CA OPS/MVS® Event Management and Automation

AOF Rules User Guide

Release 12.1

CA Technologies Product References

This document references the following CA Technologies products:

- CA 7™ Workload Automation (CA 7)
- CA ACF2™ for z/OS (CA ACF2)
- CA Automation Point
- CA Common Services (CCS)
- CA Endevor® Software Change Manager (CA Endevor SCM)
- CA Event Manager
- CA MIC Message Sharing (CA MIC)
- CA MIM™ Resource Sharing (CA MIM)
- CA Netman™ (CA Netman)
- CA NetMaster® Network Automation (CA NetMaster)
- CA NSM
- CA OPS/MVS® Event Management and Automation (CA OPS/MVS)
- 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 -AOF Variables Available in an EOM Rule (see page 169) section.
- Updated the Execution Considerations for GLV Rules (see page 185) section.
- Updated the AOF Variables Available in a GLV Rule (see page 188) section.
- Updated the OPAU Variables for All Security Events (see page 271) section.
- Updated the ADDRESS HWS (see page 364) section.
- Updated the OPS/REXX Host Environments in the)PROC Section of an API Rule (see page 72) section.
- Updated the OPS/REXX Host Environments in the)PROC Section of an ARM Rule (see page 117) section.
- Updated the OPS/REXX Host Environments in the)PROC Section of a CMD Rule (see page 136) section.
- Updated the OPS/REXX Host Environments in the)PROC Section of a DOM Rule (see page 147) section.
- Updated the OPS/REXX Host Environments in the)PROC Section of an EOJ Rule (see page 155) section.
- Updated the OPS/REXX Host Environments in the)PROC Section of an EOM Rule (see page 166) section.
- Updated the OPS/REXX Host Environments in the)PROC Section of an EOS Rule (see page 174) section.
- Updated the OPS/REXX Host Environments in the)PROC Section of a GLV Rule (see page 185) section.
- Updated the OPS/REXX Host Environments in the)PROC Section of an MSG Rule (see page 201) section.
- Updated the OPS/REXX Host Environments in the)PROC Section of an OMG Rule (see page 239) section.
- Updated the OPS/REXX Host Environments in the)PROC Section of a REQ Rule (see page 247) section.
- Updated the OPS/REXX Host Environments in the)PROC Section of an SCR Rule (see page 255) section.

- Updated the OPS/REXX Host Environments in the)PROC Section of an SEC Rule (see page 266) section.
- Updated the OPAU Variables for OPSGLOBAL Security Events (see page 288) section.
- Updated the OPS/REXX Host Environments in the)PROC Section of a TOD Rule (see page 335) section.
- Updated the OPS/REXX Host Environments in the)PROC Section of a TLM Rule (see page 322) section.
- Updated the OPS/REXX Host Environments in the)PROC Section of a USS Rule (see page 346) section.
- Updated the OPAU Variables for SOF Security Events (see page 292) section.
- Updated the Installation Requirements for GLV Rules (see page 183) section.
- Updated the Example: GLV Rule (see page 192) section.

Contents

Chapter 1: Using the AOF

This section contains the following topics:

What Is the AOF

Through the CA OPS/MVS Automated Operations Facility (AOF), you can develop applications to automate responses to system events.

The ability to react to various system events is mandatory when attempting to build effective automated applications. The AOF is a base component of CA OPS/MVS that monitors system events and automatically responds to them. You determine the system events that the AOF recognizes, and how it responds to those events, by defining special OPS/REXX programs called AOF rules.

AOF rules are classified as special OPS/REXX programs because AOF rules have a unique structure, reside in PDS data sets called rule sets, and are triggered by a system event.

The AOF can take action in response to the following types of system events:

Application Program Interface (API)

An API event occurs when an application program calls the API interface. Typically, the application that calls the API is a system service provider program, such as a tape library manager, or a network control program. When these programs detect an event that needs attention, they can initiate automation rules by calling the API. Applications can also request information from CA OPS/MVS through the API.

Automatic Restart Management (ARM)

An ARM event occurs when the z/OS Automatic Restart Manager tries to restart an ARM-registered job or started task after an unexpected termination. The restart may occur on the same system or on another system in the sysplex if the termination was due to a complete system failure.

Command (CMD)

A command event occurs when any z/OS or subsystem command is issued on the system.

Delete-operator-message (DOM)

A DOM event occurs when any z/OS component issues a DOM macro to remove a highlighted message from an MCS console; for example, a tape mount message gets internally DOMed when the mount is satisfied.

End-of-job (EOJ)

EOJ events occur when a task such as a batch job terminates and the INITSMF and EOSRULES parameters of CA OPS/MVS are set to YES.

EOJ rules have these advantages over message rules:

- Usually, two separate message rules are needed to handle both successful and unsuccessful completion cases. EOJ rules can contain the logic for both cases in one rule.
- Message rules do not work if a software component fails silently (without issuing any messages).

End-of-memory (EOM)

EOM events occur when any address space such as a TSO user or started task terminates.

EOM rules have these advantages over message rules:

- Usually, two separate message rules are needed to handle both successful and unsuccessful completion cases. EOM rules can contain the logic for both cases in one rule.
- Message rules do not work if a software component fails silently (without issuing any messages).

End-of-step (EOS)

An EOS event occurs when a step terminates in a job or started task and the INITSMF and EOSRULES parameters of CA OPS/MVS are set to YES.

Global variable (GLV)

A global variable event occurs when the value of an OPS/REXX global variable changes.

Message (MSG)

A message event occurs when a system component sends a message to a console or to a system log. The AOF recognizes and responds to these types of messages:

- z/OS
- IMS
- CICS (Transient Data Queue messages)
- JES2 or JES3
- WTOs (write-to-operator), WTORs (write-to-operator-with-reply), and WTLs (write-to-log) generated by an application
- Log file directed I/O (GDI)
- CA 7 Browse Log

OMEGAMON exception from MVS, IMS, CICS, and DB2 performance monitors (OMG)

An OMG event occurs when an OMEGAMON exception is generated from any of these products:

- OMEGAMON/MVS
- OMEGAMON/IMS
- OMEGAMON/CICS
- OMEGAMON/DB2

End user request (REQ)

A request event is triggered on demand by any end user.

Screen (SCR)

A screen event occurs when the screen or state of an EPI virtual terminal changes. Screen event rules allow you to automate any VTAM application.

Security (SEC)

A security event occurs when you invoke any CA OPS/MVS facility (for example, using the OPSCMD command processor to issue a z/OS command), allowing security for CA OPS/MVS to be performed by coding OPS/REXX programs, rather than by complex assembler exits. Security rules can be used to interface with your security product to provide comprehensive and flexible control of CA OPS/MVS facilities.

Time limit-exceeding (TLM)

A TLM event occurs when a job or task exceeds the processor time limit imposed by the system, either by default or by the TIME JCL parameter on the JOB or execute statement. A TLM event also occurs if a non-exempt job exceeds the maximum continuous wait time specified in the SMF parameters for the system.

Time-of-day (TOD)

A time event occurs at a specified time or date or after a specified time interval.

UNIX System Services (USS)

A USS event occurs when the CA OPS/MVS message exit of CCS for z/OS is driven by *one* of the following:

- The arrival of a local USS syslogd message
- A CA NSM message from the local CCS for z/OS
- A message forwarded from any remote CA NSM platform in the network

More information

Coding Each AOF Rule Type (see page 69)

AOF Rules and Rule Sets

AOF rules are stored in AOF rule sets. Rule sets are partitioned data sets whose names are defined by the CA OPS/MVS initialization parameters RULEPREFIX and RULESUFFIX. You can define from 1 to 70 rule sets, each containing any number of rules.

For example, according to the following parameter settings:

```
RULEPREFIX = 'OPSMVS.AOF'
```

```
RULESUFFIX = 'RULES'
```

AOF rule sets have a name mask of OPSMVS.AOF.*.RULES, where * is the rule set name. The following are examples of rule set names:

```
OPSMVS.AOF.SUPPRESS.RULES
OPSMVS.AOF.CICS.RULES
OPSMVS.AOF.STATEMAN.RULES
OPSMVS.AOF.DB2.RULES
OPSMVS.AOF.SECURITY.RULES
```

Each AOF rule occupies a separate member in the AOF rule set.

Examples: Individual rule in the SUPPRESS rule set

In the following example, $HASP100 is an individual rule in the SUPPRESS rule set:

```
OPSMVS.AOF.SUPPRESS.RULES($HASP100)
```

In the following example, IEF403I is an individual rule in the SUPPRESS rule set:

```
OPSMVS.AOF.SUPPRESS.RULES(IEF403I)
```

Note: Since AOF rule sets are defined by the RULEPREFIX and RULESUFFIX, it is possible to add or remove rule sets at any time without having to restart the product.

Determine AOF Rule Set Names

Choose a naming convention for your CA OPS/MVS rule sets that best fits your installation requirements.

To determine your AOF rule set names

1. Review the following examples of rule set naming conventions:
   - Rule sets named for each AOF rule type, for example, MSG for message rules, CMD for command rules, or TOD for time-of-day rules.
   - Rule sets named for a particular automated application or request, for example, a SUPPRESS rule set containing suppression rules, a JES rule set containing JES-related rules, or an IPLTIME rule set containing IPL-related rules.
   - Rule sets named for individual groups or divisions in your operations, for example, a CICSGRP rule set for CICS personnel, OPERATNS rule set for the operations personnel, or IMSGRP rule set for IMS personnel.

Multiple rule sets allow various groups using CA OPS/MVS in your data center to work independently of each other. Because each rule set is a separate data set, your security product can restrict rule set access to specific groups.

Sharing Rule Data Sets

AOF rule data sets (rule sets) can be shared between multiple copies of CA OPS/MVS that are running on either the same or different z/OS images that share DASD.

This configuration lets you easily manage your automated AOF application in a multi-system environment and is strongly recommended by CA. Even in cases where systems do not share DASD; we strongly recommended that you develop and manage a single set of AOF rule data sets from a central location and distribute these data sets to the remote systems.

For a discussion of how to implement the OPS/REXX logic to allow or disallow execution of a shared AOF rule on selected systems, see the chapter "AOF Rule Tools (see page 29)."

Allocate Rule Sets

Follow standard ISPF/PDF data set naming requirements when creating rule sets.

To allocate rule sets

1. Use either of the following methods:
   - Option 3.2 of ISPF/PDF
   - The TSO ALLOCATE command

   The Data Set Utility menu displays.
2. Complete the fields and allocate the data set.
3. Use the following parameter values when allocating AOF rule sets:
   - DSORG=PO
   - RECFM=FB
   - LRECL=80

   Use these parameters at your discretion:
   - DSN (see Determining AOF Rule Set Names in this chapter.)
   - BLKSIZE
   - SPACE
   - UNIT
   - STORCLAS

The rule sets are allocated.

Additional AOF Rule Set Information

The *Installation Guide* contains the following information:

- Security and performance-related information
- Alternative naming conventions

For information on the RULEPREFIX, RULESUFFIX, and RULEALTFIX parameters, see the *Parameter Reference*.

Chapter 2: AOF Rule Structure

This section contains the following topics:

Structure of an AOF Rule

AOF rules are classified as special OPS/REXX programs and, like any programming language, require a structured format. This chapter discusses the general coding format you should use in all AOF rules.

An AOF rule can contain up to five of the following control sections. Not all of the control sections are required, but they must appear in the following order. A unique section header in the rule identifies the control section.

Event Definition

The section header identifier is)*eventtype eventspec*.

Initialization

The section header identifier is)INIT.

Processing

The section header identifier is)PROC.

Termination

The section header identifier is)TERM.

End

The section header identifier is)END.

Section header identifiers delimit each section of a rule and logically control when the code in the associating section executes. A rule *must* contain an event definition section and at least *one* other section.

Each section header identifier must:

- Begin with a) character in column 1 of the line
- Appear on a line by itself

The fact that a rule section must begin with a) character in column one imposes a restriction on OPS/REXX code used in rules. OPS/REXX continuation lines in rules may not start with a) character in column one.

For example, the following rule will fail to enable:

```
)MSG BADRULE
)PROC
if (GLOBAL.SUPPRESSIT=1,
) then
  return "SUPPRESS"
```

)eventtype eventspec—Event Definition Section (Required)

The *event definition* section of a rule identifies the system event that causes the rule to execute. CA OPS/MVS uses the information in the event definition section to determine when to run the processing section of the rule.

The event definition section is required and is always the first section of a rule.

Use this format for coding the event definition section:

```
)eventtype eventspec
```

Following is a description of the two parts (*eventtype* and *eventspec*) that make up the event definition section.

- The *eventtype* Value

  The following *eventtype* values represent events that the AOF can recognize:

  API

  Application Program Interface

  ARM

  Automatic Restart Management event

  CMD

  Operator command event

  DOM

  Delete-operator-message event

  EOJ

  End-of-job event

  EOM

  End-of-memory event

  EOS

  End-of-step event

GLV

Global variable event

MSG

Message event

OMG

OMEGAMON exception event

REQ

End user request event

SCR

Screen event

SEC

Security event

TLM

Time limit-exceeded event

TOD

Time-of-day event

USS

UNIX System Services event

For more information about the types of system events that the AOF recognizes, see the chapter “Using the AOF (see page 15).”

- The *eventspec* Value

  The *eventspec* value is a character string template matching some event identifier (such as message IDs for MSG events, system commands for CMD events, and time specifications for TOD events).

  Each AOF rule type has a different *eventspec* template.

Examples: Event Definition Section

- Example 1: Identifies $HASP100 messages

  ```
  )MSG $HASP100
  ```

- Example 2: Identifies VARY commands

  ```
  )CMD VARY
  ```

- Example 3: Identifies an hourly time event

  ```
  )TOD ,1 HOUR
  ```

)PROC—Processing Section

The processing section of a rule specifies the actions that the rule takes in response to the AOF detecting the system event that is defined in the event definition section. This section can contain actions of varying complexity.

The processing section is optional. If included, the processing section always follows these sections:

- Event definition
- Initialization (if this section exists)

The processing section has the following format:

```
)PROC
/* Insert processing section actions */
```

Example:)PROC Section

The processing section in the following example uses a combination of AOF tools (REXX, AOF variables, OPS/REXX host environments) to send an alert message to the sysplex master console if job PRDCICSA abends.

```
)MSG IEF450I
)INIT
/* This code will fire ONCE when the rule is enabled    */
if OPSINFO('SMFID') <> 'SYSA' then
  Return 'REJECT'
TABENDS = 0

)PROC
/* This code will fire each time AOF detects an IEF450I */
/*  message event on the system.                        */
TABENDS = TABENDS + 1
if MSG.JOBNAME <> 'PRDCICSA' then
  return
parse var MSG.TEXT . 'ABEND=' ABEND
CONSOLE = OPSINFO('MSTCONSNM')
ADDRESS WTO
"MSGID(OPSAUTO1) TEXT('PRDCICSA ABEND CODE=",
ABEND" at "TIME()" ') HILITE CNNAME("CONSOLE")"
```

)INIT—Initialization Section

The *initialization section* of a rule specifies the actions the rule takes when an attempt is made to enable it. This section can contain actions of varying complexity. Use the initialization section to do the following:

- Perform one-time initialization for the rule
- Define static variables
- Prevent the enabling of a rule using the OPS/REXX RETURN statement

Note: The execution of the)INIT section of any rule type is an enable rule event. The REXX stem variable, ENA., is reserved for future enable event environment variables. Any use of the ENA. stem in any)INIT section of a rule will result in a REXX improper variable use error and cause the rule to be not enabled.

The initialization section is optional and if included, this section *always* follows the event definition section.

The initialization section has the following format:

```
)INIT
/* Insert initialization section actions */
```

Example:)INIT Section

The initialization section in the following example uses the OPS/REXX OPSINFO function to obtain the SMFID of the current system and determines if this rule is about to be enabled on system SYSA. The rule will reject (not allow) enabling for all other systems. This example also initializes a static variable called TABENDS to zero, which is used in the subsequent processing section.

```
)MSG IEF450I
)INIT
/*This code will fire ONCE when the rule is enabled */
if OPSINFO('SMFID') <> 'SYSA' then
  return 'REJECT'
TABENDS = 0
```

)TERM—Termination Section

The termination section of a rule specifies the actions that the rule takes when an attempt is made to disable it. This section can contain actions of varying complexity.

Use the termination section to do the following:

- Re-initialize or reset global variables, RDF tables, or both.
- Record information about the activity of a rule. This information may have been accumulated in static or global variables.
- Prevent disabling of a rule using the AOF RETURN statement.

Note: The execution of the)TERM section of any rule type is a disable rule event. The REXX stem variable, DIS., is reserved for future disable event environment variables. Any use of the DIS. stem in any)TERM section of a rule will result in a REXX improper variable use error and cause the rule not to be disabled.

The termination section is optional. If included, the termination section always follows these sections:

- Event description
- Initialization section (if it exists)
- Processing section (if it exists)

Use this format when coding the termination section:

```
)TERM
/* Insert termination section actions */
```

Example:)TERM Section

The termination section in the following example allows the rule to be disabled only at CA OPS/MVS shutdown. In addition, it sends a message to the OPSLOG with the value of the static variable TABENDS, which was calculated during the)PROC section of the rule.

```
)MSG IEF450I
)INIT
/* This code will fire ONCE when the rule is enabled */
if OPSINFO('SMFID') <> 'SYSA' then
  return 'REJECT'
TABENDS = 0

)PROC
/* This code will fire each time AOF detects an IEF450I */
/*  message event on the system.                        */

TABENDS = TABENDS + 1
if MSG.JOBNAME <> 'PRDCICSA' then
  return
parse var MSG.TEXT . 'ABEND=' ABEND
CONSOLE = OPSINFO('MSTCONSNM')
ADDRESS WTO
"MSGID(OPSAUTO1) TEXT('PRDCICSA ABEND CODE",
"("ABEND" at "TIME()" ') HILITE CNNAME("CONSOLE")"

)TERM
/* This code will fire ONCE when the rule is disabled */
if OPSINFO('PRODUCTSTATUS') <> 'TERM' then
  return 'REJECT'
MSG = 'OPSAUTO1 total IEF450I ABENDS = 'TABENDS
LOGTOTALS = OPSSEND('*','B',MSG)
```

)END—End Section

The end section consists of the END statement, which marks the end of a rule.

The END statement is optional and does not affect rule execution. If included, the END statement is always the last line of a rule.

The END statement has the following format:

```
)END
```

Chapter 3: AOF Rule Tools

This section contains the following topics:

Tools Available in AOF Rules

The purpose of all AOF rules is to perform an automated procedure such as issuing system messages or system commands, querying system devices, or updating user databases.

The following tools are available in AOF rules to accomplish these automated tasks:

AOF variables

Obtains, saves, and shares event data.

REXX programming techniques

Makes decisions about AOF events.

OPS/REXX host environments and built-in functions

Programmatically performs various system actions and queries the status of system resources.

AOF RETURN statement

Suppresses messages.

These tools are described in the following sections.

AOF Variables

When creating AOF rules to trigger on a particular system event, you need to know specific information regarding the event, such as the job name that issued a message, the route codes associated with the message, or the console name from which a certain system command was issued. In addition, your particular automated application may require that data be saved over various executions of an event or shared between different events. You obtain, save, and share event data by using AOF variables.

AOF variables can be one of the following types:

- Environmental (event-related)
- Dynamic
- Static
- Address space-related
- OPS/REXX global
- Event-related

Environmental (Event-related) Variables

AOF environmental (event-related) variables provide detailed information about the system event that AOF is evaluating. Each AOF event rule type (MSG, CMD, EOM, and so on) provides a set of unique environmental variables. For example, in a MSG rule type, you have access to various MSG.*xxxx* variables such as the following:

MSG.TEXT

Contains the complete text of the current message.

MSG.JOBNAME

Contains the name of the job or address space that issued the message.

MSG.REPLYID

Contains the reply for WTOR messages.

In a CMD rule type, you have access to various CMD.*xxxx* variables such as the following:

CMD.TEXT

Contains the complete text of the current command.

CMD.JOBNAME

Contains the name of the job or address space that issued the command.

CMD.CONSNAME

Contains the name of the console from which the command was issued.

Review the following additional facts about environmental (event-related) variables:

- Environmental variables are automatically provided by the AOF engine and are available *only* in the)PROC (processing) section of a rule.
- Some environmental variables are modifiable and can affect original event attributes.

  For example, changing the contents of the MSG.TEXT variable in a)MSG rule changes the text of the WTO message, or modifying the CMD.TEXT variable in a)CMD rule changes the original command that was entered.

  Note: Some environmental variables are designated as read-only; changing the value of a read-only variable is not allowed and results in an error condition. Because the values of read-only environmental variables do not change, all rules that execute for a single event get the same environmental data.
- Changes to an environmental variable by multiple rules are cumulative. The first rule to execute receives original event information. Subsequent rules (executing in response to the same event) receive event information modified by each preceding rule.

  Note: Because rules cannot change the values of read-only variables, these variables always contain original information.
- All rules that usually execute in response to an event do so regardless of how each rule changes an environmental variable.

  For example, if a message rule changes the message ID contained in the MSG.ID variable, then all rules matching the original message ID still execute (and no rules matching the new message ID execute).
- You can view most environmental variables using the DISPLAY primary command in the OPSLOG Browse facility.

  Note: For more information about the OPSLOG Browse facility, see the *OPSVIEW User Guide*.

