCH(30)`

- To wait until the next minute begins (at 13:24:00), specify the following value:

`SYNCH(1:00)`

- To wait until the next two-hour interval starts (at 14:00:00), specify the following value:

`SYNCH(2:00:00)`

MAXTIME

(Optional) Use this keyword to define the maximum amount of time that the OPSWAIT command is allowed to delay CLIST or REXX program processing by specifying either a value in *hh:mm:ss* format or the number of seconds (with up to two decimal places to indicate fractions of seconds).

If the command exceeds the specified maximum wait time, then it does not wait and a return code 8 is issued.

If this keyword is not specified, then the default maximum wait time of 8 hours is used.

OPSWAIT Used Without Event Keywords

If you omit the FOR, UNTIL, or SYNCH keyword, CA OPS/MVS reads the first value after OPSWAIT as a wait time (equivalent to the FOR keyword).

Example: OPSWAIT without Event Keywords

The following command causes a CLIST or REXX program to wait for 10 minutes and 36 seconds:

```
OPSWAIT 10:36
```

OPSWAIT Return Codes

OPSWAIT produces these return codes:

Return Code	Message Number	Message Text
0	N/A	COMMAND EXECUTION SUCCESSFUL
4	OPS1201	NO WAIT EVENTS ENTERED
8	OPS1202	MAXIMUM WAIT TIME EXCEEDED
12	OPS1203	INVALID TIME VALUE
20	OPS1204	INVALID UNTIL VALUE
84	OPS19xx	COMMAND BUFFER PARSE ERROR
88	OPS9521	INVALID COMMAND BUFFER FORMAT

OPSWTO Command Processor—Issue WTO and WTOR Messages

The OPSWTO command processor issues one line and multiline WTO and WTOR messages from an OPS/REXX program, a TSO/E REXX program or CLIST, or a TSO READY prompt.

Important! Quotes are valid only around text values, as shown here: *'messageline1'*. Using quotes around any other keyword value can cause an error.

This command has the following format:

OPSWTO

Specify *one* keyword from the following required keywords:

{TEXT(*messageline1*)|TEXT(*messageline1* through *messageline 255*)}

TEXTVAR(*prefix*)}

[AREAID(*consoleareaid*)]

[CMDRESP(CLIST|REXX)]

[CONNAME[CONNAME(*consolenames*)]]

[DELAY(*delaytime*)]

[DESC(*descriptor codes*)]

[HLITE|LOWLITE]

[KEY(*amrf key*)]

[LINETYPE(*linetype*)]

[MCSFLAGS(*flag values*)]

[MSGID(*messageid*)]

[OUTDELIM(*delimstring*,YES|NO)]

[PREFIX(*prefix*)]

[REPLY]

[ROUTE|ROUTECD(*route codes*)]

[SUBSYS(OPSS,*ssid*)]

[SYSTEM(*msfids*|ALL|EXT)]

[SYSWAIT(*seconds*)]

[TOKEN(*dom token*)]

[UPPERCASE(YES|NO)]

[WAITREPLYWAIT(300,*seconds*)]

[WTOD]

AREAID

(Optional) AREAID specifies the console area ID to display the message. You must use this keyword to specify the console to receive the message. You also must specify the DESC keyword with a value of 8 or 9.

You cannot use the AREAID keyword with the REPLY keyword.

CMDRESP

(Optional) When a CLIST or TSO/E REXX program issues the OPSWTO command processor, the CMDRESP keyword enables you to save the response to a WTO or WTOR message in CLIST or REXX variables. If you specify CMDRESP(CLIST), the response goes into CLIST variables. If you specify CMDRESP(REXX), the response goes into REXX stem variables. The default when executing under a TSO/E REXX or CLIST program is CMDRESP(CLIST). When executing under an OPS/REXX program, the default is to generate no variables.

If you specify a value for CMDRESP, OPSWTO does the following:

- Generates DOMID variables when you issue a WTO message with descriptor codes 1, 2, 3, or 11
- Generates reply variables for WTOR response text

OPS/REXX programs ignore the CMDRESP keyword and place responses to WTOR messages in the external data queue. For WTO messages, you can get the DOMID from the external data queue through the WTOID keyword.

The following lists the DOMID variables that the CMDRESP keyword generates for WTO messages:

- Variable contains the number of DOMID variables created
CMDRESP(CLIST) Variable Name: DOMID0
CMDRESP(REXX) Variable Name: DOMID.0
- Variable contains the first DOMID returned
CMDRESP(CLIST) Variable Name: DOMID (equivalent to DOMID1)
CMDRESP(REXX) Variable Name: no REXX equivalent
- Variable contains individual DOMID for console
CMDRESP(CLIST) Variable Name: DOMID n
CMDRESP(REXX) Variable Name: DOMID. n
(n is the count of DOMIDs)

The following lists the REPLY variables that the CMDRESP keyword generates for WTOR messages:

- Variable contains the number of reply lines
CMDRESP(CLIST) Variable Name: no CLIST equivalent
CMDRESP(REXX) Variable Name: REPLY.0
- Variable contains the text of the message reply
CMDRESP(CLIST) Variable Name: REPLY
CMDRESP(REXX) Variable Name: REPLY. n
(n is the count of reply lines)

- Variable contains the number of words in the reply
CMDRESP(CLIST) Variable Name: REPLYW
CMDRESP(REXX) Variable Name: REPLY.*n*.0
- Variable contains the *m*th word of the reply text
CMDRESP(CLIST) Variable Name: REPLYW*m*
CMDRESP(REXX) Variable Name: REPLY.*n*.*m*

CNNAME and CONNAME

(Optional) Either the CNNAME keyword or the CONNAME keyword specifies the names of the consoles that receive the WTO or WTOR message. Specify up to 16 alphanumeric console names containing up to eight characters each.

DELAY

(Optional) The DELAY keyword defines how long OPSWTO suspends processing before executing the current command.

Specify any delay time between 1 and 300 seconds. DELAY has no default value.

DESC

(Optional) Use the DESC keyword to specify message descriptor codes for the current message. Descriptor codes are numbers between 1 and 16; codes 1 through 6 and descriptor code 11 are mutually exclusive.

You can assign codes 7 through 10 in combination with any other code, but each code must be unique. If you specify more than one code, separate them with commas as shown in the following example:

```
DESC(1,7)
```

You can also enter any of the following descriptor code words:

- SYSFAIL - Equivalent to descriptor code 1
- IMEDACTN - Equivalent to descriptor code 2
- EVENACTN - Equivalent to descriptor code 3
- SYSSTAT - Equivalent to descriptor code 4
- IMEDCMD - Equivalent to descriptor code 5
- JOBSTAT - Equivalent to descriptor code 6
- APPLPRGM - Equivalent to descriptor code 7
- OOLMSG - Equivalent to descriptor code 8
- OPERREQ - Equivalent to descriptor code 9
- DYNSTAT - Equivalent to descriptor code 10
- CRITEVET - Equivalent to descriptor code 11
- HILITE - Equivalent to descriptor code 1

Notes:

- If numeric descriptor codes are used, leading zeros are not permitted.
- There is no support for descriptor codes when issuing a WTO with REPLY. Use of the DESC keyword and the REPLY keyword together results in an error condition and return code 12.

HILITE and LOWLITE

(Optional) These keywords determine how the WTO or WTOR message displays on the console that receives it. Specify HILITE to send a high-intensity, non-scrollable WTO or WTOR, or specify LOWLITE to make the message scrollable and displayed in low intensity.

KEY

(Optional) The KEY keyword specifies a key name that can be used to display messages that the z/OS AMRF feature retains (messages with descriptor codes 1, 2, 3, and 11). You can display AMRF messages using the following z/OS command:

`DRKEY=keyname`

Note: The value of *amrf key* must consist of 1 to 8 EBCDIC characters. It will be left-justified and padded with blanks to create an 8-character key value. The value of *amrf key* cannot be an asterisk (*).

LINETYPE

(Optional) Use the LINETYPE keyword to designate control and label lines in a multiline WTO or WTOR message. Possible values for this keyword are:

- C - Message line 1 is a control line.
- CL - Message line 1 is a control line and line 2 is a label line.
- CLL - Message line 1 is a control line and lines 2 and 3 are label lines.
- L - Message line 1 is a label line.
- LL - Message lines 1 and 2 are label lines.

MCSFLAGS

(Optional) Use this keyword to enter MCS flag values, which are passed to the operating system. Each MCS flag is a unique character string. Following are sample MCS flags:

`MCSFLAGS(REGO)`

`MCSFLAGS(BRDCST)`

You can specify any of these flag values:

- RESP - The WTO is an immediate command response.
- REPLY - The WTO is a reply to a WTOR.
- BRDCST - Broadcast the message to all active consoles.
- HRDCPY - Queue the message for hard copy only.
- NOTIME - Do not append time to the message.
- NOCPY - If a program in the supervisor state issues the WTO or WTOR macro instruction, do not queue the message for hard copy. Otherwise, ignore this flag.
- CMD - The WTO is a recording to a system command issued for hard copy log purposes. You can use the CMD flag only with MVS/XA or MVS/ESA, which are operating systems no longer supported by IBM.

MSGID

(Optional) This keyword defines the message ID that prefixes the WTO or WTOR text. If you omit the MSGID keyword, the OPSWTO command processor uses a default message ID, OPS1371I. Other message IDs can contain up to 10 characters. Message IDs are not padded with blanks.