More information:

Coding Each AOF Rule Type (see page 69)

Dynamic Variables

Dynamic variables are user-defined variables that are created each time a rule section executes. The dynamic variable data is available only when a rule section is executing. You use dynamic variables generally as a reference in the logic of an AOF rule section.

For example, consider the following rule logic:

```
)MSG $HASP094
)PROC
/**********************************************************/
/* Rule Purpose :     Hilite I/O Error messages for Line 50 */
/* $HASP094 I/O LNExx SNA ,17,0000,087D0001,JOB NAME        */
/**********************************************************/
LINE = WORD(MSG.TEXT,3)              /* Extract line no.    */
if LINE < > 'LNE50' then             /* Is this LNE50?      */
  return
MSG.DESC = OPSBITS('IMEDACTN')       /* Hilite if it is     */
```

The variable LINE is an example of a dynamic variable and is only available as each $HASP094 message is processed by this MSG rule.

Review the following additional facts about dynamic variables:

- Simple dynamic variables are used in the)PROC or)TERM sections of a rule but *not* in the)INIT section. Compound dynamic variables are used in all rule sections.
- The name of a dynamic variable can be up to 256 characters in length.
- The value assigned to a dynamic variable can be up to 32,000 bytes in length.
- A dynamic variable can be a compound symbol, such as JOB.COUNT.

  Note: It cannot begin with a reserved stem used by the environmental variables for that particular event type (for example, MSG.*xxx*, CMD.*xxx*, EOM.*xxx*).
- The value of an uninitialized dynamic variable (a variable that has not yet been assigned a value) is the variable name itself.
- The following three special variables in standard REXX are always dynamic variables in AOF rules:
  - RC
  - RESULT
  - SIGL

Static Variables

Static variables maintain their value across multiple executions of a single rule, which means that data can be shared between executions of the same AOF rule.

For example, consider the following rule logic:

```
)MSG $HASP373
)INIT
COUNT=0
)PROC
/***********************************************************/
/* Rule Purpose :     Keep a running total of HASP373 jobs  */
/* $HASP373 jobname   STARTED                               */
/***********************************************************/
COUNT= COUNT + 1                        /* Add to counter  */
)TERM
LOGIT = OPSSEND('*','B','OPSAUTO4 TOTAL 373 = 'COUNT)
```

The variable COUNT is a static variable that retains its value each time this rule processes a $HASP373 message.

Review the following additional facts about static variables:

- Static variables are rule-specific and must be defined (initialized) in the)INIT section.
- The same static variable name used in the initialization sections of two different rules refers to two different static variables.
- A static variable cannot be a compound symbol (for example, JOB.COUNT).
- The name of a static variable can be up to 50 characters in length.
- The value assigned to a static variable can be up to 256 bytes in length. Any value assigned with a length greater than 256 is truncated.
- A static variable is deleted when the rule is disabled.
- Access to static variables is not serialized. If serialization is needed, then use OPS/REXX global variables and the OPSVALUE function of OPS/REXX.

Address Space-related Variables

Address space-related variables are compound symbols that begin with a reserved stem of GLVJOBID. They let you share data between different AOF rules for events that originate from the *same* address space. This lets you save data generated during one event created by a job, and then use that data in another event created by the same job.

For example, consider the following two AOF rules:

- Rule #1

```
)MSG $HASP375
)PROC
/*****************************************************************/
/* Rule Purpose : Set a local variable to keep track of the most */
/*                current # of lines exceeded during run time    */
/*                This variable will be checked when the job     */
/*                ending event occurs.                           */
/* $HASP375 jobname ESTIMATE EXCEEDED BY # LINES                 */
/*****************************************************************/
if WORD(MSG.TEXT,5) < > 'BY' then
  return
GLVJOBID.EXCEEDED = WORD(MSG.TEXT,6)  /* # of lines              */
```

- Rule #2

```
)EOJ *
)PROC
/*****************************************************************/
/* Rule Purpose : Check to see if the batch job that just ended  */
/*                exceeded any output lines during its run time  */
/*                by testing to see if the GLVJOBID variable     */
/*                that would have been set in the $HASP375       */
/*                rule exists. Log info if variable is present.  */
/*  EOJ fires automatically when job terminates.                 */
/*****************************************************************/
if OPSVALUE('GLVJOBID.EXCEEDED','E') = 'N' then
  return
JOB = EOJ.JOBNAME
NUMLINES = GLVJOBID.EXCEEDED
MSG = 'OPSAUT01 BATCH JOB ' JOB' LAST EXCEED ='NUMLINES
LOGIT = OPSSEND('*','B',MSG)
```

Suppose that JOB1 and JOB2 start on the system at the same time. JOB1 begins to exceed expected output lines and the JOB1 address space produces a $HASP375 message. This event executes Rule #1, which sets a unique GLVJOBID.EXCEEDED variable for JOB1 only. Both JOB1 and JOB2 end at the same time, thus executing Rule #2.

While Rule #2 is processing the end-of-job event caused by JOB1 ending, the local variable GLVJOBID exists, and the rule produces the informational message in the OPSLOG that includes the value of the GLVJOBID.EXCEEDED address space-related variable that was set in Rule #1. While Rule #2 is processing the end-of-job event caused by JOB2, the local variable GLVJOBID.EXCEEDED does not exist, so no further processing is done in the rule.

Review the following additional facts about address space-related variables:

- Address space-related variables are used only in the)PROC section of a rule and are unique to the address space that triggered the rule.
- They are automatically deleted by CA OPS/MVS when the address space or batch job associated with it terminates. Batch jobs are handled differently than other address spaces. The address space-related variables associated with a batch job are deleted when the batch job ends, even though the initiator address space in which it ran remains active and may subsequently execute other batch jobs.
- An address space-related variable name must begin with a stem of GLVJOBID.
- The name can be up to 78 characters in length including the stem of GLVJOBID.
- The value can be up to 32,000 bytes.
- Some messages appear to be issued from a particular job but are actually issued on behalf of another address space. For example, the $HASP100 message that indicates a job is on the internal reader is actually issued from the JES2 address space and not the address space of the job. Use the OPSLOG JOBNAME and ASID columns to verify that the message is being issued from unique address spaces prior to selecting a GLVJOBID variable.

OPS/REXX Global Variables

OPS/REXX global variables are compound symbols that begin with a reserved stem of GLOBAL., GLOBAL*x*, or GLVTEMP*x*. They let you share data between different AOF rules for events that occur from *any* address space. Programs running in CA OPS/MVS OSF TSO servers, batch, TSO, NetView, and UNIX System Services environments can all access OPS/REXX global variables used in AOF rules.

A global variable with a stem of GLOBAL. or GLOBAL*x* is permanently saved across IPLs or restarts of CA OPS/MVS. A global variable with a stem of GLVTEMP*x* is saved only while CA OPS/MVS is active.

Examples: OPS/REXX Global Variables Used in AOF Rules

- Rule #1: This rule will execute for every DFHSI1517 message that is issued when a CICS region initializes. An OPS/REXX global variable is created using the region name as part of the compound symbol name so that a unique variable exists for each CICS region, such as GLVTEMP1.UPTIME.CICSA (for region CICSA) and GLVTEMP1.UPTIME.CICSB (for region CICSB). The variable is set to the current system time.

```
)MSG DFHSI1517
)PROC
/****************************************************************/
/* Rule Purpose : Set a unique OPS/REXX global variable with    */
/*                the initialization times of all CICS regions. */
/* DFHSI1517 cicsregion Control is being given to CICS.         */
/****************************************************************/
JOB = MSG.JOBNAME      /* set JOB to issuer of this message     */
CTIME = TIME()         /* set CTIME to current time             */
/* Create a unique global variable using the JOB value as a stem*/
/* name to make it unique. Set it to the CTIME value            */
SET = OPSVALUE('GLVTEMP1.UPTIME.'JOB,'U',CTIME)                 */
```

- Rule #2: This is an AOF request rule (a different event type from Rule #1) that will execute on demand by any TSO user and be able to access and display the GLVTEMP1.UPTIME OPS/REXX variables set by Rule #1. It is using the OPSVALUE OPS/REXX function to retrieve this variable information.

```
)REQ CICSINIT
)PROC
/****************************************************************/
/* Rule Purpose : Display initialization times of active CICS   */
/*         regions when requested. Obtain this info             */
/*         via any GLVTEMP1.UPTIME global variable.             */
/*  Invoked when a TSO users issues OPSREQ CICINIT              */
/****************************************************************/
ACTREGIONS = OPSVALUE('GLVTEMP1.UPTIME','L')
if ACTREGIONS = 0 then
  say 'No initialized regions'
else
  do ACTREGIONS
    PULL REGION
    UPTIME = OPSVALUE('GLVTEMP1.UPTIME'.REGION,'O')
    say 'CICSINIT - 'REGION' INIT TIME = 'UPTIME
  end

return
```

Note: For a detailed description of the uses and characteristics of GLOBAL., GLOBAL*x*, and *GLVTEMPx* OPS/REXX global variables, see the *User Guide*.

Event-related Variables

Event-related variables are compound symbols that begin with a reserved stem of GLVEVENT. They let you share data between different AOF rules that are processing the same event. In a case where you allow different groups (for example, operations, CICS, IMS) to have their own rule sets, there may be a need to coordinate a process between two or more rules that execute on the same event. Event-related variables have a life span of the event.

Example: Sharing Data Between AOF Rules

- Rule # 1: This first rule resides in the CICSGRP rule set

```
)MSG $HASP100
)PROC
/************************************************************/
/* Rule Purpose :  Set a event-related variable to flag that */
/*          the CICSGRP wants to currently request that      */
/*          $HASP100 messages from CICSx jobs get            */
/*          suppressed and deleted from SYSLOG               */
/* $HASP100 CICSxxx ON INTRDR                                */
/************************************************************/
JOB = WORD(MSG.TEXT,2)              /* get 2nd word of message */
if SUBSTR(JOB,1,4) <> 'CICS' then  /* not CICSx              */
  return
GLVEVENT.DISP = 'DELETE'
```

- Rule #2: This second rule resides in the IMSGRP rule set

```
)MSG $HASP100
)PROC
/************************************************************/
/* Rule Purpose :  Set a event-related variable to flag that */
/*          the IMSGRP wants to currently request that       */
/*          $HASP100 messages from IMSx jobs get             */
/*          suppressed.                                      */
/* $HASP100 IMSxxx ON INTRDR                                 */
/************************************************************/
JOB = WORD(MSG.TEXT,2)              /* get 2nd word of message */
if SUBSTR(JOB,1,3) <> 'IMS' then   /* not IMSx               */
  return

GLVEVENT.DISP = 'SUPPRESS'
```

- Rule #3: This third rule resides in the OPERATNS rule set

  ```
  )MSG $HASP*
  )PROC
  /*************************************************************/
  /* Rule Purpose :  Operations makes the final call on message */
  /*          disposition based on the value of a event-related */
  /*          variable that different groups can override       */
  /*          at their request. Assume disposition is           */
  /*          NORMAL if a group hasn't set the variable         */
  /*************************************************************/
  if OPSVALUE('GLVEVENT.DISP','E') = 'N' then
    DISP = 'NORMAL'
  else
    DISP = GLVEVENT.DISP

  return DISP
  ```

Rule #1 and Rule #2 execute on a specific $HASP100 event and will execute before Rule #3 because it is a more generic message specification of $HASP*. They will set the event-related variable GLVEVENT.DISP accordingly. Rule #3 will then interrogate this variable to determine what action to take.

Review the following additional facts about event-related variables:

- Event-related variables are used only in the)PROC section of a rule and are unique to rules that execute on the same event.
- They are automatically deleted by CA OPS/MVS when the last rule executing on the event completes.
- An event-related variable must begin with a stem of GLVEVENT.
- The name can be up to 78 characters in length including the stem of GLVEVENT.
- The value can be up to 32,000 bytes.

More information:

Building and Controlling AOF Rules (see page 49)

REXX Programming Techniques

Logic commonly implemented in all AOF rules is to manipulate and process data. Standard REXX programming tools can be used in AOF rules to perform this type of logic, such as making decisions about an AOF event or further breaking down AOF event data.

REXX instructions, such as the IF..THEN..ELSE statement or the SELECT..WHEN..OTHERWISE statement, let you make decisions about data in an AOF rule, while the PARSE instruction-or perhaps a REXX function such as WORD(), SUBSTR(), or POS()-lets you further interrogate AOF event data.

Examples: REXX Programming Techniques

The following examples demonstrate standard REXX programming used in AOF rules:

- **Example 1:** Insert REXX comments to provide information:

  ```
  /* Date Created: 11/11/12                              */
  /* Purpose : Configure initiators to handle batch window */
  ```

- **Example 2:** Make decisions through the IF..THEN..ELSE REXX instruction:

  ```
  JOB=MSG.JOBNAME             /* set JOB to issuer of msg */
  if JOB = 'MYJOBA' then
    RETURN 'SUPPRESS'
  else
    return 'DELETE'
  ```

- **Example 3:** Break down a data string with the PARSE REXX instruction:

  ```
  /* Obtain the abend code from this message and put in ABEND var */
  parse var MSG.TEXT . 'ABEND=' ABEND .
  ```

- **Example 4:** Break down a data string with the WORD and SUBSTR REXX functions:

  ```
  /* Obtain the first 3 characters of the job name in this message */
  JOBN = WORD(MSG.TEXT,2)        /* job is 2nd blank delimited word */
                                 /* in this msg */
  MASK = SUBSTR(JOBN,1,3)        /* get the job name mask */
  ```

For more detailed information about standard REXX, see any REXX reference guide. An excellent reference is the second edition of *THE REXX LANGUAGE: A Practical Approach to Programming* by M.F. Cowlishaw, available through Prentice Hall publishers. In addition, the CA OPS/MVS EasyRule facility is a good tool for learning REXX.

Note: OPS/REXX is the underlying programming language used in AOF rules. Standard REXX is a subset of OPS/REXX, so minor operational differences may exist for particular standard REXX techniques that you attempt to use in AOF rules. For information regarding these differences, see the *User Guide* and *Command and Function Reference*.

OPS/REXX Host Environments and Built-in Functions

The OPS/REXX language is considered an enhanced REXX language implementation, designed specifically for automation use, because it provides the tools needed to programmatically perform various system actions and query the status of a wide range of system resources. You can use these tools, known as OPS/REXX host environments and OPS/REXX built-in functions, in AOF rules.

OPS/REXX Host Environments

OPS/REXX host environments let you perform common automation and system operations, such as issuing messages (WTOs), z/OS commands, and UNIX System Services commands. You can also use OPS/REXX host environments in AOF rules to control various CA OPS/MVS facilities.

Use the following OPS/REXX host environments in AOF rules:

ADDRESS AOF

Programmatically control rules and create dynamic AOF rules.

ADDRESS EPI

Programmatically interact with any VTAM application.

ADDRESS LXCON

Issue commands to VM and Linux systems that are connected to the Linux Connector component.

ADDRESS MIM

The CA MIM product manages and collects this display information.

ADDRESS NETMAN

Open, update, or close records in CA Netman.

ADDRESS OPSCTL

Control COF, ECF, OSF, MSF, and OPSLOG components

ADDRESS OPER

Issue z/OS, JES, VM, subsystem commands

ADDRESS OSF

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSO servers

ADDRESS OSFTSL

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSL servers

ADDRESS OSFTSP

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSP servers

ADDRESS SOF

Provide a user interface for managing devices that are attached through ESCON directors and FICON switches.

ADDRESS SQL

Create and maintain SQL tables in the CA OPS/MVS RDF component.

ADDRESS SYSVIEWE

Retrieve output from CA SYSVIEW commands.

ADDRESS USS

Issue UNIX System Services commands

ADDRESS WTO

Issue system messages in the form of WTOs, or WTORs

Note: For complete descriptions and coding guidelines for each OPS/REXX host environment, see the *CA OPS/MVS Command and Function Reference*.

More information:

Coding Each AOF Rule Type (see page 69)

OPS/REXX Built-in Functions

These built-in functions are part of OPS/REXX that gives these AOF rules access to a wide variety of system data for programmatic actions. In some cases, certain OPS/REXX functions can perform specific system tasks. OPS/REXX functions can retrieve a system information, such as job status, device status, or JES2-related resource data. OPS/REXX functions can also perform such tasks as low lighting messages or invoking z/OS Automatic Restart Management Services.

You can use the following OPS/REXX built-in functions in these AOF rules:

OPSARM

Performs z/OS Automatic Restart Management Services.

OPSARMST

Determines whether a single job or element name is registered with ARM.

OPSCA7

Issues commands to the CA 7 scheduling product.

OPSCAWTO

Creates an SNMP trap.

OPSCLEDQ

Empties the contents of the External Data Queue (stack)

OPSCOLOR

Changes the color of OPSLOG (or console) messages.

OPSCPF

Obtains z/OS Command Prefix Facility information.

OPSDEV

Obtains device information.

OPSDOM

Deletes highlighted operator messages.

OPSECURE

Returns security package related information.

OPSHFI

Reads or writes variables to a shared VSAM file.

OPSINFO

Returns various system and CA OPS/MVS information.

OPSIPL

Obtains various IPL parameter library information.

OPSJES2

Obtains JES2 related resource data.

OPSLOG

Retrieves OPSLOG data.

OPSMTRAP

Generates a warm- or cold-start SNMP trap.

OPSPRM

Controls CA OPS/MVS parameters.

OPSSEND

Sends messages to OPSLOG or other copies of CA OPS/MVS.

OPSSMF

Creates SMF records.

OPSSRM

Collects System Resource Manager (SRM) data.

OPSTATUS

Obtains active ASID, WTOR, IMS, or WLM information.

OPSUBMIT

Builds and submit batch jobs.

OPSSYSYM

Obtains z/OS system symbol information.

OPSVALUE

Manipulates variables.

OPSVASRV

Manipulates SYSPLEX variables.

For complete descriptions and coding guidelines for each OPS/REXX built-in function, as well as a list of valid rule types in which each can execute, see the *CA OPS/MVS Command and Function Reference*.

Example: AOF Rule Using Built-in Functions

The following AOF rule uses OPS/REXX host environments and built-in functions:

```
)TOD 02:00
)INIT
/* This rule should only be active on SYSA. Use the OPSINFO       */
/* built-in function to get the SMFID of this system to see       */
/* if the rule should be enabled or not.                          */
if OPSINFO('SMFID') <> 'SYSA' then
  return 'REJECT'

)PROC
/****************************************************************/
/* Rule purpose : Set up Z initiators for batch window            */
/* TOD rule spec fires every 2:00 AM                              */
/****************************************************************/

/* Issue info message using ADDRESS WTO OPS/REXX                  */
/* Host Environment to let everybody know of change.              */
msgtxt= 'Initiators configured to handle 2:00 Batch Flow'
ADDRESS WTO
"MSGID(OPSAUTO1) TEXT(' "msgtxt" ') ROUTE(2)"

/* Let's start all drained 'Z' initiators. We'll obtain this info */
/* by using one of the options on the OPSJES2 built-in function.  */
/* Then we'll use the ADDRESS OPER OPS/REXX host                  */
/* environment to issue a command to JES.                         */
ZINITS=OPSJES2('I','INIT','Z','D')
do ZINITS                 /* Loop for all drained Z' inits        */
  pull RECORD             /* Obtain data from the EDQ             */
  INITID=WORD(RECORD,1)   /* First word is init id                */
  address OPER            /* Switch to ADDRESS OPER               */
  "C($SI"INITID") NOO"    /* Issue JES2 $Six command              */
end                       /* End of DO                            */
```

AOF RETURN Statement

You can use the AOF RETURN statement to suppress messages from going to a console or the SYSLOG and to reject system commands. You can also use it to allow or disallow a rule from being enabled or disabled.

Valid values for the RETURN statement vary according to the rule section that is processing the statement.

)INIT Section—Enable a Rule

The RETURN statement in the)INIT section of an AOF rule allows or disallows the enabling of a rule. Creating logic to prevent a rule from being enabled may be necessary in an environment in which you have multiple CA OPS/MVS images sharing rule sets, and you only want particular rules to be enabled on certain systems.

The)INIT section of an AOF rule has the following format:

```
)INIT
/* This rule should only be active on SYSA. Use the OPSINFO  */
/* built-in function to get the SMFID of this system to see  */
/* if the rules should be enabled or not.If this is not SYSA */
/* then use the AOF RETURN statement with a value of REJECT  */
/* to cause the rule to not be ENABLEd.                      */
if OPSINFO('SMFID') <> 'SYSA' then
  return 'REJECT'
```

Valid values for a RETURN statement in the initialization section of a rule are:

ACCEPT

(Default) Allows the rule to be enabled.

REJECT

Prevents the rule from being enabled.

If you do not specify a return value in the initialization section of the rule, the default return value is ACCEPT.

If a runtime error occurs in the INIT section, the return value is REJECT and the rule is not enabled.

)PROC Section—Valid Return Statement Values

The RETURN statement in the)PROC section of an AOF rule works differently according to the type of event that the rule is acting upon. Valid RETURN statement values are different in a)MSG rule than a)CMD rule. An unrecognized return value (for example, a misspelled value) defaults to a value of NOACTION.

WARNING! Make sure you use the valid RETURN values associated with each rule type. Using RETURN values for the wrong AOF rule type does not result in a syntax error and does not default to NOACTION. The results may not be what you expect them to be.