OUTDELIM

(Optional) Use this keyword to specify two values:

- The delimiter string used to generate tokens for words in WTOR response text
- YES or NO, depending on whether you want delimiter strings to generate word tokens

A blank is always a delimiter. The defaults are YES and any of the following delimiter characters:

`()/='`

Separate the two OUTDELIM operands with a comma, as shown in the following example:

```
OUTDELIM '@',NO)
```

PREFIX

(Optional) This keyword defines a prefix that becomes part of the names of variables generated when you specify the CMDRESP keyword, and you are issuing a highlighted message or a WTOR message. The default prefix for highlighted messages is DOMID. The default prefix for WTOR messages is REPLY.

REPLY

(Optional) Including this keyword in an OPSWTO command causes it to issue a WTOR message rather than a WTO. OPSWTO waits for the operator to respond (for the time specified with the WAIT keyword) and echoes any reply made with the TSO PUTLINE service. This is done so that a CLIST can pick the reply up using the TSO SYSOUTTRAP mechanism.

OPSWTO issues a return code showing whether it timed out or received a reply. The maximum reply length is 119 characters.

Notes:

- There is no support for descriptor codes when issuing a WTO with REPLY. Use of the DESC keyword and the REPLY keyword together results in an error condition and return code 12.
- The REPLY keyword and the TEXTVAR keyword are mutually exclusive.

ROUTE and ROUTECDE

(Optional) Use either keyword to enter route codes for the message. You can specify as many as 16 route codes, separated by commas.

A route code is any number between 1 and 128 or one of the following character strings:

Character String	Description
MSTRACTN	Equivalent to route code 1
MSTRINFO	Equivalent to route code 2
TAPEPOOL	Equivalent to route code 3
DASDPOOL	Equivalent to route code 4
TAPELIB	Equivalent to route code 5
DISKLIB	Equivalent to route code 6
UR	Equivalent to route code 7
TP	Equivalent to route code 8
SECURITY	Equivalent to route code 9
SYSERR	Equivalent to route code 10
PROGINFO	Equivalent to route code 11
EMULATOR	Equivalent to route code 12

SUBSYS

(Optional) For more information on the SUBSYS keyword, see Specifying a Subsystem ID on a POI Command Processor in this chapter.

SYSTEM

(Optional) For more information on the SYSTEM keyword, see Specifying an MSF System ID on a POI Command Processor in this chapter. See also the description of the SYSWAIT keyword.

SYSWAIT

(Optional) With this keyword, you can specify a number of seconds (from 1 to 300) of wait time to accommodate communication delays across systems. The CA OPS/MVS product adds the value of the SYSWAIT keyword to the MSFSYSWAIT value.

Note: See also the description of the SYSTEM keyword.

TEXT

This keyword specifies the text of a one line or multiline WTO or WTOR message.

To specify a one-line message, the text must meet these criteria:

- Contains up to 124 characters for a WTO or 124 characters for a WTOR message minus the length of the MSGID keyword.
- Uses a pair of quotes to indicate a single quote in the message text

Following are some sample text strings:

```
TEXT('this is a test value')
```

```
TEXT('put quotes in "double" quotes')
```

You can insert the contents of a REXX variable into the message text. For example, the following example inserts the contents of variable FIELD1 into the message text as a text string:

```
TEXT('||FIELD1||')
```

To specify a multiline message, the text must meet these criteria:

- Contains as many as 255 lines of message text, each using as many as 70 characters
- Each text string must be enclosed in quotes and separated by blanks or a comma

TEXTVAR

This keyword provides a REXX variable name prefix for a set of variables, each variable containing one line of a multiline WTO message.

OPSWTO appends an ascending index number (beginning with 1) to the value you specify for prefix. For example, if you specify TEXTVAR(PREFIX_DATA), the names of variables storing your WTO are PREFIX_DATA1, PREFIX_DATA2, and so on.

OPSWTO stops reading variables when it encounters a null variable value.

If the OPSWTO command processor is executing in a TSO/E REXX or CLIST environment, you can use this keyword instead of the TEXT keyword. If you authorize the OPSWTO program you must use a textvar variable starting with SYSAUTH in your TSO/E REXX or CLIST, else an RC = 4 is issued. You cannot use the TEXTVAR keyword if the command processor is executing in the OPS/REXX environment; use the ADDRESS WTO host environment instead.

Note: The TEXTVAR keyword and the REPLY keyword are mutually exclusive.

TOKEN

(Optional) This keyword specifies a one- to four-character name or a four-byte hexadecimal value, such as ABCD or X'0A204200'. This token value can be used in delete-operator-message (DOM) processing to delete all messages with the same token value.

UPPERCASE

(Optional) This keyword determines whether the message text specified with the TEXT keyword is converted to uppercase characters. YES is the default.

WAIT and REPLYWAIT

(Optional) The WAIT and REPLYWAIT keywords are identical. Each determines how long OPSWTO waits for a WTOR reply or for a timer to expire. OPSWTO terminates with a failure code if it does not receive a WTOR reply before the wait time expires. As soon as the wait time expires, OPSWTO deletes all WTOR messages.

Specify any wait time between 1 and 3600 seconds; the default is 300 seconds (five minutes).

WTOID

(Optional) This keyword causes CA OPS/MVS to issue message OPS1349I, which displays the message number assigned by z/OS. This message number is the delete-operator-message ID (DOMID) of the message in X'01234567' format; it is also the last word of the WTO or WTOR message.

STATESET Program—Set State Values

Use the STATESET program to change the current state and desired state values specified for a resource. For more information, including examples, see the chapter “Using System State Manager” in the *User Guide*.

You can use the following keywords and arguments to invoke STATESET:

```
STATESET  
{tablename.resourcename|resourcename}  
[ACTMODE(mode)]  
[CMDRESP(TERMINAL|NOWHERE|XDQ)]  
[CURRENT(currentstate)]  
[DESIRED(desiredstate)]  
[MODE(mode)]  
[PREMODE(mode)]  
[PREREQ(prereqstate)]  
[REFMODE(mode)]  
[SUBREQ(subreqstate)]  
[SUBSYS(OPSS,ssid)]  
[SYSTEM(msfids)]  
[SYSWAIT(seconds)]  
[TNGNOTIFY(ALWAYS|NEVER)]  
[WAIT(0,seconds)]
```


resourcename

This is the name of the resource as entered in the NAME column of the resource information table. If the resourcename appears in multiple tables, you can specify the resource name as tablename.resourcename to improve performance; otherwise, the System State Manager scans all tables for the resourcename that you specify.

The resourcename value can contain 1 to 18 characters. The tablename, if you specify one, can contain 1 to 18 characters.

ACTMODE

(Optional) Specifies the value to be placed in the ACTMODE column of the resource. Any one- to eight-character value may be specified.

CMDRESP

(Optional) Specifies where the messages generated by the STATESET program go. When using the STATESET program in an ISPF dialog, a NOWHERE value eliminates line mode output. TERMINAL is the default.

If you specify XDQ, response lines are put on the REXX external data queue with the QUEUE command.

Note: While the code currently only validates and uses the first character of the CMDRESP value, it is strongly recommended that you always specify the complete CMDRESP value when using STATESET in your programs.

CURRENT

(Optional) Specifies the value to be placed in the CURRENT_STATE column. This value might be UP, DOWN, STARTING, STOPPING, and so on.

DESIRED

(Optional) Determines the value to be placed in the DESIRED_STATE column. This value can be UP, DOWN, or UNKNOWN.

MODE

(Optional) Specifies the type of processing that the System State Manager uses for this resource. Valid mode values are:

- **ACTIVE** - The System State Manager is monitoring this resource and acts when the resource state changes.
- **NOPREREQ** - Same as **ACTIVE** mode, but the System State Manager does not check the state of prerequisite resources. Specify **NOPREREQ** mode when you want the System State Manager to force the resource to restart.
- **INACTIVE** - The System State Manager is not monitoring this resource.
- **PASSIVE** - The System State Manager is monitoring this resource but does nothing when its state changes.

Note: While the code currently only validates and uses the first character of the **MODE** value, it is strongly recommended that you always specify the complete **MODE** value when using **STATESET** in your programs.

PREMODE

(Optional) Specifies the value to be placed in the **PREMODE** column of the resource. **PREMODE** controls whether prerequisite and sub prerequisite processing is performed for the resource. Valid values for the **PREMODE** keyword are as follows:

- **ACTIVE** - Normal prerequisite and sub prerequisite processing are performed
- **PREREQ** - Only prerequisite processing is performed
- **SUBREQ** - Only sub prerequisite processing is performed
- **INACTIVE** - Neither prerequisite nor sub prerequisite processing is performed

PREREQ

(Optional) Defines the new one- to eight-character value to be placed in the **DESIRED_STATE** column of all resources that are prerequisites for the resource specified by `tablename.resourcename` or `resourcename`. Usually, this value is **UP** or **DOWN**.

REFMODE

(Optional) Specifies the value to be placed in the **REFMODE** column of the resource. **REFMODE** controls whether prerequisite and sub prerequisite references to this resource are evaluated or ignored. Valid values for the **REFMODE** keyword are as follows:

- **ACTIVE** - Prerequisite and sub prerequisite references are evaluated
- **PREREQ** - Only prerequisite references are evaluated
- **SUBREQ** - Only sub prerequisite references are evaluated
- **INACTIVE** - Neither prerequisite nor sub prerequisite references are evaluated

SUBREQ

(Optional) Specifies the new one- to eight-character value to be placed in the DESIRED_STATE column of all resources that have the resource specified by tablename.resourcename or resourcename as a prerequisite. This value is usually UP or DOWN.