The)PROC section on an AOF rule has the following format:

```
)MSG $HASP100
)PROC
/* Suppress $HASP100 message using the SUPPRESS RETURN value */
/* available in )MSG rules.                                  */
return 'SUPPRESS'

)CMD MOVECICS
)PROC
/* This pseudo command rule allows operators to enter the    */
/* command MOVECICS from the console to initiate the OPS/REXX*/
/* program to move all CICS regions. The RETURN value of     */
/* 'ACCEPT' in a )CMD rule causes z/OS to not process this   */
/* pseudo command.                                           */
address OSF "OI P(MOVECICS)"
return 'ACCEPT'
```

For a list of valid RETURN statement values for each AOF rule type and an outline of the coding guidelines, see the chapter "Coding Each AOF Rule Type (see page 69)."

)TERM Section—Disable a Rule

The RETURN statement in the)TERM section of an AOF rule allows or disallows the disabling of a rule.

The)TERM section of an AOF rule has the following format:

```
)TERM
/* This rule must never be disabled, unless CA OPS/MVS       */
/* is shutting down. Use the OPSINFO function to see if   */
/* the product is terminating. If it is not, then don't   */
/* allow the rule to be disabled by using the REJECT      */
/* value of the RETURN statement.                         */
if OPSINFO('PRODUCTSTATUS') <> 'TERM' then
  return 'REJECT'
```

Valid values for a RETURN statement in the termination section of a rule are:

ACCEPT

(Default) Allows the rule to be disabled.

REJECT

Prevents the rule from being disabled.

If you do not specify a return value, then the default return value is ACCEPT.

If a runtime error occurs, then the return value is ACCEPT (assuming that the error occurs while the termination section is executing).

Note: A return value of REJECT stops the disabling of a single rule only. If you disable a rule set, the enabled rules in the rule set are always disabled regardless of the RETURN values in the individual rules.

More information:

Building and Controlling AOF Rules (see page 49)

Chapter 4: Building and Controlling AOF Rules

This section contains the following topics:

Ways to Create AOF Rules

CA OPS/MVS provides various ways to create AOF rules. Such flexibility allows automation to be created at your installation by either a novice or an experienced user. Most of the tools for creating rules can be accessed through the OPSVIEW ISPF interface of CA OPS/MVS.

EasyRule

EasyRule is an ISPF-based application that lets you create AOF rules using a series of fill-in-the-blank panels.

To access EasyRule, use any of the various options of the CA OPS/MVS OPSVIEW facility.

The AOF rule REXX code is generated in the background in response to the options that you specify on the selection panels. Little, if any, AOF rule knowledge is required to use EasyRule, so it is an excellent starting place for new CA OPS/MVS users.

EasyRule lets you build suppression rules and rules that perform many sorts of tasks, including:

- Issuing WTOs and z/OS and UNIX System Services commands
- Replying to WTORs
- Updating global variables and RDF tables
- Invoking OPS/REXX programs in CA OPS/MVS TSO servers

EasyRule also provides a series of panels to help you incorporate decision-making logic in rules, such as manipulating data in a message or querying the status of a job or device. Because some automated applications require greater complexity and may need to use a CA OPS/MVS OPS/REXX function or host environment not available in EasyRule, you may be unable to create some automated applications with EasyRule.

At that point you have the following two options:

- Add your own code at selected points in the code generated by EasyRule. With this option you can continue to use EasyRule in the future to modify the EasyRule generated portions of the code.
- Directly edit the generated code or copy the code to a rule that you create outside of EasyRule.

For details on using EasyRule, see the *User Guide*.

ISPF/PDF Editor in the AOF Test Facility

The AOF Test facility lets you create and test rules outside of the production AOF environment, thus providing you with a safe method of developing rules. A standard ISPF/PDF editor is the primary tool for creating rules in the AOF Test facility, but you can also use the EasyRule application.

Note: For an overview of the AOF Test facility, see the chapter “How to Begin Using the Product” in the *User Guide*. For more information about the AOF Test facility, see the *OPSVIEW User Guide*.

ISPF/PDF Editor in the AOF Production Facility

This facility is the primary way of controlling your production AOF rules. As you become more proficient in understanding the syntax of AOF rules and the various automation tools that they use, the ISPF/PDF editor of the AOF production environment will most likely become your primary method of creating rules.

To access the AOF production environment, choose option 4.5 of the CA OPS/MVS primary OPSVIEW panel.

For details on accessing the AOF production environment, see the *OPSVIEW User* Guide.

Automation Analyzer

The Automation Analyzer facility examines and displays a statistical analysis of the message activity of your system, and lets you directly create suppression rules and invoke the EasyRule facility.

To access this facility, choose option 7.2 of the CA OPS/MVS primary OPSVIEW panel.

For an overview of the Automation Analyzer, see the chapter “How to Begin Using the Product” in the *User Guide*. Additional details can also be found in the *OPSVIEW User Guide*.

MPF Conversion Facility

The MPF Conversion Facility reads the installation Message Processing Facility (MPF) member that is currently being used to suppress messages and automatically generates CA OPS/MVS suppression rules.

To access this facility, choose option 7.3 of the CA OPS/MVS OPSVIEW primary panel.

For an overview of the MPF Conversion Facility, see the chapter “How to Begin Using the Product” in the *User Guide*. More details can also be found in the *OPSVIEW User Guide*.

Sample AOF Rules

The CA OPS/MVS installation tape includes data sets that contain various pre-coded AOF rules. You can use these samples and the examples in this guide as models to create your own AOF rules.

Control AOF Rule Status

You control AOF rule status primarily in the AOF production environment.

To access the AOF production environment, choose option 4.5 of the CA OPS/MVS primary OPSVIEW panel.

Definition of AOF Rule Status

This section explains the AOF rule control terminology.

ENABLE

You must enable a rule so CA OPS/MVS can react to the event that you specified. A rule responds to a system event only if it is enabled.

DISABLE

A disabled rule is an inactive rule. CA OPS/MVS does not respond to the specified system event.

AUTOENABLE

An auto-enabled rule is automatically enabled when CA OPS/MVS initializes through either a system IPL or a restarting of the product. A reserved area in the ISPF directory entry is internally set when a rule is auto-enabled. ISPF statistics must be turned on to auto-enable a rule.

RESET AUTOENABLE

The reset auto-enable operation turns off the auto-enable flag, which means CA OPS/MVS does not enable the rule when it initializes through either a system IPL or a restart.

COMPILE

A compiled rule is a rule whose compiled version is saved in a predefined data set. Compiled rules expedite the enabling of auto-enabled rules during CA OPS/MVS initialization. For details on setting up the required AOFPRECOMPILED parameters and creating the pre-compiled AOF data set, see the *Parameter Reference*.

Note: Using AOF precompiled rules is optional. The volume of auto-enabled rules you have determines whether you need to use AOF precompiled rules.

DELCOMP

Deleting a compiled rule deletes the compiled versions from the AOF pre-compiled data set. For details on setting up the required AOF PRECOMPILED parameters and creating the pre-compiled AOF data set, see the *Parameter Reference*.

How to Control Rules from the Rule Set Level

Most AOF rule control is performed at the individual rule level but it can also be done at the rule set level.

For example, suppose the rule set OPSMVS.PROD.SUPPRESS.RULES contains all of your individual suppression rules and a situation arises in which you want to disable all suppression rules. Rather than having to disable each individual suppression rule that resides in the SUPPRESS rule set, you can disable the rule set itself.

To control rules at the rule set level, use the following guidelines:

- Enable an AOF rule set

  Enables only those rules that have the auto-enable bit set in the rule set.
- Disable an AOF rule set

  Disables all the rules that are currently enabled in the rule set.
- Auto-enabling an AOF rule set

  Sets the auto-enable bit for all rules in the rule set.

More information:

Using the AOF (see page 15)

Create and Control Rules Programmatically

You can programmatically control AOF rules using the OPS/REXX ADDRESS AOF host environment.

While AOF rules are mainly controlled manually using OPSVIEW option 4.5, the ADDRESS AOF OPS/REXX host environment provides you with the flexibility of controlling your rules in an automated application. This feature is commonly used as a tool in the form of a pseudo-AOF command rule that can be invoked from anywhere a z/OS command can be entered, which eliminates the need to use the OPSVIEW 4.5 option.

In addition to programmatically controlling rules, the ADDRESS AOF OPS/REXX host environment gives you a way to dynamically create AOF rules. The ADDRESS AOF host environment lets you build an AOF rule in other AOF rules or OPS/REXX programs. This type of logic is mainly used to trigger automation for events that can occur at any time.

To programmatically create and control rules

- Suppose your suppression rules are in a rule set called OPSMVS.PROD.SUPPRESS.RULES and you wanted to create a quick way to disable all suppression rules. You can create the following rule:

```
)CMD SUPPOFF
)PROC
/* Emergency pseudo command rule to disable all suppression */
/* Enter the command SUPPOFF from any system console to     */
/* trigger this rule.                                       */
/* Send DISABLE command to AOF for SUPPRESS ruleset         */
ADDRESS AOF
"DISABLE SUPPRESS"
```

- Suppose you want to invoke an OPS/REXX program that performs various checks against VTAM-related nodes and to trigger this program 10 minutes after VTAM initializes. You can code the following:

```
)MSG IST020I
 )PROC
 /* This message indicates that VTAM has initialized. We want     */
 /* to schedule our VTAM check EXEC 10 minutes from now. Since    */
 /* we never know in advance the exact time that VTAM will        */
 /* initialize, we create a dynamic AOF TOD rule to wake up 10    */
 /* minutes from the time this message was issued.                */
 queue ")TOD *+10 MINS"
 queue ")PROC"
 queue "ADDRESS OSF"
 queue "'OI PROGRAM(VTAMCHCK)'"
 address AOF
 "ENABLE *DYNAMIC.VTAMCHCK"
```

For more information on using the ADDRESS AOF OPS/REXX host environment to programmatically control and create AOF rules, see the *Command and Function Reference*.

AOFINITREXX Parameter

The recommended method for automatically enabling AOF rules at CA OPS/MVS initialization is to set the auto-enable flag, which is set in the ISPF directory of each rule in the rule data set. CA OPS/MVS checks the flag during initialization and enables those rules that have this flag set.

An alternative is to use the CA OPS/MVS AOFINITREXX parameter. This parameter specifies the name of an OPS/REXX program that gets control before the AOF process in CA OPS/MVS initializes. That OPS/REXX program can use the ADDRESS AOF host environment to enable your rules. If you are using a product that may modify the ISPF directory statistics of your AOF rules data set, such as CA Endevor SCM, then you must use the AOFINITREXX parameter to specify the name of the OPS/REXX program that enables your rules.

Note: For more information, see the *Parameter Reference*.

Process Modified AOF Rules

When CA OPS/MVS enables a rule in the production AOF environment, it validates syntax, compiles the rule, and then loads the compiled rule into the CA OPS/MVS address space for execution.

This means that if you edit your rules from native ISPF-that is, outside of the ISPF/PDF editor in OPSVIEW option 4.5-you need to disable then re-enable the rule to process the changes. Also, when preparing to rename a rule member, you must first disable the rule, rename it, and then re-enable it after renaming. If the modified rule is auto-enabled, you must reset the auto-enable flag.

Execution of Enabled Rules

The following sections discuss how to control the execution of enabled rules.

Protect Against Rule Errors

The AOF provides ways to protect against common programming problems in rules.

To protect against rule errors, you can set limits to protect against runaway (endlessly looping) rules and other programming errors.

Parameters for Setting Global AOF Rule Limits

The following CA OPS/MVS parameters affect all AOF rules:

AOFMAXCLAUSES and REXXMAXCLAUSES (REQ rules only)

Limits the number of OPS/REXX clauses that can execute in one rule invocation.

Default: 10000 and 1000000 respectively

AOFMAXSAYS and REXXMAXSAYS (REQ rules only)

Limits the number of OPS/REXX SAY statements in one rule invocation.

Default Values: 1000 and 100000 respectively

AOFMAXCOMMANDS and REXXMAXCOMMANDS (REQ rules only)

Limits the number of host commands that can execute in one rule invocation.

Default Values: 100 and 100000 respectively

AOFMAXSECONDS and REXXMAXSECONDS (REQ rules only)

Limits the amount of elapsed time that a rule has to execute for each invocation.

Default Values: 10 and unlimited respectively

AOFFIRELIMIT

Limits the number of times that a rule can execute in one minute. A value of 0 turns limit checking off (both globally and at the rule level).

Default Value: 10000

AOFLIMITDISABLE

A YES value disables a rule if the AOFFIRELIMIT limit has been reached. A NO value temporarily disables the rule until the AOFFIRELIMIT interval has expired.

Default Value: NO

MESSAGEMAX

Limits the number of OPSx messages issued on behalf of some work done by CA OPS/MVS.

Default Value: 1000

COMMANDMAX

Determines the total number of commands that CA OPS/MVS can issue per second.

Default Value: 200

Depending on the logic that you implement in your AOF rules, you may need to increase the default values of these parameters. For details, see the *Parameter Reference*.

Set Limits for Individual Rules

You can override *some* of the AOF rule limit values for *any* rule section.

To set limits for individual rules

1. Use the OPS/REXX OPTIONS statement in that rule section.

   Since the limit values are only changed when the OPTIONS statement is executed, OPTIONS should be one of the first statements executed in the rule section.

Note: For more information about the OPS/REXX OPTIONS instruction, see the chapter "Using OPS/REXX" in the *User Guide*.

How Multiple Rules Execute in Response to a Single Event

You can write any number of rules that respond to a single event. Except for time rules with exactly matching event criteria, rules execute in a predictable order, as shown:

1. Rules with the most specific event criteria are tested first.

   For example, message rules with most specific to least specific event specifiers are tested in this order:

   - IST020I (Seven significant characters-most specific)
   - IST*I (Four significant characters-less specific)
   - IST* (Three significant characters-least specific)

2. Event specifiers containing the same number of significant characters are tested in the order of longest prefix length.

   For example, if three event specifiers all contain six significant characters, they are tested in this order:

   - IST02*I (Five-character prefix)
   - IST0*0I (Four-character prefix)
   - IST*20I (Three-character prefix)

3. Rules containing identical event specifiers are tested in unpredictable order. If you want the rules to execute in a particular order, then combine them into a single rule.

   However, do not combine unrelated rules because doing so makes applications difficult to manage.

Note: If you have to continually rely on the above executing order, you may be developing an ineffective application. You should incorporate the logic of the multiple rules into one rule.

ABENDLOG Automation

ABENDLOG demonstrates how to collect an event (the logic uses IEF450I abend MLTWOs), store it in an OPS/MVS RDF table, and then offload the saved data into a sequential data set.

This type of automation creates reports with specific events. Utilization of the RDF table as an initial repository eliminates the possibility of bottlenecks within the automation, which occur when attempting to write/save each event directly to the sequential data set.

Chapter 5: Code and Debug AOF Rules

This section contains the following topics:

Coding Guidelines

The following sections discuss tools, procedures, and other pertinent information to help you create effective AOF rules.

Note: For information on how to implement common coding guidelines, see the CA OPS/MVS *User Guide.*

Automation Tools

CA OPS/MVS provides a variety of tools designed to help you build effective applications for automating all areas of your system environment. Understanding exactly what these tools can do is crucial when you build your AOF rules. Take time to familiarize yourself with the OPS/REXX built-in functions and OPS/REXX host environments because they are the primary tools that you use when creating AOF rules.

Note: For complete details of each of the OPS/REXX built-in functions and OPS/REXX host environments, see the *Command and Function Reference*. For an overview of the automation tools provided by the CA OPS/MVS base and optional components, see the *User Guide*.

How to Add Comments in AOF Rules

Good structured REXX code begins with detailed comments. Comments can assist those who review the rule and can help in debugging, if necessary.

Create a detailed comment model that can be implemented in all of your AOF rules.

Include detailed comments as follows:

- At the beginning of each rule that fully describe the following:
  - The logic of the rule
  - The variables
  - Any related OPS/REXX programs
- In the logic of the rule.

REXX Functions and Routines in AOF Rules

The *User Guide* and *Command and Function Reference* completely outline the differences of various standard REXX instructions and functions as they are used in any OPS/REXX environment, such as a program running in a CA OPS/MVS OSF TSO server or in an AOF rule.

The following are special guidelines for the SAY instruction, stem variables, and external subroutines.

How to Use the SAY Instruction

The destination of a SAY-generated message is different among rule types and often even among sections of a rule.

The only logical use of the SAY instruction is as follows:

- Direct message traffic to a user from a request rule or an OPS/REXX program running in the address space of that user.
- Debugging purposes, such as dumping the contents of a variable.

When you need to issue a message to a system console, consoles, or even directly to the SYSLOG, always use the OPS/REXX ADDRESS WTO host environment-*not the SAY instruction*. Also, you can use the OPS/REXX OPSSEND built-in function in rules to directly send messages only to the OPSLOG. These tools are far more effective than the SAY instruction because you have complete control of where the messages are directed.

Reference a Stem Variable in an INTERPRET Statement or VALUE Function

If a REXX INTERPRET statement or VALUE function references a stem variable that is represented as an OPS/REXX global variable (for example, GLVTEMP1.SomeName or GLOBAL.SomeName) or an AOF environmental variable applicable to the current rule (for example, MSG.TEXT or CMD.TEXT), then the stem variable is resolved only if a variable that has the same stem is directly referenced somewhere else in the rule section, even if that section is never executed.

Example: Stem Variable in an Interpret Statement

The value of the MSG.TEXT variable in the following example is “Test for OPS/MVS”.

- In this rule:

  ```
  )MSG TEST
  )PROC
  interpret "say msg.text"
  return "NOACTION"
  ```

  The result of the SAY instruction in the INTERPRET statement is:

  ```
  OPS1000I MSG.TEXT
  ```

- In this rule, there is a direct reference to the msg. stem (msg.id) in the)PROC section:

  ```
  )MSG TEST
  )PROC
  interpret "say msg.text"
  return "NOACTION"
  NeverExecuted = msg.id
  ```

  The result of the SAY instruction in the INTERPRET statement is:

  ```
  “Test for OPS/MVS”
  ```

Use External Subroutines

When enabling a rule, any referenced external subroutines called in AOF rules are compiled and placed in memory with the rule itself.

To use external subroutines

1. Change an external subroutine
2. Disable and re-enable the calling rule.
   - The change takes effect.
   - The AOF searches for external subroutine references in this order:
     a. The current rule set or the data set containing the rule
     b. The SYSEXEC concatenation of the OPSMAIN started task

Interactive Automation or Automation that Requires Waiting

A powerful feature of AOF rules is the ability to process inline, or synchronously, as an event occurs. Such realtime automation is possible because AOF rules execute in the address space from which an event occurred. Furthermore, the design of the OPS/REXX functions and host environments used in AOF rules not only lets you synchronously issue WTOs and system commands, but obtain vital system-related data needed to make programmatic decisions as an event occurs.

The only type of automation that you cannot perform in rules is interactive automation or automation that requires waiting. Examples are:

- Issuing a WTOR to a console and needing to interrogate the reply.

  Note: If no reply interrogation is needed, then issue the WTOR directly from the rule.

- Issuing a z/OS or UNIX System Services command and needing to interrogate the command output.

  Note: If no command response interrogation is needed, then issue the command directly from the rule. Also, verify that no OPS/REXX built-in function exists for the type of command you are attempting to interrogate. For example, to see if a job is active, you can use the synchronous OPSTATUS built-in function in the rule, rather than invoking an OPS/REXX program in a server to issue the z/OS DISPLAY ACTIVE command.

- Any type of file I/O.
- A cross-system request through the CA OPS/MVS MSF facility that causes output to be retrieved.