SUBSYS

(Optional) For more information on the SUBSYS keyword, see Specifying a Subsystem ID on a POI Command Processor in this chapter.

SYSTEM

(Optional) For more information on the SYSTEM keyword, see Specifying an MSF System ID on a POI Command Processor in this chapter.

Note: You may specify a single system only.

SYSWAIT

(Optional) Specifies the number of seconds (from 1 to 300) that CA OPS/MVS waits for a remote SQL response. The default is the value of MSFSYSWAIT.

TNGNOTIFY

(Optional) Allows the TNGNOTIFY column used by the CA Event Manager to be set to ALWAYS or NEVER when it exists in the resource directory table. If the column does not exist in the table, this keyword is ignored and the current value is not changed. The default is NEVER.

WAIT

(Optional) Specifies the number of seconds that the STATESET program waits for the current state to become equal to the desired state. The STATESET program compares the current-state and desired-state values every three seconds; if the wait time expires, the STATESET program returns RC=2 along with a message. The default of WAIT is 0, but you can specify a value from 1 to 300.

STATESET Return Codes

The STATESET program issues these return codes:

Return Code	Message Number	Message Text
0	OPS3736E	STATESET COMMAND SYNTAX ERROR
2	OPS3733E	table.resource NOT AT DESIRED STATE
4	OPS3731E	table.resource NOT FOUND
12	OPS3732E	table.resource NOT UPDATED DUE TO ERRORS

Return Code	Message Number	Message Text
16	OPS3736E	STATESET COMMAND SYNTAX ERROR

Appendix A: REXX/EDQ Output from IMS Type 2 Message Responses

This section contains the following topics:

[XML Encapsulation of Types 1 and 2 Responses](#) (see page 793)

[Type 1 Response](#) (see page 794)

[Type 2 Response](#) (see page 795)

[Standard REXX and EDQ Output](#) (see page 796)

XML Encapsulation of Types 1 and 2 Responses

All data returned from an IMS Type 2 command is encapsulated within XML framing, much of which is not pertinent to viewing the desired results. By default, IMS Type2 command support will present only what is considered significant, thus reducing the number of created variables.

To make the complete collection of XML frames into variables, specify XML(YES) on the ADDRESS OPER command.

Examples 1 and 2 below illustrate the difference. In both examples, the stem name prefix defaults to IMSTYPE2.

For more information on XML usage in IMS Type 2 output, see the IBM manual SC18-9967, *IMS V10 System Programming API Ref* section "Descriptions of XML Tags Returned as CSL OM Response."

Type 1 Response

In the case of output from a Type 1 command issued to the Type 2 API, the ADDRESS OPER command does the following:

- Creates a running PREFIX.*n* stem name for each text line.
PREFIX.0 contains the total line count.
- If XML(YES) is specified, creates a corresponding stem variable for every text line, including those which may be considered as unnecessary, such as system information like the start time.
- Frames each line front and back with the identifying XML tags.

Specifying XML(NO) creates a considerably smaller number of REXX variables, consisting only of what would correspond to the output of an IMS operator command. No system information is included, and the XML framing tags are stripped off the front and back of each data line.

Example 1: /DIS ACT with XML(YES)

This example requests IMS to display the active IMS systems and create a REXX variable for each text line.

```
ADDRESS OPER "IMSPLEX(*) MSID(svpa) COMMAND(/DIS ACT.) XML(YES)"
```

The following is a partial output sample. The actual data lines are identified by the XML tag <msg>.

```
MSTY P2.0=30
MSTY P2.1=<imsout>
MSTY P2.2=<ctl>
MSTY P2.3=<omame>OMTOM </omame>
MSTY P2.4=<xmivsn>20 </xmivsn>
MSTY P2.5=<statime>2008.219 20:10:28.645491</statime>
MSTY P2.6=<stotime>2008.219 20:10:28.646038</stotime>
MSTY P2.7=<staseq>C2CE072AB2E7320A</staseq>
MSTY P2.8=<stoseq>C2CE072AB3096B8B</stoseq>
MSTY P2.9=<rc>00000000</rc>
MSTY P2.10=<rsn>00000000</rsn>
MSTY P2.11=</ctl>
MSTY P2.12=<cmd>
MSTY P2.13=<master>IMST </master>
MSTY P2.14=<userid>NESA S01 </userid>
MSTY P2.15=<verb>DIS </verb>
MSTY P2.16=<kw d>ACT </kw d>
MSTY P2.17=<input>/DIS ACT</input>
MSTY P2.18=</cmd>
MSTY P2.19=<msgdata>
MSTY P2.20=<msg> REGID JOBNAME TYPE TRANSTEP PROGRAM </msg>
```

```

MSTYP2.21=<msg>      JMFRGN  JMP  NONE</msg>
MSTYP2.22=<msg>      MSGRGN  TP   NONE</msg>
MSTYP2.23=<msg>      JBFRGN  JBP  NONE</msg>
MSTYP2.24=<msg>      BATCHREG BMP  NONE</msg>
MSTYP2.25=<msg>      DBTRGN  DBT  NONE</msg>
MSTYP2.26=<msg>      MSTDBRC DBRC      </msg>
MSTYP2.27=<msg>      MSTDLI  DLS  </msg>
MSTYP2.28=</mbr>
MSTYP2.29=</msgdata>
MSTYP2.30=</imsout>

```

Example 2: /DIS ACT with XML(NO)

A similar /DIS ACT command with XML set to NO. The member name is returned as *prefix.1*.

```

MSTYP2.0=10
MSTYP2.1="SVPA  "
MSTYP2.2=  REGID JOBNAME  TYPE  TRANSSTEP  PROGRAM  STATUS  CLASS
MSTYP2.3=  4 SVD10M11 TP           WAITING    1
MSTYP2.4=  JMFRGN  JMP  NONE
MSTYP2.5=  JBFRGN  JBP  NONE
MSTYP2.6=  BATCHREG BMP  NONE
MSTYP2.7=  3 SVD10F11 FPM  NO MSG.  DFSVP4
MSTYP2.8=  1 SVD10F13 FPM  NO MSG.  DBFSAMP3
MSTYP2.9=  2 SVD10F12 FPM  NO MSG.  DFSVP5
MSTYP2.10= DBTRGN  DBT  NONE

```

Type 2 Response

If the results are from a Type 2 command issued to the Type 2 API, the following actions occur:

1. Generates a set of *prefix.n* variables, one per line, as in the case above. The XML keyword has the same function as described above.
2. Creates a REXX stem variable for each column within each row.

The stem construct gets more complex because each column must be uniquely identified. The XML keyword has no effect on the creation of this set. The structured nature of the XML construct of Type 2 responses allows reference to an entire line through a single REXX variable, or every column within each line can be individually referenced.

3. The VERBOSE(YES) option can be specified if tokenization by individual words is necessary. An example of this is shown later in example 4 in this appendix.

Standard REXX and EDQ Output

Every execution of an IMS Type 2 command will result in one overall return code and one overall reason code. If multiple IMS systems are participating, some may encounter errors while others may not. Error information specific to an IMS system is grouped together by that system's name. A minimum number of REXX variables or EDQ records are always created to allow further error analysis if necessary. The actual number will vary depending on specific error conditions and the number of participating IMS systems.

Every invocation of ADDRESS OPER containing the IMSPLEX keyword will always result in the creation of a number of REXX variables. OPSCMD execution will result in EDQ records where the execution environment allows.

prefix.CTL.IMSRC

Returns with the overall return code.

prefix.CTL.IMSRSN

Returns with the overall reason code. If this is not 0, it will be followed by the following two messages:

1. prefix.CTL.RSNMSG – Returns the number of the appropriate IMS Type 2 message.
2. prefix.CTL.RSNTXT – Returns the message text of the above message.

prefix.CMDSECERR.SAF.IMSRC

Returns null, or the overall security return code.

prefix.CMDSECERR.SAF.RACFRC

Returns null, or the specific security return code

prefix.CMDSECERR.SAF.RACFRSN

Returns null, or the security reason code

prefix.CMDERR.0

Returns the number of participating member IMS systems with error information. A 0 value indicates no member is reporting errors. If this variable returns with a positive integer, *n*, the following are created:

prefix.CMDERR.*n*.0

Returns with number of individual error lines from the member identified as *n*. If multiple members are reporting errors, each will create a unique set per member, with the index *n* incrementing to *n*+1, *n*+2, and so on.

This sample shows five error lines, but the design allows for a variable amount.

1. prefix.CMDERR.*n*.TYP – Returns an error type indicator
2. prefix.CMDERR.*n*.NAME – Returns the member name corresponding to *n*.

3. prefix.CMDERR.n.IMSRC – Returns the return code specific to that member's results.
4. prefix.CMDERR.n.IMSRSN – Returns the reason code specific to that member's results.
5. prefix.CMDERR.n.RSNTXN – Returns the text describing the reason code above.