You can perform automation of this type by scheduling an OPS/REXX program to a CA OPS/MVS server. Use the OPS/REXX ADDRESS OSF host environment, 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  */

FROMNODE = WORD(MSG.TEXT,7)
TONODE = WORD(MSG.TEXT,9)
if FROMNODE ¬= 'A45PROD' then      /* NOT A45PROD          */
  RETURN

/* Trigger OPS/REXX program IST521I in OPSOSF TSO server   */
/* passing it the nodes as arguments. IST521I would be     */
/* located in the OPSEXEC or SYSEXEC concatenation of the  */
/* OPSOSF procedure.                                       */

ADDRESS OSF
"OI PROGRAM(IST52I1) ARG("NODE1 NODE2") "
```

Note: For details on calling OPS/REXX programs, see the *Command and Function Reference*.

More information:

Coding Each AOF Rule Type (see page 69)

Logic in Automated Applications

As you design your automated applications, avoid inefficient logic for triggering rules. Specifically, you want to avoid the triggering of an AOF rule as a result of an action taken by another AOF rule.

For example, do not issue a command in a message event rule (MSG) using the ADDRESS OPER host environment, and then try to have a command event (CMD) trigger on the command created by the ADDRESS OPER command.

Similarly, do not implement logic to trigger a MSG rule by an ADDRESS WTO host environment instruction that is issued by other AOF rules or OPS/REXX programs. Attempting to code logic in rule A that causes rule B to execute, and then having rule C execute on logic performed by rule B is not only confusing to maintain, but may also cause CA OPS/MVS to use process blocks unnecessarily. The process blocks are crucial to AOF processing and other CA OPS/MVS components.

If you have a piece of common logic that needs to be performed by various AOF rules and requires no waiting, then invoke this code as a REXX external subroutine.

Events Specified with the Wildcard Character

Many AOF rule types let you use the wildcard character (*) to specify events, for example)MSG * or)CMD *.

Performance may be degraded in CA OPS/MVS and the address space that generates the event if a rule:

- Uses the generic specifier (*)
- Contains a)PROC section that references either the OPS/REXX OPSVALUE function or OPS/REXX ADDRESS SQL host environment
- Contains lengthy programmatic logic

For example, the rule containing)MSG * will execute for every message on the system, including hard-copy messages and any message generated by any CA OPS/MVS facility (such as IOF) that you may have implemented.

More information:

Coding Each AOF Rule Type (see page 69)

Debugging Techniques

CA OPS/MVS has several tools and techniques you can use to debug problems in your AOF rules.

For debugging information for each type of AOF rule and additional information on the following parameters, see the chapter "Coding Each AOF Rule Type (see page 69)."

OPSLOG Facility

The CA OPS/MVS OPSLOG facility is a good source of information when debugging problems with your automated applications.

OPSLOG PROFILE

This command provides you with filtering capabilities that let you drill down to specific events.

OPSLOG DISPLAY

This command lets you view valuable internal data of all events such as the following:

- route codes
- job name
- job ID
- originating consoles
- event types

Depending on the logic implemented in your rules, the values in these columns can be compared to event variables that you are using in the rule.

The OPSLOG also provides an audit trail of all CA OPS/MVS activity, including messages that may indicate various execution errors in your applications. These messages begin with OP*x* where *x* is the fourth character of the subsystem ID of your production CA OPS/MVS, for example, S for the default subsystem OPSS.

Becoming proficient with these OPSLOG tools will help you to debug your applications quickly.

Note: For more information on the OPSVIEW OPSLOG option, see the *OPSVIEW User Guide*.

REXX TRACE Built-in Function

Use the REXX TRACE instruction for debugging logic problems in your AOF rules. The TRACE instruction is the primary tool for determining why a rule is not doing what you expect. Use this statement for debugging purposes only. Having the TRACE statement in frequently triggered AOF rules will flood the OPSLOG with trace messages.

You can view the trace output in the OPSLOG. Depending on the TRACE setting, the function can resolve expression results, variable values, results of function calls, and so on. The most commonly used TRACE setting is R, which you can implement in a rule by coding the REXX statement TRACE R.

Be sure to include the TRACE statement in the appropriate section of the rule you are debugging. For example, coding a TRACE R statement in the)INIT section causes only the)INIT section code to produce trace output.

Note: For more details regarding the TRACE instruction, see any REXX guide.

How RULETRACE Parameter Works

More than one AOF rule can process a single system event, which can often lead to conflicting logic.

To determine if you have multiple AOF rules executing on the same event, display the following columns in the OPSLOG facility:

COUNT

Displays the number of rules that executed on the event.

RULESET

Displays the rule set and the rule that executed on the event first (or the only rule that executed if COUNT=1).

By default, the OPSLOG records only the first rule that executed on the event. If COUNT > 1 and you need to reveal all rules that processed the event, do the following:

1. Set the RULETRACE parameter to ON.

   Use this setting for debugging purposes only.

2. Reinitiate the event.

This parameter setting causes the OPSLOG to record each AOF rule that processed the event.

You can set the RULETRACE parameter manually using OPSVIEW option 4.1.1 or programmatically using the OPS/REXX OPSPRM built-in function.

BROWSExxx Parameter

Many of the AOF rules can optionally record event data in the OPSLOG by setting a unique BROWSE*xxx* parameter. For example, if you are debugging a particular EOJ rule, you can set the BROWSEEOJ parameter to YES to cause EOJ events to be recorded in the OPSLOG.

More information:

Coding Each AOF Rule Type (see page 69)

Chapter 6: Coding Each AOF Rule Type

This section contains the following topics:

Generic Event Application Program Interface

The CA OPS/MVS generic event Application Program Interface (API) enables CA software products to trigger CA OPS/MVS events directly. CA OPS/MVS responds to API-generated events by executing AOF rules. The interface is synchronous, meaning all rules that execute due to API activity complete execution before control returns to the application.

Coding guidelines are described for each type of AOF rule. The following specific information is presented in alphabetical order by rule type for all AOF rule types:

- Event specifier options
- Additional implementation steps
- Available AOF event variables

Traditionally, CA software products triggered a CA OPS/MVS event by issuing a console message, which was trapped by CA OPS/MVS in the subsystem interface and processed in a MSG rule. Using the API instead of issuing a console message reduces some of the overhead associated with messages. In addition, the information passed to CA OPS/MVS is not constrained by message restrictions such as message length. Generic events are processed by API type rules.

For each event generated by an application, the application provides CA OPS/MVS the information needed to create a set of environmental variables related to that event. There is a set of common environmental variables that are generically described in this chapter. However, each application may provide more detailed information about these variables in its own documentation set. In addition, see the documentation of the calling application for details of the application-specific variables that are created by each application for each event.

More information:

Summary of AOF Coding Guidelines (see page 363)

Install and Activate API Rules

Certain conditions must be satisfied before you can activate API rules.

To activate API rules

1. Set the APIACTIVE parameter to ON in CA OPS/MVS.

   The API is enabled. For example, a production image and a few test images.

   Note: Use caution when enabling APIACTIVE on more than one system. Duplicate event processing could occur.

2. Write and enable the API rules.

   CA OPS/MVS can now process the application events.

Event Specifier of API Rules

API rules process all events that are generated through the API. To write an API rule, you must know the name of the event you want to process and the names of the environmental variables that are available to the rule when the event occurs. The event name can be up to 10 characters long and is used on the API statement. For example:

```
)API EVNAME
```

A rule may or may not be coded for each event name the application creates. Wildcard matches can be used to allow a single rule to react to more than one event. It is possible for two or more rules to react to a single event. For example, EVENT1 and EVEN* both match event EVENT1.

You must know the format of the data that the application places in environmental variables that are available to your rule, whether each environmental variable is read-only or read/write, and the format of the data that the application should receive from CA OPS/MVS.

Initialization, Processing, and Termination Sections of API Rules

API rules follow the general guidelines for coding the initialization, processing, and termination sections and the various AOF tools that you can use.

More information:

AOF Rule Structure (see page 21)
AOF Rule Tools (see page 29)

RETURN Statements in the)PROC Section of an API Rule

The OPS/REXX RETURN statement has no special meaning in the processing section of an API rule. The return value has no effect on AOF processing.

Execution Considerations for API Rules

The processing section of an API rule executes in the address space of the application that generated the event. Therefore, any complex logic or interactive logic that may cause a wait to occur should be done in an OPS/REXX program that gets scheduled to an OSF TSO server on behalf of the API rule. For a discussion of dispatching OPS/REXX programs to OSF TSO servers, see the chapter "Code and Debug AOF Rules (see page 59)."

The AOF execution limits apply to the processing section of a rule that responds to a screen event.

More information:

Building and Controlling AOF Rules (see page 49)

OPS/REXX Host Environments in the)PROC Section of an API Rule

The)PROC section of an API rule has the following host environments with the following API rule characteristics. The AOFDEFAULTADDRESS parameter specifies the default host environment for the API rules.

ADDRESS AOF

Sent to CA OPS/MVS. Does not wait. Output is not returned.

ADDRESS AP

Sent to MSF and then forwarded to the CA Automation Point system. Does not wait. Output is not returned.

ADDRESS EPI

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS HWS

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS ISPEXEC

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS LXCON

The VM and Linux command requests sent to USS server for execution. Does not wait. Output is not returned. The List request runs inline and returns VM and Linux system data in stem variables.

ADDRESS MESSAGE

Sent as a WTO. The AOFDEST parameter specifies the destination.

ADDRESS MIM

The request is sent to CA MIM. Output is returned in stem variables. Error messages, if any, are returned to an external data queue.

Note: Within a rule, storage demands are limited to 32 K. QNAME and TAPE requests for all available information can fail. Therefore, name filtration is highly recommended.

ADDRESS MQ

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS NETMAN

Sent to a CA OPS/MVS internal CA Netman request queue to issue MGPT commands. Does not wait. Output is not returned.

ADDRESS NETMASTR

Sent to CA NetMaster NM for SNA on the local system. Does not wait. Output is not returned.

ADDRESS OPER

Sent to target console as specified through the OCCONSOLENAME parameters when no CONNAME operands are present. Output is not returned. Schedule an OPS/REXX program in a server when a command output interrogation is needed.

ADDRESS OPSCTL

Sent to the specified facility. If the facility is ECF or OSF, does not wait. If the facility is MSF, slight wait occurs. The external data queue returns the output.

Note: If the command is MSF LIST, no wait occurs.

ADDRESS OPSDYNAM

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS OSF

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSO servers.

ADDRESS OSFTSL

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSL servers.

ADDRESS OSFTSP

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSP servers.

ADDRESS SOF

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS SQL

Does not wait. Processed synchronously for requests that can be satisfied on the local system. Output is returned in stem variable. Error messages, if any, are returned to an external data queue.

ADDRESS SYSVIEWE

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS TSO

Sent to an OSF TSO server. Does not wait. Output is not returned.

ADDRESS USS

Sent to a USS server. Does not wait. Output is not returned. Schedule an OPS/REXX program in a server if a command output interrogation is needed.

ADDRESS WTO

Does not wait. Output is sent to specified (or default) destination. When attempting a WTOR, host command is sent to a TSO server for execution. The response is then returned to the server. Schedule an OPS/REXX program in a server if a WTOR response interrogation is needed.

More information:

Code and Debug AOF Rules (see page 59)

Common API Event Variables

The following common API event variables are supplied to every API rule:

API.APPLICATION

The application identifier string as defined by the application and registered with CA OPS/MVS development to ensure uniqueness

Data Type: Character, read-only

Sample Value: PDSMAN

API.COLOR

The color that certain messages will display in OPSLOG Browse

Data Type: 1-byte binary (unprintable), read/write

Sample Value: '00'X

Note: For additional information on setting this environmental variable, see the chapter "OPS/REXX Built-In Functions" in the *Command and Function Reference*.

API.ID

When a rule executes due to a wildcard match, API.ID can be used to determine which event caused the rule to execute.

The complete event name that the rule is processing

Data Type: Character, read-only

Sample Value: XXXXCLOSE

API.LEVEL

A 1- to 8-character string provided by the application. One intended purpose of this variable is to allow the application to differentiate between multiple copies of the application executing on the same system, if it is possible to do so. Otherwise, the application may or may not provide information in this variable.

Data Type: Character, read-only

Sample Value: 1

API.USER

Before AOF processing, this variable is initialized to binary zeros. It is then passed to each rule that executes for the same API event; each rule can look at or change the variable contents before passing the variable to the next rule for the API event.

The primary purpose of the USER variable is to provide a method to pass a small amount of data between the rules. This data may be in any format. The USER field may also be used for filtering in the OPSLOG; however, USER data used for OPSLOG filtering must be uppercase and displayable.

An 8-byte variable providing communication between rules that execute for the same API event. The variable can contain any installation data that these rules need, and it can store a character string displayable through OPSLOG Browse.

Data Type: User-defined, read/write

API.VERSION

A 1- to 8-character string provided by the application to identify the version of the application that generated the API event. Note that different versions of the application may provide different environmental variables. In such cases, the rule writer can use this variable to determine which application version-specific environmental variables to use.

Data Type: Character, read-only

Sample Value: 05.01

API.TEXT

This variable is not usually used by the API rules. Its primary purpose is to provide a readable explanation of the event in the OPSLOG.

The first 128 characters of data of this variable are displayed in OPSLOG.

Determined by the application

Data Type: Character, read-only

Sample Value: XXXX EVENT RECORDED 10MAY2011

API.suffix—Specific API Event Variables

In addition to the common API event variables, each application that uses the API defines its own application-specific variables.

These variables have the following format:

```
API.suffix
```

suffix

Specifies a length of 1 to 15 characters.

The names and characteristics of the specific API event variables are supplied in documentation provided by the application. There may be a few specific variables or as many as 256. Individual variables may hold up to 4096 bytes in length, and the total data length allowed by the API is 32768 bytes per event. Specific API variables may be read-only or read/write variables.

Debug an API Rule

The following are additional API rule debugging techniques:

- To enable error messages from the API processor, set OPSPARM DEBUGAPI YES .
- To view all API events, set the CA OPS/MVS BROWSEAPI parameter to YES and the API event profile of your OPSLOG display to Y. To see recorded API events with these parameters set, display the OPSLOG EVENT column.
- If OPSLOG is not recording API events, see Install and Activate API Rules (see page 70).

More information:

Code and Debug AOF Rules (see page 59)

SOF API Rules

The Switch Operations Facility (SOF) generates API events to report status changes to CA OPS/MVS, which responds by either logging the information or taking action in response to the information. You can write)API rules that specify how CA OPS/MVS responds to API-generated events.

SOF produces the following two events through the API:

- OPSOF001-Status change for devices
- OPSOF002-Status change for device paths

OPSOF001 Event—Device Status Change

The OPSOF001 event indicates that a switch operation has resulted in a status change for one or more devices. The API.TEXT environmental variable contains the details of the change.

This event has the following format:

```
OPSOF001  cmd  status  dev1  dev2
```

cmd

Identifies the name of the switch command that caused the status change.

Valid Values: BLOCK, UNBLOCK, CONNECT, DISCONNECT, ALLOW, PROHIBIT, SYNC, or ACTIVATE

status

Specifies the new status for the devices, which can be one of the following:

- ONLINE indicates that the device is now online.
- OFFLINE indicates that the device is now offline.
- AVAILABLE indicates that the device is now available, but was not brought online by SOF because either the NOVARY option was specified or the device does not match any of the device classes specified on the VARYCLASS option.

dev1

Specifies the device number for the first device to undergo a status change.

dev2

Specifies the device number for the last device to undergo a status change. When only a single device is affected, *dev2* and *dev1* will be the same.

OPSOF002 Event—Device Path Status Change

The OPSOF002 event indicates that a switch operation has resulted in a status change for one or more device paths. The API.TEXT environmental variable contains the details of the change.

This event has the following format:

```
OPSOF002  cmd  status  chpid  dev1  dev2
```

cmd

Identifies the name of the switch command that caused the status change.

Valid Values: BLOCK, UNBLOCK, CONNECT, DISCONNECT, ALLOW, PROHIBIT, SYNC, or ACTIVATE

status

Specifies the new status for the device paths. The status can be one of the following:

- ONLINE indicates that the device is now online.
- OFFLINE indicates that the device is now offline.
- AVAILABLE indicates that the device is now available, but was not brought online by SOF because the NOVARY option was specified.

chpid

Specifies the ID of the channel path that changed status.

dev1

Specifies the device number for the first device to undergo a status change.

dev2

Specifies the device number for the last device to undergo a status change. When only a single device is affected, *dev2* and *dev1* will be the same.

OPSOF003 Event—Connectivity Command Accepted

The OPSOF003 event indicates that a switch operation command, which can alter device or path status, has been accepted. The API.TEXT environmental variable contains the details of the command.

This event has the following format:

```
OPSOF003  origin  cmd  operands
```

origin

Identifies the origin of the command, that is, the console or TSO user that issued the command.

cmd

Identifies the name of the command.

operands

Provides the full text of the command operands.

How OPSOF001 Rule Selectively Varies Devices Online

Event OPSOF001 indicates that a switch operation has resulted in a status change for one or more devices.

When SOF makes connectivity changes, the following process occurs:

1. The generic event OPSOF001 is issued for each contiguous block of devices affected by the command.
2. The rule looks for devices which are now available.

   This indicates that the one or more paths for the device should now be functional, but the devices were not brought online by SOF.
3. The rule then examines the device numbers and issues a VARY ONLINE command for only devices falling within a specific range of device numbers, for example, 8902-8907.

Example: API Rule for the OPSOF001 Event

The following sample API rule selectively varies online devices affected by a CA OPS/MVS SOF switch command.

```
)API OPSOF001
)PROC
  DEVSTAT = word(API.TEXT, 3)
  LOWDEV = x2d('8902')
  HIDEV  = x2d('8907')
  if DEVSTAT == 'AVAILABLE' then do
     DEV1 = x2d(word(API.TEXT,4))
     DEV2 = x2d(word(API.TEXT,5))
/*-------------------------------------------------------------------*/
/* If no devices are in our range, we have nothing to do, so quit.  */
/*-------------------------------------------------------------------*/
     if DEV1 > HIDEV  |  DEV2 < LOWDEV then return 0
/*-------------------------------------------------------------------*/
/* Remove device numbers that are too low                           */
/*-------------------------------------------------------------------*/
     if DEV1 < LOWDEV  then DEV1 = LOWDEV
```

```
/*-------------------------------------------------------------------*/
/* Remove device numbers that are too high                           */
/*-------------------------------------------------------------------*/
     if DEV2 > HIDEV then DEV2 = HIDEV
/*-------------------------------------------------------------------*/
/* Construct the VARY command                                        */
/*-------------------------------------------------------------------*/
     CMD = 'VARY ' || d2x(DEV1,4) || '-' || d2x(DEV2,4) || ',ONLINE'
/*-------------------------------------------------------------------*/
/* Issue the command                                                 */
/*-------------------------------------------------------------------*/
     address oper  "COMMAND('" || CMD || "')"
     end
  return 0
```

Hardware Event API Rules

CA OPS/MVS Hardware Services (HWS) interfaces with the Hardware Interface Service to deliver hardware event notifications and their associated data as OPS API events. You can write)API rules that specify how CA OPS/MVS responds to API-generated hardware events. Every hardware API event ID begins with the prefix HWS. This allows you to write API rules for specific hardware events or a single rule for all events (example:)API HWS*)

For more information on Hardware Services, see Hardware Services in the *CA OPS/MVSEvent Management and Automation Administration Guide*.

Note: Generally, HWS hardware events should only be received and automated by one CA OPS/MVS in the hardware (HMC) network to avoid duplicate automation of events.

Entity-related Hardware Events

The following *entity-related* hardware events are available through API rules. An entity is an object within the hardware (HMC) network such as a CPC, Image, or Activation Profile.

Activation Profile Change

An activation profile has been changed.

Application Ended

An application has ended

Application Started

An application has started

Attribute Added/Updated

An attribute has been added/updated for an entity.

Capacity Change

A capacity change has been detected.

Capacity Record

A capacity record event has been detected.

Command Response

A command response has been received for a command issued against an entity.

Disabled Wait

A disabled wait has been detected.

Entity Exception

An entity exception has been detected.

Hardware Communication Error

A hardware communication error has occurred.

Hardware Message

A hardware message has been issued.

Hardware Message Delete

A previously issued hardware message has been deleted.

New Child

The entity has a new child.

New Entity

A new entity has been detected.

Operating System Message

An operating system message has been issued.

Power Change

A power change has been detected.

Security Event

A security event has occurred.

Status change

A status change has occurred.

System-related Hardware Events

The following *system-related* hardware events are available through the API. A system event relates to the infrastructure providing hardware events rather than to the entities in the hardware network. For example, system events may indicate a change in the status of the underlying interface used by the Hardware Interface Service.

Hardware Interface Up

An underlying hardware interface used by the Hardware Interface Service has become active. For example, this event will be produced when the Hardware Interface Service detects that BCPii has become active.

Hardware Interface Down

An underlying hardware interface used by the Hardware Interface Service has terminated. For example, this event will be produced when the Hardware Interface Service detects that BCPii has terminated.

Topology Complete

The Hardware Interface Service has completed its hardware topology discovery.

Topology Error

The Hardware Interface Service encountered an error during topology discovery.

OPS/REXX Rule Variables For Hardware Events

All variables listed below are available for all event types. However, some variables are not applicable for all events. If a variable represents event data that is not applicable for a particular event, the variable will have a zero length for that event.

API.ALARMMSGFLG

Description

Indicates alarm message.

Data Type

Character, read-only, 1 character.

Values

0

Not an alarm message.

1

Alarm message.

Example

0

Applicable events

HWSOSMSG

API.APPLICATION

Description

Name of application providing the event.

Data Type

Character, read-only, up to 8 characters.

Values

Application name

Example

HIS

Note: HIS is the Hardware Interface Service

Applicable events

All events

API.CAPCHVAL

Description

Capacity change value.

Data Type

Character, read-only, up to 10 characters.

Values

Capacity change value.

Example

2

Applicable events

HWSCAPCHG

Capacity Change Values

0	FENCED_BOOK
1	DEFECTIVE_PROCESSOR
2	CONCURRENT_BOOK_REPLACE
3	CONCURRENT_BOOK_ADD
4	CHECK_STOP
5	CHANGES_ALLOWED
6	CHANGES_NOT_ALLOWED

API.CAPRECVL

Description

Capacity record value.

Data Type

Character, read-only, up to 10 characters.

Values

Capacity record value

Example

3

Applicable events

HWSCAPREC

Capacity Record Values

0

RECORD_ADD

1

RECORD_DELTA

2

RECORD_DELETE

3

RECORD_ACCOUNTING

4

ACTIVATION_LEVEL

5

PRIORITY_PENDING

6

RECORD_OTHER

API.CMDLASTFLG

Description

Indicates command last.

Data Type

Character, read-only, 1 character.

Values

0

Flag is off.

1

Flag is on.

Example

0

Applicable events

HWSCMDRESP

API.CMDRC

Description

Command return code.

Data Type

Character, read-only, up to 10 characters.

Values

Return Code

Example

12

Applicable events

HWSCMDRESP

API.CMDTYPE

Description

Command type.

Data Type

Character, read-only, up to 10 characters

Values

Command type

Example

5

Applicable events

HWSCMDRESP

API.DWPARTID

Description

Disabled wait partition ID.

Data Type

Character, read-only, up to 10 characters

Values

Partition ID

Example

4

Applicable events

HWSDBLWAIT

API.DWPROCNUM

Description

Disabaled wait processor number

Data Type

Character, read-only, up to 10 characters

Values

Processor number

Example

1

Applicable events

HWSDBLWAIT

API.DWPSW

Description

Disabled wait PSW.

Data Type

Character, read-only, up to 32 characters.

Values

PSW

Example

00020000000000000000000000009003

Applicable events

HWSDBLWAIT

API.DWSERIALNO

Description

Disabled wait serial number.

Data Type

Character, read-only, up to 16 characters

Values

Serial number

Example

00001316F574

Applicable events

HWSDBLWAIT

The variables API.NUMENTRIES, API.ELEVELx, API.ETYPEx, and API.ENAMEx describe the *hierarchy* of entities for an event. For each entity related event, the entity level, type and name are returned for the entity directly related to the event. In addition, the entity level, type and name are returned for each entity in the event entity hierarchy. That is, the level, type and name are returned for each parent, grandparent, and any other entity of the event entity. There can be up to 10 entities defined in the entity hierarchy.

Variables API.ELEVELx, API.ETYPEx, and API.ENAMEx where x is 1 through 10 are always defined for each event.

Variable API.NUMENTRIES indicates how many entities are actually returned in the hierarchy.

API.ENAME1, APIETYPE1, and APIELEVEL1 will contain the information for the entity which is directly related to the event. Starting at API.ENAME2, API.ETYPE2, and API.ELEVEL2, the variables will contain information related to the parent, grandparent, and any other entity. The entities go up the topology hierarchy.

Example

If API.NUMENTRIES=2, two entities are returned in the hierarchy. Variables API.ELEVEL1, API.ETYPE1, API.ENAME1, API.ELEVEL.2, API.ETYPE2, and API.ENAME2 will be filled in with entity information.

Variables API.ELEVELx, API.ETYPEx, and API.ENAMEx where x is 3-10 will have a zero length.

API.ELEVELx

Description

Level of entity in the returned entity hierarchy.

Data Type

Character, read-only, up to 10 characters.

Values

Level of entity in the hierarchy

Example

3

Applicable events

Entity related events

API.ENAMEx

Description

Name of entity in the returned entity hierarchy

Data Type

Character, read-only, up to 32 characters.

Values

Name of entity

Example

IBM500EX.SY01

Applicable events

Entity related events

API.ETYPEx

Description

Type of entity in the returned entity hierarchy.

Data Type

Character, read-only, up to 10 characters.

Values

Entity type values:

1

Enterprise or Root

5

Installation or Data Center

10

Ensemble

12

Machine

15

CPC

16

zBX

20

LPAR

21

Capacity Record

22

Reset Activation Profile

23

Image Activation Profile

24

Load Activation Profile

30

Sysplex

31

System

32

Coupling Facility

Example

15

Applicable events

Entity related events

API.EVDATE

Description

Event date.

Data Type

Character, read-only, 10 characters.

Values

Format: YYYY/MM/DD

Example

2011/04/01

Applicable events

All events.

API.EVDESC

Description

Event description.

Data Type

Character, read-only, up to 60 characters.

Values

Each event has its own description. See the values listed with individual events in the Event ID, Associated Entity, and Description Table below.

Example:

ENTITY STATUS CHANGE

Applicable events

All events.

API.EVLEVEL

Description

Entity hierarchy level.

Data Type

Character, read only, up to 10 characters.

Values

0 for system related event, otherwise level of event entity in hierarchy.

Example

6

Applicable events

All events

API.EVTIME

Description

Event time

Data Type

Character, read-only, up to 11 characters.

Values

Format: HH:MM:SS.TH

Example

10:53:56.97

Applicable events

All events

API.EXCSTATE

Description

Exception State.

Data Type

Character, read-only, 1 character.

Value

0

Not exception state

1

Exception state

Example

0

Applicable events

HWSENTEXC

API.HELDMSGFLG

Description

Indicates held message

Data Type

Character, read-only, 1 character

Values

0

Not a held message

1

Held message

Example

0

Applicable events

HWSOSMSG

API.HWIFNAME

Description

Hardware I/F name. This is the name of the underlying hardware interface being used by the application in API.APPLICATION.

Data Type

Character, read-only, up to 16 characters.

Values

Name of underlying interface.

Example

BCPII

Applicable events

HWSINTFUP, HWSINTFDWN

API.ID

Description

Name of the event.

Data Type

Character, read-only, up to 10 characters.

Values

Each event has its own event ID. The event ID is the value used to *match* API rules. See the values listed with individual events in the Event ID, Associated Entity, and Description Table below. All event IDs start with the HWS prefix.

Example

HWSSTATCHG

Applicable events

All events.

API.IMGLIST

Description

Image List

Data Type

Character, read-only, up to 32000 characters.

Values

List of images

Example

TS03 TS12 TS22 TS30 COUPLEA1 COUPLEA2

Applicable events

HWSHWMSG, HWSHWMSGD

API.ITOPIP

Description

Indicates event was issued while topology initialization was in progress by application in API.APPLICATION.

Data Type

Character, read-only, 1 character.

Values

0

Topology initialization not in progress when event issued

1

Topology initialization was in progress when event issued

Example

0

Applicable events

HWSNEWCHLD, HWSNEWENT

API.LEVEL

Description

Reserved for future use.

Data Type

Character, read-only, up to 8 characters.

Values

Text indicating reserved field.

Example

RESERVED

Applicable events

All events.

API.MSGDATE

Description

Message Date

Data Type

Character, read-only, 8 characters.

Value

Format: YYYYMMDD

Example

20110402

Applicable events

HWSOSMSG

API.MSGID

Description

Message ID

Data Type

Character, read-only, up to 16 characters

Values

ID of message

Example

187898401

Applicable events

HWSOSMSG

API.MSGTEXT

Description

Message Text.

Data Type

Character, read-only, up to 32000 characters.

Values

Text of message. May include multiple lines separated by the EBCDIC ‘NL’ (X’15’) character. API.MTNUML specifies the number of lines and can be used in conjunction with the ‘NL’ character to parse into individual lines if desired. The ‘NL’ character is only present if there are two or more lines.

Example

The reset profile RESETSY01. was changed.

Applicable events

HWSOSMSG, HWSHWMSG, HWSSECUR

API.MSGTIME

Description

Message Time

Data Type

Character, read-only, 8 characters.

Value

Format: HHMMSSTH

Example

17015043

Applicable events

HWSOSMSG

API.MSGTS

Description

Message time stamp.

Data Type

Character, read-only, up to 32000 characters.

Values

Time stamp

Example

03-28-2011 16:14:03:806

Applicable events

HWSHWMSG, HWSSECUR

API.MTNUML

Description

Number of lines in API.MSGTEXT.

Data Type

Character, read-only, up to 10 characters.

Values

0 if no message text is included with the event, 1 or more if message text is included.

Example

2

Applicable events

HWSHWMSG, HWSOSMSG, HWSSECUR

API.NEWETYPE

Description

New entity type.

Data Type

Character, read-only, up to 10 characters.

Values

See Values under API.ETYPEx (see page 91)

Example

15

Applicable events

HWSNEWCHLD, HWSNEWENT

API.NEWMSGFLG

Description

Indicates new message.

Data Type

Character, read-only, 1 character.

Values

0

Not a new message.

1

New message.

Example

0

Applicable events

HWSOSMSG, HWSHWMSG

API.NEWNAME

Description

New name.

Data Type

Character, read-only, up to 32 characters.

Values

Name

Example

SY03

Applicable events

HWSNEWCHLD, HWSNEWENT, HWSACTPCHG

API.NEWPMV

Description

New power mode value.

Data Type

Character, read-only, up to 10 characters.

Values

Power mode value

Example

4

Applicable events

HWSPOWCHG

API.NEWPSMA

Description

New power save mode allowed.

Data Type

Character, read-only, up to 10 characters

Values

Power mode allowed.

Example

3

Applicable events

HWSPOWCHG

API.NEWSTATUS

Description

New status value.

Data Type

4 byte binary (unprintable), read-only.

Values

See Status Values (see page 102).

Example

'00000002'X

Applicable events

HWSSTATCHG

Status Values

'00000001'X

OPERATING

'00000002'X

NOT_OPERATING

'00000004'X

NO_POWER

'00000008'X

NOT ACTIVATED

'00000010'X

EXCEPTIONS

'00000020'X

STATUS_CHECK

'00000040'X

SERVICE

'00000080'X

LINKNOTACTIVE

'00000100'X

POWERSAVE

'00000200'X

SERIOUSALERT

'00000400'X

ALERT

'00000800'X

ENVALERT

'00001000'X

SERVICE_REQ

'00002000'X

DEGRADED

'01000000'X

STORAGE_EXCEEDED

'02000000'X

LOGOFF_TIMEOUT

'04000000'X

FORCED_SLEEP

'08000000'X

IMAGE_NOT_OPERATING

'10000000'X

IMAGE_NOT_ACTIVATED

'20000000'X

IMAGE_NOT_CAPABLE

API.NUMENTRIES

Description

Number of entries for entity hierarchy.

Data Type

Character, read-only, up to 10 characters.

Values

A number between 0 and 10.

Example

4

Applicable events

All events, but generally 0 for system related events.

API.OLDNAME

Description

Old/prior name.

Data Type

Character, read-only, up to 32 characters.

Values

Name

Example

COUPLEA1

Applicable events

HWSACTPCHG

API.OLDPMV

Description

Old power mode value.

Data Type

Character, read-only, up to 10 characters.

Values

Power mode value.

Example

5

Applicable events

HWSPOWCHG

API.OLDPSMA

Description

Old power save mode allowed.

Data Type

Character, read-only, up to 10 characters.

Values

Power mode allowed.

Example

6

Applicable events

HWSPOWCHG

API.OLDSTATUS

Description

Old/prior status value.

Data Type

4 byte binary (unprintable), read-only.

Values

See Status Values (see page 102)

Example

'00000004'X

Applicable events

HWSSTATCHG

API.OPSSSNA

Description

Name of CA OPS/MVS subsystem receiving the event.

Data Type

Character, read-only, 4 characters.

Value

CA OPS/MVS subsystem name.

Example

OPSS

Applicable Events

All events.

API.OSNAME

Description

Operating system instance name.

Data Type

Character, read-only, up to 16 characters.

Values

Name of the operating system.

Example

SY11

Applicable events

HWSOSMSG

API.PERMHWEFLG

Description

Indicates permanent hardware error.

Data Type

Character, read-only, 1 character.

Values

0

Not permanent error (temporary error).

1

Permanent error

Example

1

Applicable events

HWSHWCOMER

API.PRMPT

Description

Prompt text.

Data Type

Character, read-only, up to 32,000.

Values

Text of prompt

Example

NULL

Applicable events

HWSOSMSG

API.PRTYMSGFLG

Description

Indicates priority message.

Data Type

Character, read-only, 1 character.

Values

0

Not a priority message.

1

Priority message.

Example

0

Applicable events

HWSOSMSG

API.SYSTEMFLG

Description:

Indicates system or entity related event.

Data Type

Character, read only, 1 character.

Values

0

Entity related event.

1

System related event.

Example

0

Applicable events

All events.

API.TEXT

Description

Text for event.

Data Type

Character, read-only, up to 128 characters.

Values

Order of text fields:

- Event/Rule ID
- Event Description
- Current Entity type
- Current Entity Name
- Message text (if any)

Example

HWSSECUR   SECURITY EVENT CPC IBM500EX.SY01 The reset profile RESETSY01. was changed.

Applicable events

All events.

API.VERSION

Description

Version of application identified in API.APPLICATION.

Data Type

Character, read-only , up to 8 characters.

Values

For application HIS (Hardware Interface Service) the version is in the form VV.MM.RR.

Example

01.00.09

Note: This is example in the format used by the HIS application.

Applicable events

All events.

Event ID, Associated Entity and Description Table

Event Name	Event ID (API.ID value)	Associated Entity Type	Event Description (API.EVDESC value)
Activation Profile Change	HWSACTPCHG	CPC/LPAR	ACTIVE PROFILE CHANGED
Application Ended	HWSAPPEND	CPC	APPLICATION ENDED
Application Started	HWSAPPSTRT	CPC	APPLICATION STARTED
Attribute Added/Updated	HWSATRUPDT	Any	ATTRIBUTES ADDED OR UPDATED
Capacity Change	HWSCAPCHG	CPC	CAPACITY CHANGED
Capacity Record	HWSCAPREC	CPC	CAPACITY RECORD CHANGE
Command Response	HWSCMDRESP	CPC/LPAR	ENTITY COMMAND RESPONSE
Disabled Wait	HWSDBLWAIT	LPAR	DISABLED WAIT
Entity Exception	HWSENTEXC	CPC/LPAR	ENTITY EXCEPTION
Hardware Communication Error	HWSHWCOMER	CPC	HARDWARE COMMUNICATIONS ERROR
Hardware Interface Down	HWSINTFDWN	N/A	SERVER HARDWARE INTERFACE DOWN
Hardware Interface Up	HWSINTFUP	N/A	SERVER HARDWARE INTERFACE UP
Hardware Message	HWSHWMSG	CPC	HARDWARE MESSAGE ISSUED BY ENTITY
Hardware Message Delete	HWSHWMSGD	CPC	HARDWARE MESSAGE DELETED
New Child	HWSNEWCHLD	Any	NEW CHILD UNDER ENTITY
New Entity	HWSNEWENT	Any	NEW ENTITY CREATED
Operating System Message	HWSOSMSG	LPAR	OPERATING SYSTEM MESSAGE

Event Name	Event ID (API.ID value)	Associated Entity Type	Event Description (API.EVDESC value)
Power Change	HWSPOWCHG	CPC	POWER CHANGE
Security Event	HWSSECUR	LPAR	SECURITY EVENT
Status Change	HWSSTATCHG	CPC/LPAR	ENTITY STATUS CHANGE
Topology Complete	HWSTOPCOMP	N/A	INITIAL H/W TOPOLOGY COLLECTION COMPLETE
Topology Error	HWSTOPERR	N/A	INITIAL H/W TOPOLOGY COLLECTION HAD ERRORS
Other (See Note)	HWSOTHER	*varies*	*varies*

Note: The Other event is generated when CA OPS/MVS receives an unknown event type. The Other event type is designed to handle new Hardware Interface Service events that have not yet been defined as their own HWS event type. For example, if the Hardware Interface Service adds support for a new event and delivers the event notification to CA OPS/MVS, CA OPS/MVS will see the event and generate it as an HWSOTHER event with associated event data. Once CA OPS/MVS adds support for the new event type, it will have its own HWS event type and will no longer fall under the Other event. Implementing the Other event type in this way allows CA OPS/MVS to receive and process new events as they become available.

Linux Connector API Rules

CA OPS/MVS Linux Connector interface (LXCON) connects with the Linux Connector component through a local IP connection. This interface delivers unsolicited message events from monitored VM and Linux systems as normalized messages that are processed as CA OPS/MVS API events. Write API rules that specify how CA OPS/MVS can respond to these Linux and VM events. Typically, the System State Manager (SSM) component of CA OPS/MVS is used to monitor and control the availability of Linux systems that run as VM guest computers. The Address LXCON host command can be used to display any connected VM and Linux systems and to issue commands to the systems. Every Linux Connector API event ID begins with a common prefix, *LX*. Write individual API rules for specific LXCON events or a single rule for all events with the rule specification:

```
)API LX*
```

For more information about the Linux Connector interface and component, see the Linux Connector Service in the *CA OPS/MVS Administration Guide.*

Note: Generally, Linux Connector unsolicited message events should only be received and automated by one CA OPS/MVS per Linux Connector component. This avoids duplicate automation of events.

The following Linux Connector unsolicited message events are available through API rules:

Z/VM Messages

Description

LXMSG001I z/VM-node message-type user ID message-text

Example

```
LXMSG001I ZVM002 MSG POLLGEN NMVM0001 00:34:49 Hello
```

Z/VM Events

Description

LXEVT001I z/VM-node user ID event-type

Example

```
LXEVT001I ZVM002 LINUX113 RUNNABLESTATEENABLED
```

Linux Syslog-ng Messages

Description

LXLOG001I Linux-name z/VM-host facility severity message-text

Example

LXLOG001I LINUX113 ZVM002 user notice logger: Test Message

OPS/REXX Rule Variables For Linux Connector Events

These variables are available for all unsolicited VM and Linux message events.

Note: When a variable representing event data is not applicable for that event, the variable-length is zero.

API.APPLICATION

Description

The name of the application providing the event.

Data Type

Character, read-only, up to eight characters.

Fixed Value

LXC

API.HOSTNAME

Description

The name of the VM or Linux system that issued the unsolicited message.

Data Type

Character, read-only, 1-16 characters.

Example

LINUX113

API.ID

Description

The first word of the unsolicited message text.

Data Type

Character, read-only, 1-10 characters.

Example

LXLOG0001

API.LEVEL

Description

This value is set to the CA OPS/MVS subsystem name that created the API event.

Data Type

Character, read-only, four characters.

Example

OPSS

API.SYSTYPE

Description

The system type that generated the message.

Data Type

Character, read-only, character

Values

VM

VM or Linux guest computers

LPAR

A Linux guest system that does not support the VMCP command.

INTEL

A Linux system running on an Intel platform

OTHER

A Linux system running on a non-Intel platform

API.TEXT

Description

The complete normalized message text that was passed from the Linux Connector component. The first 128 characters appear in OPSLOG when the BROWSELXC parameter is set to YES.

Data Type

Character, read-only, 1-4096 characters.

Example