Example 3: REXX and EDQ Output

The following example is from an actual run. The sequence is arranged to match the description above. The indentation is intended to show the cause and effect relationship where the creation of one variable can result in more variables. REXX and EDQ output are very similar. In the case of REXX, the character string to the left of the first blank is the REXX variable name, and is followed by its value. The following are EDQ records:

```

MSTYP2.CTLIMSRC 0200000C
MSTYP2.CTLIMSRSN 00003000
    MSTYP2.CTLRSNMSG CSLN023I
    MSTYP2.CTLRSNTXT At least one request was successful
MSTYP2.CMDSECERR.SAF.IMSRC
MSTYP2.CMDSECERR.SAF.RACFRC
MSTYP2.CMDSECERR.SAF.RACFRSN
MSTYP2.CMDERR0 1
    MSTYP2.CMDERR1.0 5
        MSTYP2.CMDERR1.TYP UNKNW
        MSTYP2.CMDERR1.NAME MST
        MSTYP2.CMDERR1.IMSRC 02000010
        MSTYP2.CMDERR1.IMSRSN 00004008
        MSTYP2.CMDERR1.RSNTXT MJ is not registered

```

Example 4: VERBOSE(YES) Output

The following is an example of the granulated tokenization when the VERBOSE(YES) option is selected. Each word on every response line results in a unique REXX variable.

```

<?xml version="1.0"?>
MS2.XML = xrf1 version="1.0"
<DOCTYPE insout SYSTEM "insout.dtd">
MS2.DOC = DOCTYPE insout SYSTEM "insout.dtd"

<omname>OM2OM </omname>
MS2.CTL.OMNAME = OM2OM

<xm1vsr>20 </xm1vsr>
MS2.CTL.XMLVSN = 20

<stime>2008.268 18:56:31.852817</stime>
MS2.CTL.STATIME = 2008.268 18:56:31.852817

```

<stime>2008.268 18:56:31.853002</stime>
MS2.CTL.STOTIME = 2008.268 18:56:31.853002

<rc>0200000C</rc>
MS2.CTL.IMSRC = 0200000C

<rsn>00003000</rsn>
MS2.CTL.IMSRSN = 00003000

<rsnmsg>CSLN023I</rsnmsg>
MS2.CTL.RSNMSG = CSLN023I

<cmderr>
<nbr name="SVPA ">
MS2.CMDERR.1.NAME = SVPA

<typ>UNKWN </typ>
MS2.CMDERR.1.TYP = UNKWN

<rc>02000010</rc>
MS2.CMDERR.1.IMSRC = 02000010

<rsn>00004004</rsn>
MS2.CMDERR.1.IMSRSN = 00004004

</nbr>
MS2.CMDERR.1.0 = 3

<userid>NESAS01 </userid>
MS2.CMD.USERID = NESAS01

<verb>QRY </verb>
MS2.CMD.VERB = QRY

<kw d>MSFLEX </kw d>
MS2.CMD.KWD = MSFLEX

<input>QUERY MSFLEX </input>
MS2.CMD.INPUT = QUERY MSFLEX

```

<hdr slbl="MSPLX" lbl="MSplex" scope="LCL" sort="A" key="1" scroll="NO" len="8" dtype="CHAR" align="left" />
MS2.HDR.1.SLBL = MSPLX
MS2.HDR.1.LLBL = MSplex
MS2.HDR.1.SCOPE = LCL
MS2.HDR.1.SORT = A
MS2.HDR.1.KEY = 1
MS2.HDR.1.SCROLL = NO
MS2.HDR.1.LEN = 8
MS2.HDR.1.DTYPE = CHAR
MS2.HDR.1.ALIGN = left

<hdr slbl="MBR" lbl="MbrName" scope="LCL" sort="N" key="0" scroll="YES" len="8" dtype="CHAR" align="left" />
MS2.HDR.2.SLBL = MBR
MS2.HDR.2.LLBL = MbrName
MS2.HDR.2.SCOPE = LCL
MS2.HDR.2.SORT = N
MS2.HDR.2.KEY = 0
MS2.HDR.2.SCROLL = YES
MS2.HDR.2.LEN = 8
MS2.HDR.2.DTYPE = CHAR
MS2.HDR.2.ALIGN = left

<hdr slbl="CC" lbl="CC" scope="LCL" sort="N" key="0" scroll="YES" len="4" dtype="NT" align="right" />
MS2.HDR.3.SLBL = CC
MS2.HDR.3.LLBL = CC
MS2.HDR.3.SCOPE = LCL
MS2.HDR.3.SORT = N
MS2.HDR.3.KEY = 0
MS2.HDR.3.SCROLL = YES
MS2.HDR.3.LEN = 4
MS2.HDR.3.DTYPE = NT
MS2.HDR.3.ALIGN = right

<hdr slbl="MSMBR" lbl="Member" scope="LCL" sort="N" key="0" scroll="NO" len="8" dtype="CHAR" align="left" />
MS2.HDR.4.SLBL = MSMBR
MS2.HDR.4.LLBL = Member
MS2.HDR.4.SCOPE = LCL
MS2.HDR.4.SORT = N
MS2.HDR.4.KEY = 0
MS2.HDR.4.SCROLL = NO
MS2.HDR.4.LEN = 8
MS2.HDR.4.DTYPE = CHAR
MS2.HDR.4.ALIGN = left

<cmdsphdr>
MS2.HDR.0 = 4

```

```
<sp>MSPLX(CSLPLEX1) MBR(OM2OM) MSMBR(SVD9CQS) CQ( 0)</sp>  
MS2RSP.1.1 = MSPLX(CSLPLEX1)  
MS2RSP.1.2 = MBR(OM2OM)  
MS2RSP.1.3 = MSMBR(SVD9CQS)  
MS2RSP.1.4 = CQ( 0)  
MS2RSP.1.0 = 4
```

```
<sp>MSPLX(CSLPLEX1) MBR(OM2OM) MSMBR(SCI1SC) CQ( 0)</sp>  
MS2RSP.2.1 = MSPLX(CSLPLEX1)  
MS2RSP.2.2 = MBR(OM2OM)  
MS2RSP.2.3 = MSMBR(SCI1SC)  
MS2RSP.2.4 = CQ( 0)  
MS2RSP.2.0 = 4
```

```
<sp>MSPLX(CSLPLEX1) MBR(OM2OM) MSMBR(SVPA) CQ( 0)</sp>  
MS2RSP.3.1 = MSPLX(CSLPLEX1)  
MS2RSP.3.2 = MBR(OM2OM)  
MS2RSP.3.3 = MSMBR(SVPA)  
MS2RSP.3.4 = CQ( 0)  
MS2RSP.3.0 = 4
```

```
<sp>MSPLX(CSLPLEX1) MBR(OM2OM) MSMBR(SCI2SC) CQ( 0)</sp>  
MS2RSP.4.1 = MSPLX(CSLPLEX1)  
MS2RSP.4.2 = MBR(OM2OM)  
MS2RSP.4.3 = MSMBR(SCI2SC)  
MS2RSP.4.4 = CQ( 0)  
MS2RSP.4.0 = 4
```

```
<sp>MSPLX(CSLPLEX1) MBR(OM2OM) MSMBR(OM2OM) CQ( 0)</sp>  
MS2RSP.5.1 = MSPLX(CSLPLEX1)  
MS2RSP.5.2 = MBR(OM2OM)  
MS2RSP.5.3 = MSMBR(OM2OM)  
MS2RSP.5.4 = CQ( 0)  
MS2RSP.5.0 = 4
```

```
<sp>MSPLX(CSLPLEX1) MBR(OM2OM) MSMBR(RM2RM) CQ( 0)</sp>  
MS2RSP.6.1 = MSPLX(CSLPLEX1)  
MS2RSP.6.2 = MBR(OM2OM)  
MS2RSP.6.3 = MSMBR(RM2RM)  
MS2RSP.6.4 = CQ( 0)  
MS2RSP.6.0 = 4
```

```
<sp>MSPLX(CSLPLEX1) MBR(OM2OM) MSMBR(SVD10HWS) CQ( 0)</sp>  
MS2RSP.7.1 = MSPLX(CSLPLEX1)  
MS2RSP.7.2 = MBR(OM2OM)  
MS2RSP.7.3 = MSMBR(SVD10HWS)  
MS2RSP.7.4 = CQ( 0)  
MS2RSP.7.0 = 4
```

```
<rsp>MSPLX(CSLPLEX1) MBR(OM2OM) MSMBR(SVP9) CC( 0)</rsp>  
MS2.RSP.8.1 = MSPLX(CSLPLEX1)  
MS2.RSP.8.2 = MBR(OM2OM)  
MS2.RSP.8.3 = MSMBR(SVP9)  
MS2.RSP.8.4 = CC( 0)  
MS2.RSP.8.0 = 4
```

```
<rsp>MSPLX(CSLPLEX1) MBR(OM2OM) MSMBR() CC( 0)</rsp>  
MS2.RSP.9.1 = MSPLX(CSLPLEX1)  
MS2.RSP.9.2 = MBR(OM2OM)  
MS2.RSP.9.3 = MSMBR()  
MS2.RSP.9.4 = CC( 0)  
MS2.RSP.9.0 = 4
```

```
<rsp>MSPLX(CSLPLEX1) MBR(OM2OM) MSMBR(SVD91HWS) CC( 0)</rsp>  
MS2.RSP.10.1 = MSPLX(CSLPLEX1)  
MS2.RSP.10.2 = MBR(OM2OM)  
MS2.RSP.10.3 = MSMBR(SVD91HWS)  
MS2.RSP.10.4 = CC( 0)  
MS2.RSP.10.0 = 4
```

```
<rsp>MSPLX(CSLPLEX1) MBR(OM2OM) MSMBR(SVDACQS) CC( 0)</rsp>  
MS2.RSP.11.1 = MSPLX(CSLPLEX1)  
MS2.RSP.11.2 = MBR(OM2OM)  
MS2.RSP.11.3 = MSMBR(SVDACQS)  
MS2.RSP.11.4 = CC( 0)  
MS2.RSP.11.0 = 4
```

```
<cmdrspdata>  
MS2.RSP.0 = 11
```