```
LXLOG001I LINUX113 ZVM002 user notice logger: Test Message
```

API.VERSION

Description

This value is set to the CA OPS/MVS product version code.

Data Type

Character, read-only, eight characters.

Example

12.00.00

API.VMNODE

Description

The name of the VM system that issued a VM message or the Linux guest VM system name.

Data Type

Character, read-only, 1-8 characters.

Example

ZVM002

CA Product API Event Types

For information on other CA products that generate API events, please see the documentation for the desired product.

Automatic Restart Management Rules

An Automatic Restart Management (ARM) rule triggers when the Automatic Restart Manager component of z/OS attempts to restart an ARM-registered job or started task after an unexpected termination.

The restart may occur on the same system or on another system in the sysplex if the termination was due to a complete system failure.

ARM rules provide the ability to:

- Intercept ARM restart events before the restart occurs.
- Override JCL or start text by setting the modifiable ARM event variables or by terminating the restart event.

If System State Manager is being used to manage resources that are using ARM for restart, ARM rules provide the ability to either:

- Suppress the ARM restart if System State Manager has already acted to restart the resource.
- Allow ARM to proceed and prevent System State Manager from duplicating the restart action.

Installation Requirements for ARM Rules

To install ARM rules, set the parameters INITARM and ARMRULES to YES. For more information, see the *Parameter Reference*.

)ARM—Event Specifier of ARM Rules

The following is the format for coding the ARM-event definition section:

`)ARM` *`elementnamespec`*

elementnamespec

Specifies the element name. Follow these guidelines when specifying the character string:

- Specify 1 to 16 characters of the registered ARM element name.
- The string cannot contain embedded blank spaces.
- You can use the wildcard (*) character. For example:
  - CICS* matches CICSA, CICSABC, CICS123, and any other element name containing a CICS prefix.
  - CICS*05 matches CICSD05, CICS205, CICS1105, and so on.
  - *05 matches any element name ending with 05.
  - * alone matches all element names.
- Lowercase letters are acceptable, but the AOF converts them to uppercase for event testing.

Initialization, Processing, and Termination Sections of ARM Rules

ARM rules follow the general guidelines for coding the initialization, processing, and termination sections and the various AOF tools that you can use.

More information:

AOF Rule Structure (see page 21)
AOF Rule Tools (see page 29)

RETURN Statements in the)PROC Section of an ARM Rule

The OPS/REXX RETURN statement specifies whether the ARM restart for the element should be prevented or allowed to continue. This statement may specify the following values:

NORMAL

Continue the element restart in accordance with the current value of the ARM.RESTARTTYPE variable. Other modifiable event variables may have also changed values.

SUPPRESS

The value of the ARM.RESTARTTYPE variable is set to 1, which tells ARM that a restart should not be performed for this element.

Default: RETURN NORMAL

The return values listed here are character *constants* rather than keywords. An unrecognized return value (for example, a misspelled value) defaults to a value of NORMAL.

Execution Considerations for ARM Rules

The processing section of a rule that responds to an ARM event executes in the z/OS XCFAS address space. Therefore, any type of logic that could possibly suspend the processing of an ARM rule should be performed by scheduling an OPS/REXX program to run in a CA OPS/MVS OSF TSO, TSL, or TSP server.

More information:

Code and Debug AOF Rules (see page 59)

OPS/REXX Host Environments in the)PROC Section of an ARM Rule

The)PROC section of an ARM rule has the following host environments with the following ARM rule characteristics. Specify the AOFDEFAULTADDRESS parameter for the default host environment for ARM rules.

ADDRESS AOF

Sent to CA OPS/MVS. Does not wait. Output is not returned.

ADDRESS AP

Sent to MSF and then forwarded to the CA Automation Point system. Does not wait. Output is not returned.

ADDRESS EPI

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS HWS

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS ISPEXEC

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS LXCON

The VM and Linux command requests sent to USS server for execution. Does not wait. Output is not returned. The List request runs inline and returns VM and Linux system data in stem variables.

ADDRESS MESSAGE

Sent as a WTO. The AOFDEST parameter specifies the destination.

ADDRESS MQ

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS NETMAN

Sent to a CA OPS/MVS internal CA Netman request queue to issue MGPT commands. Does not wait. Output is not returned.

ADDRESS NETMASTR

Sent to CA NetMaster NM for SNA on the local system. Does not wait. Output is not returned.

ADDRESS OPER

Sent to target console as specified through the OCCONSOLENAME parameters when no CONNAME operands are present. Output is not returned. Schedule an OPS/REXX program in a server when a command output interrogation is needed.

ADDRESS OPSCTL

Sent to a specified facility. If the facility is ECF or OSF, does not wait. If the facility is MSF, slight wait occurs. The external data queue returns the output.

Note: If the command is MSF LIST, no wait occurs.

ADDRESS OPSDYNAM

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS OSF

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSO servers.

ADDRESS OSFTSL

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSL servers.

ADDRESS OSFTSP

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSP servers.

ADDRESS SQL

Does not wait. Processed synchronously for requests that can be satisfied on the local system. The output is returned in a stem variable. Error messages, if any, are returned to an external data queue.

ADDRESS SOF

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS SYSVIEWE

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS TSO

Sent to an OSF TSO server. Does not wait. Output is not returned.

ADDRESS USS

Sent to a USS server. Does not wait. Output is not returned. Schedule an OPS/REXX program in a server if a command output interrogation is needed.

ADDRESS WTO

Does not wait. Output is sent to specified (or default) destination. When attempting a WTOR, host command is sent to a TSO server for execution. The response is returned to the server. Schedule an OPS/REXX program in a server if a WTOR response interrogation is needed.

More information:

Code and Debug AOF Rules (see page 59)

AOF Variables Available in an ARM Rule

You can use all AOF variable types in ARM rules. The following unique AOF event variables are available in the)PROC section of an ARM rule. It also lists the corresponding OPSLOG display field that you can manually interrogate as an aid in debugging or implementing rule logic.

ARM.CLONEID

Specifies the z/OS sysplex clone ID of the system on which the job originally registered with ARM.

Data Type: 2-byte character, read-only

Sample Value: 02

ARM.COLOR

Specifies the color that the ARM event message text will use in OPSLOG Browse.

Data Type: 1-byte binary (unprintable), read/write

Sample Value: '00'X

Notes:

- Use the OPSCOLOR function of OPS/REXX to set the ARM.COLOR variable.
- If multiple rules set ARM.COLOR for a single ARM message, CA OPS/MVS uses only the last value. To trace the color set by each rule, set the RULETRACE parameter to ON.
- For a description of the RULETRACE parameter, see the *Parameter Reference*.

OPSLOG Browse Column: COLOR

ARM.ELEMENT

Specifies the ARM element name of the job or started task that is being restarted. ARM element names are unique across a sysplex. The first 10 characters of the element name are also in the OPSLOG MSGID field.

Data Type: 1 to 16-byte character, read-only

Sample Value: OPSMVSSYSA

OPSLOG Browse Column: Text is always displayed

ARM.ELEMTYPE

Specifies the ARM element type, which is used in the ARM policy to define restart characteristics for groups of related resources so that individual definitions are not required for every element.

Data Type: 0 to 8 characters, read-only

Sample Value: OPSMVS

OPSLOG Browse Column: TERMNAME

ARM.EVENTCODE

Specifies a code value that represents the reason that ARM restarted the job.

Data Type: 2-byte binary (unprintable), read-only

Possible Values:

- 1 (the restart was caused by the termination of the element)
- 2 (the restart was caused by the termination of the system)

Sample Value: 1

ARM.FROMSYS

Specifies the z/OS system name of the system on which the job was last executed.

Data Type: 1- to 8-byte character, read-only

Sample Value: SYS03

OPSLOG Browse Column: DSPNAME

ARM.HOMESYS

Specifies the z/OS system name of the system on which the job originally registered with ARM.

Data Type: 1- to 8-byte character, read-only

Sample Value: SYS02

ARM.JCLDSN

Specifies the name of the data set that contains the JCL that will be submitted to restart the job.

Data Type: 0 to 44 characters, read/write

Sample Value: USER.CNTL

Note: If this value is changed, the ARM.RESTARTTYPE variable must be set to 4. If the JCL requires system variable substitution, variable values from the home system will be used.

ARM.JCLMEM

Specifies the PDS member name of the data set name specified in ARM.JCLDSN, which contains the JCL that will be used to restart the job. If the data set is not a PDS, the value of this variable will be null.

Data Type: 0 to 8 characters, read/write

Sample Value: USERJOB

ARM.JOBNAME

Specifies the job name of the ARM element that is being restarted.

Data Type: 1- to 8-byte character, read-only

Sample Value: OPSMAIN

OPSLOG Browse Column: JOBNAME

ARM.PERSISTJCL

Specifies a value indicating whether persistent JCL is available for the restart of the job.

Data Type: Integer, read-only

Possible Values: 0 (persistent JCL is not available) or 1 (persistent JCL is available)

Sample Value: 0

ARM.POLICYSTART

Specifies a value indicating whether the start command text for a restart that is being performed using a start command (ARM.RESTARTTYPE=2) is from the ARM policy definition or a system-entered start command (persistent text).

Data Type: Integer, read-only

Possible Values:

- 0-start text is persistent
- 1-start text is from ARM policy

Sample Value: 1

ARM.RESTARTTYPE

Specifies a value indicating the type of restart to be performed.

Data Type: 1-byte binary, read/write

Possible Values:

- 1-Do not restart this job (same as RETURN 'SUPPRESS')
- 2-Restart this started task using the start text value in ARM.STARTTEXT
- 3-Restart this job using the persistent JCL
- 4-Restart this job using the override JCL specified in ARM.JCLDSN and ARM.JCLMEM

Sample Value: 1

ARM.STARTTEXT

Specifies the text of the z/OS start command that will be issued to restart the job. This may be the original start command text, command text from the ARM policy couple data set, or an override value from an AOF ARM rule.

Data Type: 0 to 126 characters, read/write

Sample Value: START USERJOB, PARM='RESTART'

Note: If this value is changed, the variable ARM.RESTARTTYPE must be set to 2. If the text requires system variable substitution, variable values from the home system will be used.

ARM.TEXT

Specifies the message text generated for this ARM AOF event. The text is constant except for the last word, which is the ARM element name.

Data Type: Character, read-only

Sample Value: MVS ARM RESTART OF OPSMVSSYSA

OPSLOG Browse Column: Text is always displayed

ARM.TOSYS

Specifies the current z/OS system name on which the job is being restarted

Data Type: 1- to 8-byte character, read-only

Sample Value: SYS03

OPSLOG Browse Column: SYSNAME

ARM.USER

Contains an 8-byte variable providing communication between rules that execute for the same ARM event. The variable can contain any installation data that these rules need, and it can store a character string displayable through OPSLOG Browse.

Data Type: User-defined, read/write

Notes:

- Before AOF processing, this variable is initialized to binary zeros. It is then passed to each rule that executes for the same ARM event; each rule can look at or change the variable contents before passing the variable to the next rule for the ARM event.
- The primary purpose for the USER variable is to provide a method to pass a small amount of data between the rules. This data may be binary or mixed case. The USER field may also be used for filtering in the OPSLOG. However, USER data used for OPSLOG filtering must be uppercase and displayable.

OPSLOG Browse Column: USER

More information:

AOF Rule Tools (see page 29)

Debug an ARM Rule

The following example assumes:

- A System State Manager resource table for all CICS regions exists on all systems in a sysplex.
- The resource table entries for CICS regions that are not running on each system have a mode of INACTIVE.
- If one of the systems fail, ARM restarts the CICS region from the failing system on one of the other systems.
- The System State Manager on the new system allows ARM to restart the transferred CICS region; however, System State Manager will handle any subsequent restarts.
- An action entry table for CURRENT=UP DESIRED=UPARM will reset the desired state to UP.

Example: Debug an ARM Rule

```
)ARM CICS*
)PROC
  if arm.fromsys <> arm.tosys then  /* CICS region transfer */
    do
      address SQL "Update CICS_TABLE Set CURRENT_STATE='DOWN',",
        "DESIRED_STATE='UPARM', MODE='ACTIVE',",
        "Where JOBNAME='"arm.jobname"' And ",
        "MODE='INACTIVE'"
      return "NORMAL"                 /* Let ARM do 1st start */
  End
  /* Local system restart */
  address SQL "Select NAME From CICS_TABLE",
     "Where JOBNAME='"arm.jobname"' And",
     "MODE='ACTIVE'"
  if rc=0 & sqlcode=0 then           /* Found in SSM table    */
    return "SUPPRESS"                /* SSM will do restart   */
  return "NORMAL"                    /* Not under SSM control */
  )END
```

Note: For debugging techniques that you can use with all AOF rules, see the chapter "Code and Debug AOF Rules (see page 59)."

Command Rules

An AOF command (CMD) rule is triggered by commands that are broadcast through the Subsystem Interface (SSI), such as z/OS, JES2, and JES3 commands.

CMD rules let you perform the following tasks:

- Disallow commands
- Modify command operands
- Replace commands
- Create new commands

In addition, CMD rules can be used to intercept pseudo (user created) commands, allowing you to trigger some automated process from anywhere in your environment that allows you to issue system commands through the SSI.

Installation Requirements for CMD Rules

Set the parameter SSICMD to YES to allow CA OPS/MVS to process commands before any other subsystems. For details, see the *Parameter Reference*.

The CA OPS/MVS optional IMS Operator Facility is required to process IMS commands that are issued by an IMS MTO.

Note: For more information on the IOF facility, see the *Administration Guide* and *User Guide*.

)CMD—Event Specifier of CMD Rules

The CMD-event definition section has the following format:

```
)CMD cmdverbspec
```

Adhere to the following guidelines when specifying the character string for the *cmdverbspec:*

- Specify 1 to 10 characters.
- The string cannot contain embedded blank spaces.

- z/OS Command Considerations:
  - Specify the full command verb or a mask of the command verb when attempting to trigger on z/OS commands.

    For example, when writing a command rule that triggers whenever a z/OS DISPLAY command is issued, specify)CMD DISPLAY rather than)CMD D, or specify)CMD MODIFY rather than)CMD F to execute on z/OS modify commands. Although you must specify the full z/OS command verb in the event definition section of the rule, a command rule recognizes a command event if an operator issues a short form (alias) of the command. Additional logic can be implemented into the)PROC section of the rule to interrogate the CMD.TEXT event variable to see the exact command that was entered.
  - Z/OS may reissue some commands internally if they do not originate from the CONSOLE address space, that is, if a program issues the commands rather than a z/OS console.

    z/OS reissues such commands so that the processing occurs in the CONSOLE address space, thus causing a CMD rule to possibly execute twice. z/OS reissues DISPLAY ACTIVE commands and any other command that creates paged-frame display output on a z/OS console.
- JES2 Command Considerations:

  Use these guidelines when you are writing rules that respond to JES2 commands:
  - When you attempt to trigger on JES2 commands, specify the JES2 command character followed by the first letter of the JES2 command. For example, if $ is the JES2 command character and you want to trigger on the $TI initiator command, then specify)CMD $T.
  - You can add logic to the)PROC section of the rule to interrogate the CMD.TEXT event variable and see the JES2 command that was entered.
  - For both JES2 and z/OS, you can use a delimiter character to enter more than one command on a single line. JES2 uses a semicolon to delimit multiple commands and z/OS uses the character specified by the CMDDELIM parameter in the CONSOL*xx* member of the logical PARMLIB concatenation.

The CMD rule specifier needed to trap stacked JES2 commands is impacted by the type of issuing console (extended, MCS, SMCS, and so on). It is also impacted by the setting of the z/OS CMDDELIM parameter.

Use these guidelines when you are creating CMD rules to process JES2 stacked commands:

- If CMDDELIM is set to a semicolon (;), and the issuing console is a MCS or SMCS console, then stacked JES2 commands can only be issued in the form of: $cmd1;$cmd2;$cmd3. For example, $PI1;$TI1,C=X;$SI1.

  Each stacked JES2 command will trigger an associating unique CMD rule, or a wild card catch all JES2 CMD rule will be executed for each stacked command. In this example, a)CMD $* rule would execute three times (once for each command), a)CMD $P rule would execute once, a)CMD $T would execute once, and a)CMD $S rule would execute once.

- If CMDDELIM is not set (there is no z/OS command stacking), or CMDDELIM is set to some value other than a semicolon (;), or CMDDELIM is set to a semi-colon (;) and the issuing console is not a MCS or SMCS console, then stacked JES2 commands can be issued in the form of $cmd1;cmd2;cmd3. For example, $PI1;TI1,C=X;SI1.

  The entire list of commands stacked together is treated as one command. This command is processed by either one CMD rule that processes all JES2 commands or a specific CMD rule that processes the first command within the list of stacked commands.

  For example, a)CMD $* rule would execute once for the complete stacked command, or a)CMD $P rule would execute once for the complete stacked command. In both cases, additional rule logic to interrogate the value of the *cmd.text* environmental variable should be coded to process each command accordingly.

If you are attempting to process and automate various JES2 commands, it may be suitable to create a catch all JES2 rule that rejects command stacking. To implement this type of control logic, see the sample rule JESSTACK in the ops.sample data set.

- JES3 Command Considerations:

  When writing rules that respond to JES3 commands, begin the *cmdverbspec* event identifier string with the first character in the JESCHAR parameter string. Only the first character in JESCHAR is meaningful in a JES3 environment. When CA OPS/MVS processes a command on a JES3 system, it attempts to match the command prefix with one of the JES3 system or sysplex prefixes. If a match is found, CA OPS/MVS creates a common command verb (see CMD.VERB), regardless of which command prefix was used or whether the command was abbreviated. The purpose of this is to make sure that you have to write only a single command rule for each JES3 command, regardless of how it was issued.

The command verb (and CMD.VERB) consists of the real command verb with the original prefix stripped off and replaced by the first character in JESCHAR. This does not affect the command text itself, only the verb used to execute the rules. For example, assume you have the following:

```
JES3 SYN prefix =  8
JES3 PLEXSYN prefix = %%
CA OPS/MVS JESCHAR parameter = *
```

When you issue the JES3 command 8I S, a rule with a command verb (CMD.VERB) of *INQUIRY is executed. CMD.TEXT will not be changed and remains 8I S (unless CMD.TEXT is changed by the rule itself). The same rule is executed if you issue the command %%I S; CMD.VERB will be the same as in the prior case but the CMD.TEXT will be %%I S.

The use of multiple system or sysplex command prefixes is fully supported by CA OPS/MVS, whether these prefixes are single- or multiple-character.

If your CMD rule needs to make decisions based on the original JES3 command prefix, then use the following three environmental variables:

CMD.JES3PREFIX

Indicates the original command prefix

CMD.JES3SYN

Set to 1 if the original JES3 command prefix is a system scope prefix (SYN)

CMD.JES3PLEXSYN

Set to 1 if the original JES3 command prefix is defined as a sysplex scope prefix (PLEXSYN)

If you set JESCHAR to the same value on all your systems, one rule should be able to work on all your JES3 systems, regardless of what the SYN and PLEXSYN values are on those systems.

- If the first character in JESCHAR is an * (which is also the wildcard character), you may need to perform additional checking in the PROC section of the rule. Because the *cmdverbspec* string in the event definition section begins with the wildcard (*) character, a rule triggered by an INQUIRY command is also triggered by NOINQUIRY (not a JES3 command) and INQUIRY (a possible z/OS command). You can solve the problem by coding the processing section of your rule as shown in this example:

  ```
  )CMD *INQUIRY
  )PROC
   if CMD.VERB ¬= '*INQUIRY' then
   return 'NOACTION'
  ```

- A CMD rule can change *START PRTR1 to *START PRTR2 but not to VARY 2F0,OFFLINE.

- Subsystem Command Character Considerations other than JES2 Commands:

  Specify the command character followed by the wildcard character (*) to execute on commands issued using a product-defined command character. For example, assuming the character / is the command character of a particular subsystem, code a specification of)CMD /* to trigger on commands issued using its command character. Additional logic can be implemented in the)PROC section of the rule to interrogate the CMD.TEXT event variable to see the exact command that was entered.

- You can use the wildcard character (*) where applicable. For example,
  - ST* matches z/OS START and STOP commands or any pseudo command that beings with ST.
  - * matches all command events on the system.
- Lowercase letters are acceptable, but the AOF converts them to uppercase for event testing.

z/OS Command Guidelines

Specify the full command verb or a mask of the command verb when attempting to trigger on z/OS commands.

Example

This command rule triggers whenever a z/OS Display command is issued.

```
)CMD DISPLAY rather than )CMD D
```

This command executes on z/OS modify commands.

```
)CMD MODIFY rather than )CMD F
```

Specify the full z/OS command verb which is required in the event definition section of the rule. A command rule recognizes a command event when an operator issues a short form (alias) of the command.

Logic can be implemented into the)PROC section of the rule to interrogate the CMD.TEXT event variable to see the exact command that was entered.

Z/OS can reissue some commands internally when they do not originate from the CONSOLE address space. That is, if a program issues the commands rather than a z/OS console.

z/OS reissues such commands so that the processing occurs in the CONSOLE address space, thus causing a CMD rule to execute possibly twice. z/OS reissues DISPLAY ACTIVE commands and any other command that creates paged-frame display output on a z/OS console.

JES2 Command Guidelines

Use these guidelines when writing rules that respond to JES2 commands:

- To trigger on JES2 commands, specify the JES2 command character followed by the first letter of the JES2 command.

  Example

  If $ is the JES2 command character and you want to trigger on the $TI initiator command, then specify:

  ```
  )CMD $T.
  ```

- Add logic to the)PROC section of the rule to interrogate the CMD.TEXT event variable and see the JES2 command that was entered.

- For both JES2 and z/OS, use a delimiter character to enter more than one command on a single line. JES2 uses a semicolon to delimit multiple commands. z/OS uses the character that is specified by the CMDDELIM parameter in the CONSOLxx member of the logical PARMLIB concatenation.

  The CMD rule specifier traps stacked JES2 commands with an impact from the:

  - Type of issuing console. For example, extended, MCS, and SMCS.
  - Setting of the z/OS CMDDELIM parameter.

JES2 Stacked Command Guidelines

Use these guidelines when you are creating CMD rules to process JES2 stacked commands:

- When CMDDELIM is set to a semicolon (;), and the issuing console is an MCS or SMCS console. Then stacked JES2 commands can only be issued in the form of: $cmd1;$cmd2;$cmd3. For example, $PI1;$TI1,C=X;$SI1.

  Each stacked JES2 command triggers an associating unique CMD rule. Or a wildcard catch all JES2 CMD rule is executed for each stacked command.

  Example

  A)CMD $* rule would execute three times (once for each command).

  A)CMD $P rule would execute once.

  A)CMD $T would execute once.

  A)CMD $S rule would execute once.

- When CMDDELIM is not set there is not any z/OS command stacking.

  When CMDDELIM is set:

  To some value other than a semicolon (;). Or is set to a semi-colon (;) and the issuing console is not an MCS or SMCS console. Then the stacked JES2 commands can be issued in the form of:

  ```
  $cmd1;cmd2;cmd3.
  ```

  For example:

  ```
  $PI1;TI1,C=X;SI1.
  ```

  The entire list of commands that is stacked together is treated as one command. This command processes by either one CMD rule that processes all JES2 commands. Or a specific CMD rule that processes the first command within the list of stacked commands.

  Example

  A)CMD $* rule would execute once for the complete stacked command.

  A)CMD $P rule would execute once for the complete stacked command.

  In both cases, code additional rule logic to interrogate the value of the cmd.text environmental variable.

  To process and automate various JES2 commands, create a catch all JES2 rule that rejects command stacking. To implement this type of control logic, see the sample rule JESSTACK in the CCLXRULS data set.

Respond Rules For JES3 Command Guidelines

When writing rules that respond to JES3 commands, do:

Begin the *cmdverbspec* event identifier string with the first character in the JESCHAR parameter string. Only the first character in JESCHAR is meaningful in a JES3 environment.

CA OPS/MVS attempts to match the command prefix with one of the JES3 system or sysplex prefixes, while processes a command.

When CA OPS/MVS, finds a match it creates a common command verb (CMD.VERB). Regardless of which command prefix was used or whether the command was abbreviated.

Use these guidelines, to write a single command rule only for each JES3 command, regardless of how it was issued.

The command verb (and CMD.VERB) consists of the real command verb with the original prefix stripped off and replaced by the first character in JESCHAR. This first character does not affect the command text itself, only the verb that is used to execute the rules.

Example

```
JES3 SYN prefix =  8
JES3 PLEXSYN prefix = %%
CA OPS/MVS JESCHAR parameter = *
```

When issuing the JES3 command 8I S, a rule with a command verb (CMD.VERB) of *INQUIRY is executed. CMD.TEXT does not change and remains 8I S unless the rule itself changes CMD.TEXT. The same rule is executed when the command %%I S. CMD.VERB is the same as in the prior case but the CMD.TEXT is %%I S.

CA OPS/MVS supports the use of multiple system or sysplex command prefixes, whether these prefixes are single- or multiple-character.

When decision making using a CMD rule-based on the original JES3 command prefix, use the following three environmental variables:

CMD.JES3PREFIX

Indicates the original command prefix

CMD.JES3SYN

Set to 1 if the original JES3 command prefix is a system scope prefix (SYN)

CMD.JES3PLEXSYN

Set to 1 if the original JES3 command prefix is defined as a sysplex scope prefix (PLEXSYN)

When the JESCHAR is set to the same value on all your systems. One rule works on all JES3 systems, regardless of the SYN and PLEXSYN values are on those systems.

- When the first character in JESCHAR is a wildcard (*) character. Perform additional checking in the PROC section of the rule.
- The cmdverbspec string in the event definition section begins with the wildcard (*) character. A rule that is triggered by an INQUIRY command is also triggered by NOINQUIRY (not a JES3 command) and INQUIRY (a possible z/OS command).
- This issue is solved by coding the processing section of the rule as follows.

```
)CMD *INQUIRY
)PROC
 if CMD.VERB ¬= '*INQUIRY' then
 return 'NOACTION'
```

- A CMD rule can change:

  `*START PRTR1 to *START PRTR2`

  But not to:

  `VARY 2F0,OFFLINE`

Subsystem Command Character Guidelines

When using subsystem command character other than JES2 Commands, do:

Specify the command character followed by the wildcard (*) character to execute on commands issued using a product-defined command character.

Example

Assume the character / is the command character of a particular subsystem. Code a specification of)CMD /* to trigger on commands that are issued using its command character. Logic can be implemented in the)PROC section of the rule to interrogate the CMD.TEXT event variable to see the exact command that was entered.

- Use the wildcard character (*) where applicable.

  Example

  - ST* matches z/OS START and STOP commands or any pseudo command that beings with ST.
  - * matches all command events on the system.
- Lowercase letters are acceptable, but the AOF converts them to uppercase for event testing.

Initialization, Processing, and Termination Sections of CMD Rules

The general guidelines for coding the initialization, processing, and termination sections and the various AOF tools that you can use, apply to CMD rules.

More information:

AOF Rule Structure (see page 21)
AOF Rule Tools (see page 29)

RETURN Statements in the)PROC Section of a CMD Rule

The OPS/REXX RETURN statement specifies the final disposition of a z/OS command.

The following are valid values for a RETURN statement in the processing section of a command rule:

NOACTION

Allows z/OS to process a command (after CMD rule processing, if any)

ACCEPT

Prevents z/OS from processing a command (after CMD rule processing, if any)

REJECT

Causes z/OS to reject a command as invalid, resulting in message IEE707I *cmd* NOT EXECUTED (after CMD rule processing, if any)

Default: RETURN 'NOACTION'

The return values listed here are character *constants* rather than keywords. An unrecognized return value (for example, a misspelled value) defaults to a value of NOACTION.

Other RETURN Statement Considerations

In a command rule, the return value can affect command processing as follows:

- If multiple rules respond to a single command event, the AOF uses the highest-precedence return value; the order of precedence is:
  - REJECT (highest precedence)
  - ACCEPT
  - NOACTION (lowest precedence)
- To effectively intercept a command before any other subsystem processes it, perform the following steps:
  1. Set the SSICMD parameter to YES.
  2. Within the CMD rule logic, set the cmd.text environmental variable to null (cmd.text='') before exiting the rule with a RETURN 'ACCEPT.' See the AOF rule sample member JES2$TJ for an example of intercepting and processing a JES2 command.

  Other types of commands (such as JES, DB2, NetView, BDT) are not affected by the RETURN statement. In these cases, you must modify CMD.TEXT to prevent these subsystems from processing the command.

Execution Considerations for CMD Rules

The processing section of a rule that responds to a command event executes in the address space from which the command originated, which is usually the CONSOLE address space. Therefore, any type of logic that could possibly suspend the processing of a CMD rule should be performed by scheduling an OPS/REXX program to run in a CA OPS/MVS OSF TSO, TSL, or TSP server.

More information:

Code and Debug AOF Rules (see page 59)

OPS/REXX Host Environments in the)PROC Section of a CMD Rule

The)PROC section of a CMD rule has the following host environments with the following CMD rule characteristics. The AOFDEFAULTADDRESS parameter specifies the default host environment for CMD rules.

ADDRESS AOF

Sent to CA OPS/MVS. Does not wait. Output is not returned.

ADDRESS AP

Sent to MSF and then forwarded to the CA Automation Point system. Does not wait. Output is not returned.

ADDRESS EPI

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS HWS

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS ISPEXEC

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS LXCON

The VM and Linux command requests sent to USS server for execution. Does not wait. Output is not returned. The List request runs inline and returns VM and Linux system data in stem variables.

ADDRESS MESSAGE

Sent as a WTO to the issuing console.

ADDRESS MQ

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS NETMAN

Sent to a CA OPS/MVS internal CA Netman request queue to issue MGPT commands. Does not wait. Output is not returned.

ADDRESS NETMASTR

Sent to CA NetMaster NM for SNA on the local system. Does not wait. Output is not returned.

ADDRESS OPER

Sent to target console as specified through the OCCONSOLENAME parameters when no CONNAME operands are present. Output is not returned. Schedule an OPS/REXX program in a server when a command output interrogation is needed.

ADDRESS OPSCTL

Sent to a specified facility. If the facility is ECF or OSF, does not wait. If the facility is MSF, slight wait occurs. The external data queue returns the output.

Note: If the command is MSF LIST, no wait occurs.

ADDRESS OPSDYNAM

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS OSF

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSO servers.

ADDRESS OSFTSL

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSL servers.

ADDRESS OSFTSP

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSP servers.

ADDRESS SOF

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS SQL

Does not wait. Processed synchronously for requests that can be satisfied on the local system. Output is returned in stem variable. Error messages, if any, are returned to an external data queue.

ADDRESS SYSVIEWE

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS TSO

Sent to an OSF TSO server. Does not wait. Output is not returned.

ADDRESS USS

Sent to a USS server. Does not wait. Output is not returned. Schedule an OPS/REXX program in a server if the command input interrogation is needed.

ADDRESS WTO

Does not wait. Output is sent to specified (or default) destination. When attempting a WTOR, host command is sent to a TSO server for execution. The response is returned to the server. Schedule an OPS/REXX program in a server if a WTOR response interrogation is needed.

More information:

Code and Debug AOF Rules (see page 59)

AOF Variables Available in a CMD Rule

You can use all AOF variable types in CMD rules. The following unique AOF event variables are available in the)PROC section of a CMD rule. Also listed are the corresponding OPSLOG display field that you can manually interrogate as an aid in debugging or implementing rule logic.

CMD.AOFCMD

A value indicating whether CA OPS/MVS issued the current command from within an AOF rule

Data Type: Integer, read-only

Possible Values:

- 0-The command was not issued by a CA OPS/MVS AOF rule
- 1-The command was issued by a CA OPS/MVS AOF rule

Sample Value: 1

Notes:

- When the value of the CMD.AOFCMD variable is 1, the value of the CMD.PRODCMD variable is also 1.
- When the value of the CMD.AOFCMD variable is 1, it is most likely that the command was issued through an ADDRESS OPER host command in an AOF rule.

OPSLOG Browse Column: Second bit of OPSFLAGS when the EVENT column indicates a CMD event

CMD.CONSNAME

The name of the console from which the command was issued **Data Type:** Character, read-only

Sample Value: MASTSYSA

Note: For commands originating in the subsystem interface, CMD.CONSNAME contains the console name.

OPSLOG Browse Column: CONSNAME

CMD.IMSID

The ID of the IMS control region that issued this command, or NONE for non-IMS commands. The CMD.IMSID variable is available only if the IOF is licensed, installed, and active at your site (that is, the INITIMS parameter must be set to YES).

Data Type: Character, read-only

Sample Value: IMSA

OPSLOG Browse Column: IMSID

CMD.JES3PLEXSYN

Indicates whether the command prefix is a JES3 sysplex scope prefix. Set to 1 if the original JES3 command prefix is defined as a sysplex scope prefix (PLEXSYN).

Data Type: Character, read-only

Possible Values:

- 0-The prefix is not a JES3 sysplex scope prefix
- 1-The prefix is a JES3 sysplex scope prefix

Sample Value: 1

Note: This variable is only meaningful for JES3 commands in a JES3 environment.

OPSLOG Browse Column: Eighth bit of OPSFLAGS when the event column indicates a CMD event, and a JES command prefix was used.

CMD.JES3PREFIX

The original JES3 prefix used on the JES3 command

Data Type: Character, read-only

Possible Values: Any valid JES3 system or sysplex command prefix, which can be from 1 to 8 characters in length. With the special exception of the character 8, JES3 command prefixes cannot start with a numeric character.

Sample Value: *

Note: This variable is only meaningful for JES3 commands in a JES3 environment.

OPSLOG Browse Column: AUTOTOKN when the event column indicates a command event, and a JES command prefix is used.

CMD.JES3SYN

Indicates whether the command prefix is a JES3 system scope prefix. Set to 1 if the original JES3 command prefix is a system scope prefix (SYN).

Data Type: Character, read-only

Possible Values:

- 0-The prefix is not a JES3 system scope prefix
- 1-The prefix is a JES3 system scope prefix

Sample Value: 0

Note: This variable is only meaningful for JES3 commands in a JES3 environment.

OPSLOG Browse Column: Seventh bit of OPSFLAGS when the event column indicates a CMD event, and a JES command prefix was used.

CMD.JOBNAME

The name of the job or the TSO user who issued the command

Data Type: Character, read-only

OPSLOG Browse Column: JOBNAME

CMD.MSFID

The MSF system name of the copy of CA OPS/MVS that issued the command

Data Type: Character, read-only

Sample Value: OPSP

Note: For all commands issued from sources other than remote copies of CA OPS/MVS, the value of CMD.MSFID is the MSF ID of the local copy of CA OPS/MVS.

OPSLOG Browse Column: MSFID

CMD.ORIGINSYS

The system name from which the command originated. This variable enables you to identify the origin of commands routed to this system through the ROUTE command. CMD.ORIGINSYS can only return the correct information when the SSICMD product parameter is set to YES. When SSICMD is set to NO, this variable and the SYSNAME column in OPSLOG are set to UNKNOWN.

Data Type: Character, read-only.

Sample Value: SYSA

OPSLOG Browse Column: SYSNAME

CMD.OTEXT

The original text of the command, unmodified by subsequent rule processing

Data Type: Character, read-only

CMD.PRODCMD

A value indicating whether CA OPS/MVS issued the current command

Data Type: Integer, read-only

Possible Values: 0 (if CA OPS/MVS did not issue the command) or 1 (if CA OPS/MVS issued the command)

Sample Value: 1

Notes:

- When the value of the CMD.PRODCMD variable is 1, the command was issued through either an ADDRESS OPER host command or the OPSCMD command processor.
- The value of the CMD.PRODCMD variable is 1 only if the command was issued through a z/OS service. Thus, for JES3 commands this value is always 0.

OPSLOG Browse Column: First bit of OPSFLAGS when the EVENT column indicates a CMD event

CMD.SSMCMD

A value indicating whether System State Manager issued the current command

Data Type: Integer, read-only

Possible Values:

- 0-The command was not issued by System State Manager
- 1-The command was issued by System State Manager

Sample Value: 1

Notes:

- When the value of the CMD.SSMCMD variable is 1, the value of the CMD.PRODCMD variable is also 1.
- When the value of the CMD.SSMCMD variable is 1 and the value of SSM.AOFCMD is 1, the command was issued from a System Manager State Manager REQ rule (EVRULE or RULE action keywords).
- The value of the CMD.SSMCMD variable will be 0 if the command was issued from an OSF TSO server as a result of an asynchronous System State Manager action (TSOCMD, CLIST or REXX action keywords).

OPSLOG Browse Column: Third bit of OPSFLAGS when the EVENT column indicates a CMD event

CMD.TERMNAME

The JES3 console name that submitted the current command, or the string NONE if the current JES3 command is not associated with a JES3 console

Notes:

- If the command was issued from a TSO address space, the terminal name associated with the logged-on user is available in CMD.TERMNAME.
- If the command was issued from CA Remote Console, the CMD.TERMNAME environmental variable contains the name of the terminal with which the RCS user logged on.

Data Type: Character, read-only

OPSLOG Browse Column: TERMNAME

CMD.TEXT

The command text as it will execute, taken from:

- The MGCRTEXT field of the MGCR parameter list or the MGCETXT field of the MGCRE parameter list (z/OS or JES3 commands)
- An internal IMS buffer (IMS commands)

Data Type: Character, read/write

Sample Value: 'D TS,L'

Notes:

- You cannot change IMS command length, but z/OS commands can be lengthened or shortened.
- z/OS and IMS always process modified command text.
- If subsystems such as JES2 and JES3, DB2, and NetView receive the command text before CA OPS/MVS, the changes made by AOF CMD rules are ignored. However, these changes take effect if CA OPS/MVS gets the command first. To make sure that CA OPS/MVS processes commands prior to other subsystems, set the SSICMD parameter to YES. For a description of the SSICMD parameter, see the *Parameter Reference*.

OPSLOG Browse Column: Text is always displayed.

CMD.TYPE

The exit environment where the command was trapped

Data Type: Character, read-only

Possible Values:

- MVS-Subsystem interface exit
- JES3-JES3 IATUX18 exit
- IMS-IMS AOI or command processing exit
- NONE-Command was not obtained from an exit

OPSLOG Browse Column: Not applicable

CMD.USER

An 8-byte variable providing communication between rules executing for the same command event. The variable can contain any installation data that these rules need, and it can store a character string displayable through OPSLOG Browse.

Data Type: User-defined, read/write

Notes:

- Before AOF processing, this variable is initialized to binary zeros. It is then passed to each rule that executes for the same command event. Each rule can look at or change the variable contents before passing the variable to the next rule for the command event.
- The primary purpose for the USER variable is to provide a method to pass a small amount of data between the rules. This data may be binary or mixed case. The USER field may also be used for filtering in the OPSLOG. However, USER data used for OPSLOG filtering must be uppercase and displayable.

OPSLOG Browse Column: USER

CMD.USERID

The security user ID of the command issuer for the security product on your system. This value is usually the CA ACF2, CA Top Secret, or RACF user ID from the UTOKEN associated with the command. If the command was issued from CA Remote Console, this value is the user ID of the particular RCS user who issued the command.

Data Type: Character, read-only

Sample Value: TSOID01

Note: The CMD.USERID variable may contain the same value as the JOBNAME (which is typical for a TSO address space). These values need not match; for example, user IDs and the job names for batch jobs or started tasks may differ.

OPSLOG Browse Column: USERID

CMD.VERB

The command verb as CA OPS/MVS extracts it

Sample Value: DISPLAY

Data Type: Character, read-only

Notes:

- The CMD.VERB value determines which command rules execute for a command event.
- The variable contains the best verb found in the current command. For example, z/OS command aliases such as F for MODIFY are converted to the base verb, MODIFY. Likewise, z/OS command abbreviations and JES3 command aliases or abbreviations also revert to the base verb.
- Except for $ADD, $DEL, $VS, and $TRACE, JES2 command verbs are always the first two characters of the command. For example, $D is the verb in the JES2 command $DJ1234.
- In cases such as IMS commands, the command verb is the first blank delimited word of the command text.
- Because the command verb is read-only, do not change it.

OPSLOG Browse Column: MSGID

CMD.XCONID

The decimal value corresponding to the 4-byte extended console ID of the console that issued the command. This field is not set for IMS commands.

Data Type: Integer, read-only

Sample Value: 16777253

Note: The XCONID column in OPSLOG displays this value as a hexadecimal value. For example, an extended console ID of 16777253 will be displayed in the XCONID column in OPSLOG as M01000025.

OPSLOG Browse Column: XCONID

More information:

AOF Rule Tools (see page 29)

Debug a CMD Rule

For debugging techniques that you can use with all AOF rules, see the chapter "Code and Debug AOF Rules (see page 59)."

Examples of CMD Rules

- Example 1: This example show how to have a CMD rule secure that only JES2 initiator control commands ($TIx) can be issued from the current sysplex master console:

```
)CMD $T
)PROC
/* Attempting to fire on a JES2 command means we must have a */
/* specifier of the JES2 command character (normally $)      */
/* followed by the first letter of the desired command       */
/* (T FOR TIxxx). Since many JES2 commands begin with a 'T'  */
/* (For example: $TIXXX,TPRTXXX) we must check the           */
/* environmental variable CMD.TEXT to see the exact text of  */
/* the command that was entered. Leave the rule if this      */
/* is not a JES2 initiator control command. SSICMD parm must */
/* be set to YES for JES2 CMD control.                       */

if SUBSTR(CMD.TEXT,1,3) ¬= '$TI' then
  return

/* Use the OPS/REXX OPSINFO function to get current sysplex  */
/* master console value, then compare this value to the      */
/* value of the console that issued the command which is     */
/* contained in the CMD.CONSNAME event variable. If this     */
/* is not the sysplex master, we'll send a message back to   */
/* console and null out the command so JES2 won't see it.    */

PLEXMSTR= OPSINFO('MSTCONSNM')
if CMD.CONSNAME ¬= PLEXMSTR then
  do
    msgtxt = 'JES2 init control not allowed from this console'
    ADDRESS WTO
    "MSGID(OPSMVS01) TEXT('"msgtxt"') HILITE",
    "CNNAME("CMD.CONSNAME")"
    CMD.TEXT = ''
    return 'ACCEPT'
  end
else
  return                                  /* OK to issue */
```

- Example 2: This example shows how to use a pseudo command rule to cycle a VTAM node:

```
)CMD VNET
)PROC
/* The purpose of this pseudo CMD rule is to give operators  */
/* or anyone wanting to cycle any VTAM node, a tool to       */
/* facilitate the issuing of the V NET,INACT and V NET,ACT   */
/* commands with one command. From any console you simply    */
/* enter 'VNET nodeid' and the logic of this rule will simply*/
/* issue a V NET,INACT and then a V NET,ACT command to the   */
/* extracted nodeid using the console that invoke the pseudo */
/* command so that the command responses get routed back.    */

NODEID= WORD(CMD.TEXT,2)         /* get the passed node id   */
ADDRESS OPER                     /* Issue vtam command .     */
"COMMAND(V NET,INACT,ID="NODEID",F) CONNAME("CMD.CONSNAME")"
"COMMAND(V NET,ACT,ID="NODEID",SCOPE=ALL) CONNAME("CMD.CONSNAME")"
return 'ACCEPT'                  /* z/OS won't see pseudo cmd */
```

Delete-Operator-Message Rules

An AOF delete-operator-message (DOM) rule is triggered when any system component issues the z/OS DOM macro instruction to low-light some previously issued action message. For example, when a tape mount has been satisfied, the associating highlighted tape mount message gets internally DOMed or low-lighted, thus producing a DOM event that could be trapped through a DOM rule. DOM rules are commonly used in conjunction with other rules, such as MSG and TOD rules, to create applications such as monitoring tape mounts and performing some type of notification for outstanding tape mounts. In addition, DOM rules can be useful for console consolidation, allowing you to send DOM macro requests from several systems to a single system.

Installation Requirements for DOM Rules

None are required.

)DOM—Event Specifier of DOM Rules

The following is the format for coding the DOM-event definition section:

```
)DOM *
```

Because a DOM rule responds to every DOM event on the system, the event specifier will always be an asterisk. Logic can be added to manipulate the DOM.WTOID event variable (as shown in the following example section) to detect specific DOM events.

Initialization, Processing, and Termination Sections of DOM Rules

DOM rules follow the general guidelines for coding the initialization, processing, and termination sections and the various AOF tools that you can use.

More information:

AOF Rule Structure (see page 21)
AOF Rule Tools (see page 29)

RETURN Statements in the)PROC Section of a DOM Rule

The OPS/REXX RETURN statement has no special meaning in the processing section of a DOM rule. The return value has no effect on AOF processing.

Execution Considerations for DOM Rules

The processing section of a rule that responds to a DOM event executes in the address space in which the DOM was issued. Therefore, any type of logic that could possibly suspend the processing of a DOM rule should be performed by scheduling an OPS/REXX program to a CA OPS/MVS OSF TSO, TSL, or TSP server.

More information:

Code and Debug AOF Rules (see page 59)

OPS/REXX Host Environments in the)PROC Section of a DOM Rule

The)PROC section of a DOM rule has the following host environments with the following DOM rule characteristics. Specify the AOFDEFAULTADDRESS parameter for the default host environment for DOM rules.

ADDRESS AOF

Sent to CA OPS/MVS. Does not wait. Sent as a WTO. Output is not returned.

ADDRESS AP

Sent to MSF and then forwarded to the CA Automation Point system. Does not wait. Output is not returned.

ADDRESS EPI

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS HWS

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS ISPEXEC

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS LXCON

The VM and Linux command requests sent to USS server for execution. Does not wait. Output is not returned. The List request runs inline and returns VM and Linux system data in stem variables.

ADDRESS MESSAGE

Sent as a WTO. The AOFDEST parameter specifies the destination.

ADDRESS MQ

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS NETMAN

Sent to a CA OPS/MVS internal CA Netman request queue to issue MGPT commands. Does not wait. Output is not returned.

ADDRESS NETMASTR

Sent to CA NetMaster NM for SNA on the local system. Does not wait. Output is not returned.

ADDRESS OPER

Sent to target console as specified through the OCCONSOLENAME parameters when no CONNAME operands are present. Output is not returned. Schedule an OPS/REXX program in a server when a command output interrogation is needed.

ADDRESS OPSCTL

Sent to a specified facility. If the facility is ECF or OSF, does not wait. If the facility is MSF, slight wait occurs. The external data queue returns the output.

Note: If the command is MSF LIST, no wait occurs.

ADDRESS OPSDYNAM

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS OSF

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSO servers.

ADDRESS OSFTSL

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSL servers.

ADDRESS OSFTSP

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSP servers.

ADDRESS SOF

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS SQL

Does not wait. Processed synchronously for requests that can be satisfied on the local system. Output is returned in stem variable. Error messages, if any, are returned to an external data queue.

ADDRESS SYSVIEWE

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS TSO

Sent to an OSF TSO server. Does not wait. Output is not returned.

ADDRESS USS

Sent to a USS server. Does not wait. Output is not returned. Schedule an OPS/REXX program in a server if the command output interrogation is needed.

ADDRESS WTO

Does not wait. Output is sent to the specified console. When attempting a WTOR, host command is sent to a TSO server for execution. The response is returned to the server. Schedule an OPS/REXX program in a server if WTOR response interrogation is needed.

More information:

Code and Debug AOF Rules (see page 59)

AOF Variables Available in a DOM Rule

You can use all AOF variable types in DOM rules. You can use the following unique AOF event variables in the)PROC section of a DOM rule, and you can manually interrogate the corresponding OPSLOG display field as an aid in debugging or implementing rule logic.

DOM.SYSPLEX

Indication of whether this DOM was routed to this system by sysplex processing

Data Type: Integer, read-only

Possible Values: 0 if the DOM was issued on this system; 1 if it was routed to this system by sysplex processing

Sample Value: 1

Note: Sysplex reissued DOMs are only processed by AOF rules if the AOFMESSAGES parameter is set to a value of MVSGLOBAL.

OPSLOG Browse Column: X'0004' flag of OPSFLAGS when the EVENT column indicates DOM event

DOM.TOKEN

The TOKEN value used to DOM one or more messages that were WTOed using the same TOKEN value. For more information, see the explanations for using the TOKEN keyword with the ADDRESS WTO host command environment and the OPSWTO command processor.

Data Type: 4-byte binary (possibly printable), read-only

Possible Values: Any possible 4-byte token used by the issuer of the WTO

Sample Value: A009

Notes:

- You can use a token to DOM a group of related messages that were WTOed using that same TOKEN value.
- The token may or may not contain a printable value. If it is not printable, you can display its hexadecimal value using the C2X(DOM.TOKEN) REXX construct.
- When the TOKEN field is not binary zeros, then the value of the DOM.WTOID variable will be binary zeros and the value of DOM.WTOIDNUM will be zero.
- When DOM.WTOIDNUM is non-zero, this variable will always contain binary zeros.

OPSLOG Browse Column: TOKEN

DOM.USER

An 8-byte variable providing communication between rules executing for the same DOM event. The variable can contain any installation data that these rules need, and it can store a character string displayable through OPSLOG Browse.

Data Type: User-defined, read/write

Notes:

- Before AOF processing, this variable is initialized to binary zeros. It is then passed to each rule that executes for the same DOM event; each rule can look at or change the variable contents before passing the variable to the next rule for the DOM event.
- The primary purpose for the USER variable is to provide a method to pass a small amount of data between the rules. This data may be binary or mixed case. The USER field may also be used for filtering in the OPSLOG. However, USER data used for OPSLOG filtering must be uppercase and displayable.

OPSLOG Browse Column: USER

DOM.WTOID

The internal sequence ID of the message to be deleted, taken from the DOMCID field of the DOM control block (WTO or WTOR sequence number)

Data Type: 4-byte binary (unprintable), read-only

Sample Value: '00004C94'

Notes:

- The DOM.WTOID value provides information that the OPSSEND function of OPS/REXX uses in transmitting DOM events to other systems.
- Your system assigns sequence numbers to WTO or WTOR messages. You can use these sequence numbers to delete (using DOM) highlighted, non-scrollable messages. Use the sequence of a message as its unique token identifier, but do not use the contents of the sequence number itself.

OPSLOG Browse Column: WTOID

DOM.WTOIDNUM

The internal sequence ID of the message to be deleted, taken from the DOMCID field of the DOM control block (WTO or WTOR sequence number)

Data Type: Integer, read-only

Sample Value: 1694508484

Note: The only difference between this variable and the DOM.WTOID variable is its display format.

OPSLOG Browse Column: WTOID

More information:

AOF Rule Tools (see page 29)

Debug a DOM Rule

To debug a DOM Rule

1. Set the CA OPS/MVS BROWSEDOM parameter to YES
2. Set the DOM event profile of your OPSLOG display to view all DOM events.

   OPSLOG will now record entries of DOM.
3. With these parameters set, display the OPSLOG WTOID column to see the associating message ID.

   The WTOID on the DOM event should be that of the WTOID for the associating action message in the OPSLOG in which the DOM occurred.

More information:

Code and Debug AOF Rules (see page 59)

Example: DOM Rule

The following is an example of a rule that responds to a DOM event. The example assumes that you have an associating MSG rule that is sending messages to some focal system. This DOM rule would allow those remotely sent action messages to be low-lighted. This might be a desired application in a non-sysplex CA OPS/MVS MSF connected environment, or a normal sysplex.

```
)DOM *
)PROC
/* SYSTEM A is our focal system, as to which we are currently */
/* shipping all message traffic. This DOM rule will use       */
/* the OPS/REXX OPSSEND function to ship over this DOM event  */
/* We also need to exclude shipping sysplex generated DOMS    */
/* to avoid a possible rule loop.                             */

 if DOM.SYSPLEX = 0 then             /* locally generated DOM */
   sendrc = OPSSEND(“SYSA”,”D”)

 return
```

Note: The AOFMESSAGES parameter controls whether reissued messages and DOMs are processed by AOF rules. This includes messages and DOMs that originated on another system and were transported and reissued on this system by MSF, CA MIC, or sysplex services.

See examples TAPEMNT1, TAPEMNT2, and TAPEMNT3 of your CA OPS/MVS data set that contains the downloaded OPS.CCLXRULS file. These examples demonstrate how to use a DOM rule in conjunction with an MSG and TOD rule to control outstanding tape mounts.

End-of-Job Rules

An end-of-job (EOJ) rule is triggered when any job or started task ends. EOJ rules facilitate the process of detecting when a job or started task ends, because one EOJ rule usually replaces several MSG rules that need to be coded to detect job ending states such as abend and normal termination messages. In addition, some address spaces may end silently (no message notification). An EOJ rule can effectively detect this type of termination.

In many cases, you can use an EOJ rule (and an EOS rule) to replace your existing IEFACTRT SMF exit, which is written in assembler language.

Installation Requirements for EOJ Rules

Set the parameters INITSMF, EOJRULES, and EOSRULES to YES.

The installation IEFACTRT SMF exits must be implemented and SMF type 30 subtype records must be generated.

Note: For more information, see the *Parameter Reference*.

)EOJ—Event Specifier of EOJ Rules

The following is the format for coding the EOJ-event definition section:

```
)EOJ jobnamespec
```

jobnamespec

Specifies the job name. Follow these guidelines when specifying the character string:

- Specify one to eight characters of the job name.
- The string cannot contain embedded blank spaces.
- You can use the wildcard (*) character. For example,
  - CICS* matches CICSA, CICSABC, CICS123 and any other job name containing a CICS prefix.
  - CICS*05 matches CICSD05, CICS205, CICS1105, and so on.
  - *05 matches any job name ending with 05.
  - * alone matches all job names.
- Lowercase letters are acceptable, but the AOF converts them to uppercase for event testing.

Initialization, Processing, and Termination Sections of EOJ Rules

The general guidelines for coding the initialization, processing, and termination sections and the various AOF tools that you can use, apply to EOJ rules.

More information:

AOF Rule Structure (see page 21)
AOF Rule Tools (see page 29)

RETURN Statements in the)PROC Section of an EOJ Rule

The OPS/REXX RETURN statement has no special meaning in the processing section of an EOJ rule. The return value has no effect on AOF processing.

Execution Considerations for EOJ Rules

The processing section of a rule that responds to an EOJ event executes in the address space of the job or task that is ending. Therefore, schedule an OPS/REXX program to a CA OPS/MVS OSF TSO, TSL, or TSP serve to perform any type of logic that could possibly suspend the processing of an EOJ rule.

The active JSCB in the ending address space is the region control task, or the initiator. This causes the ACCOUNT, EXECPGM, and MODULE operands of OPSINFO to return the values for the RCT or initiator program, rather than the application program that is ending.

Since SMF may generate more than one type 30 record for reach EOJ event, only the first record containing all the job data that only occurs once is used to generate the EOJ event. The additional records containing repeatable sections such as EXCP counts for every data set are not visible to the EOJ rules.

More information:

Code and Debug AOF Rules (see page 59)

OPS/REXX Host Environments in the)PROC Section of an EOJ Rule

The)PROC section of an EOJ rule has the following host environments with the following EOJ rule characteristics. Specify the AOFDEFAULTADDRESS parameter for the default host environment for EOJ rules.

ADDRESS AOF

Sent to CA OPS/MVS. Does not wait. Output is not returned.

ADDRESS AP

Sent to MSF and then forwarded to the CA Automation Point system. Does not wait. Output is not returned.

ADDRESS EPI

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS HWS

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS ISPEXEC

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS LXCON

The VM and Linux command requests sent to USS server for execution. Does not wait. Output is not returned. The List request runs inline and returns VM and Linux system data in stem variables.

ADDRESS MESSAGE

Sent as a WTO. The AOFDEST parameter specifies the destination.

ADDRESS MQ

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS NETMAN

Sent to a CA OPS/MVS internal CA Netman request queue to issue MGPT commands. Does not wait. Output is not returned.

ADDRESS NETMASTR

Sent to CA NetMaster NM for SNA on the local system. Does not wait. Output is not returned.

ADDRESS OPER

Sent to target console as specified through the OCCONSOLENAME parameters when no CONNAME operands are present. Output is not returned. Schedule an OPS/REXX program in a server when a command output interrogation is needed.

ADDRESS OPSCTL

Sent to a specified facility. If the facility is ECF or OSF, does not wait. If the facility is MSF, slight wait occurs. The external data queue returns the output.

Note: If the command is MSF LIST, no wait occurs.

ADDRESS OPSDYNAM

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS OSF

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSO servers.

ADDRESS OSFTSL

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSL servers.

ADDRESS OSFTSP

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSP servers.

ADDRESS SOF

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS SQL

Does not wait. Proceed synchronously for requests that can be satisfied on the local system. Output is returned in stem variable. Error messages, if any, are returned to an external data queue.

ADDRESS SYSVIEWE

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS TSO

Sent to an OSF TSO server. Does not wait. Output is not returned.

ADDRESS USS

Sent to a USS server. Does not wait. Output is not returned. Schedule an OPS/REXX program in a server if the command output interrogation is needed.

ADDRESS WTO

Does not wait. Output is sent to the specified console. When attempting a WTOR, the host command is sent to a TSO server for execution. The response is returned to the server. Schedule an OPS/REXX program in a server if the WTOR response interrogation is needed.

More information:

Code and Debug AOF Rules (see page 59)

AOF Variables Available in an EOJ Rule

You can use all AOF variable types in EOJ rules. You can use the following unique AOF event variables in the)PROC section of a EOJ rule, and you can manually interrogate the corresponding OPSLOG display field as an aid in debugging or implementing rule logic.

EOJ.ACCOUNT

The value of the job card accounting field. This field is a character string where each accounting field is separated by a comma.

Data Type: Character, read-only

Sample Value: 12 (Reason code 12)

EOJ.COLOR

The color that the message text has in OPSLOG Browse

Data Type: 1-byte binary (unprintable), read/write

Sample Value: '00'X

Note: Use the OPSCOLOR function of OPS/REXX to set the EOJ.COLOR variable.

OPSLOG Browse Column: COLOR

EOJ.CONDCODE

The condition code of the last step of the job that was executed. The format is the same as EOJ.MAXCC and is often the same value. The value for this field is derived from SMF30SCC in the type 30 SMF record.

Data Type: Character, read-only

Sample Value: 0008 (Condition code 8)

EOJ.CPUSRB

The amount of CPU time (measured in hundredths of seconds) that was consumed by the job while running in SRB mode. This is roughly equivalent to the amount of CPU time to service I/O requests by the application. This value is field SMF30CPS in the type 30 SMF record.

Data Type: Integer, read-only

Sample Value: 20 (.2 seconds of SRB time consumed)

EOJ.CPUTCB

The amount of CPU time (measured in hundredths of seconds) that was consumed by the job while running under a z/OS TCB. This is roughly equivalent to the CPU usage of the application program. This value is field SMF30CPT in the type 30 SMF record.

Data Type: Integer, read-only

Sample Value: 910 (9.1 seconds of CPU time consumed)

EOJ.EXCPCNT

The total number of data blocks transferred from I/O channel program executions. This is a measure of the amount of I/O done by the job. This value is field SMF30TEP in the type 30 SMF record.

Data Type: Integer, read-only

Sample Value: 400 (400 blocks transferred)

EOJ.JOBCLASS

The JES job class for an initiated batch job. This value is field SMF30C18 in the type 30 SMF record.

Data Type: Character, read-only

Sample Value: A

EOJ.JOBNAME

The job name or started task that has ended. JOBNAME is taken from the identification section of the SMF type 30 record.

Data Type: Character, read-only

Sample Value: IBMUSER

OPSLOG Browse Column: JOBNAME

EOJ.MAXCC

The maximum condition code of any step executed during the job. This value is always five characters.

The condition code has the following formats:

- S0XXX-System hexadecimal abend code
- unnnn-User decimal abend code
- nnnnn-Normal decimal return code
- FLUSH-All steps of the job were flushed

System abends are considered the highest values, followed by user abends and normal return codes. FLUSH is only returned if all steps of the job are not executed.

Data Type: Character, read-only

Sample Value: S00C7 (system abend 0C7)

Note: EOSRULES needs to be set to YES for the EOJ.MAXCC to work properly. The end-of-step cc is tracked to place the proper data in this variable.

EOJ.NONSPTAPE

The number of non-specific tape mounts for the job. This value is field SMF30PTM in the type 30 SMF record.

Data Type: Integer, read-only

Sample Value: 2

EOJ.PGMRNAME

The programmer name field from the JOB statement. This value is field SMF30USR in the type 30 SMF record.

Data Type: Character, read-only

Sample Value: JOHN DOE

EOJ.REASCODE

If an abend occurs in the last executed step of the job, the reason code passed in register 15 is sometimes a reason code for the abend. The value for this field is derived from SMF30ARC in the type 30 SMF record.

Data Type: Integer, read-only

Sample Value: 12 (Reason code 12)

EOJ.RESGROUP

The WLM resource group name for the job if the system is using the z/OS Workload Manager for system management. This value is field SMF30GRN in the type 30 SMF record.

Data Type: Character, read-only

Sample Value: ALLCICS

EOJ.SECGROUP

The security group ID taken from the ACEE. This value is field SMF30GRP in the type 30 SMF record.

Data Type: Character, read-only

Sample Value: OPERS

EOJ.SECUSER

The security user ID taken from the ACEE. This value is field SMF30RUD in the type 30 SMF record.

Data Type: Character, read-only

Sample Value: OPER1

EOJ.SERVCLAS

The WLM service class for the job if the system is using the z/OS Workload Manager for system management. This value is field SMF30SCN in the type 30 SMF record.

Data Type: Character, read-only

Sample Value: SPEEDY

EOJ.SMF30AD

The address of the SMF type 30 record. This address may be used with the OPSTORE function of OPS/REXX to access any field in the type 30 record to obtain data that is not provided by the EOJ event variables. The IBM macro IFASMFR (30) generates the assembler DSECT for the SMF type 30 record.

Data Type: 4-byte binary (unprintable), read-only

Sample Value: '00702C00'X

EOJ.SPTAPE

The number of volume specific tape mounts for the job. This value is field SMF30TPR in the type 30 SMF record.

Data Type: Integer, read-only

Sample Value: 6

EOJ.STARTDATE

The sample value date that the system began execution of this job or started task. The date is in the format YYYY/MM/DD.

Data Type: Character, read-only

Sample Value: 2000/05/12

EOJ.STARTTIME

The time that the system began execution of this job or started task. The time value is in hundredths of seconds since midnight.

Data Type: Character, read-only

Sample Value: 3600000 (10AM)

EOJ.SUBSYS

The subsystem name of the job used by SMF for workload accounting. Subsystem names are defined in the SMFPRMxx member of PARMLIB and extracted from the OUCBSUBN field of the OUCB control block.

Data Type: Character, read-only

Sample Value: TSO

EOJ.TERMNAME

The symbolic name of the TSO terminal for a TSO session. This value is field SMF30TSN in the type 30 SMF record.

Data Type: Character, read-only

Sample Value: OPSS1

OPSLOG Browse Column: TERMNAME

EOJ.TEXT

The OPSLOG message text that describes the end of a job event including the maximum condition code

Data Type: Character, read-only

Sample Value: IBMUSER JOB00123 ENDED MAXCC=00000 SUBSYS=TSO

OPSLOG Browse Column: Text is always displayed

EOJ.USER

An 8-byte variable providing communication between rules executing for the same EOJ event. The variable can contain any installation data that these rules need, and it can store a character string displayable through OPSLOG Browse.

Data Type: User-defined, read/write

Notes:

Before AOF processing, this variable is initialized to binary zeros. It is then passed to each rule that executes for the same EOJ event; each rule can look at or change the variable contents before passing the variable to the next rule for the EOJ event.

The primary purpose for the USER variable is to provide a method to pass a small amount of data between the rules. This data may be binary or mixed case. The USER field may also be used for filtering in the OPSLOG. However, USER data used for OPSLOG filtering must be uppercase and displayable.

OPSLOG Browse Column: USER

EOJ.USERCOM

The value contained in the JMRUCOM of the JMR control block. This field is sometimes used to point to tables or control blocks used by installation SMF exits.

Data Type: 4-byte binary (unprintable), read-only

Sample Value: '0A002CFC' X

Note: Use the OPSTORE function of OPS/REXX to access any storage pointed to by EOJ.USERCOM.

EOJ.WORKLOAD

The WLM workload name of the job if the system is using the z/OS Workload Manager for system management. This value is field SMF30WLM in the type 30 SMF record.

Data Type: Character, read-only

Sample Value: PRODCICS

More information:

AOF Rule Tools (see page 29)

Debug an EOJ Rule

The following are EOJ Rule Debugging Techniques:

- Set the CA OPS/MVS BROWSEEOJ parameter to YES and the EOJ event profile of your OPSLOG display to Y to view all EOJ events. With these parameters set, display the OPSLOG EVENT column to see recorded EOJ events.
- If OPSLOG is not recording EOJ events, see Installation Requirements for EOJ Rules in this chapter.

For additional debugging techniques that you can use with all AOF rules, see the chapter “Code and Debug AOF Rules.”

More information:

Code and Debug AOF Rules (see page 59)

Example: EOJ Rule

Assume that you defined an RDF table containing information on the production payroll jobs. When each batch job in the processing sequence ends, the start and stop times and the maximum condition codes must be recorded.

```
)EOJ PAY*
)PROC
/* Only process productions jobs a by checking EOJ event variable */

if eoj.jobclass = 'P' then

/* Update RDF table with the EOJ data. Call the internal CONVTIME */
/* subroutine to format the start time to HH:MM:SS. This value    */
/* is originally in hundredths of seconds since midnight.         */

 address SQL Update PAYTAB Set
  "Start_date='"eoj.startdate" ',",
  "Start_time='"Convtime(eoj.starttime)"' ,",     /* convert data */
  "End_date='"Date('B')"',",
  "End_time='"Time('N')"',",
  "Cond_code='"eoj.maxcc"'",
  "Where Jobname='"eoj.jobname" ' "

return "NORMAL"

/* Convert binary time from hundredths of seconds since midnight  */
CONVTIME:
 cvtime=Arg(1)%100
 return Right(cvtime%3600,2,'0')':'||,    /* HH: */
   Right((cvtime%60)//60,2,'0')':'||,     /* MM: */
   Right(cvtime//60,2,'0')                /* SS  */

)END
```

End-of-Memory Rules

An end-of-memory (EOM) rule is triggered when a started task ends. EOM rules facilitate the process of detecting when a started task ends, because one EOM rule usually replaces several MSG rules that need to be coded to detect job ending states such as abend and termination messages. In addition, some stated tasks might end silently (no message notification), and an EOM can effectively detect this type of termination.

Note: EOM rules have similar functionality to EOJ rules. The primary difference is that EOM rules only detect the ending of an address space and cannot detect the end of a batch job, as do EOJ rules. Also, the amount of environmental data available in EOJ rules is much greater than in EOM rules. In some cases you may choose an EOJ rule instead of an EOM rule.

Installation Requirements for EOM Rules

None are required.

)EOM—Event Specifier of EOM Rules

The following is the format for coding the EOM-event definition section:

`)EOM` *`jobnamespec`*

jobnamespec

Specifies the job name. Follow these guidelines when specifying the character string:

- Specify one to eight characters of the job name.
- The string cannot contain embedded blank spaces.
- You can use the wildcard (*) character. For example,
  - CICS* matches CICSA, CICSABC, CICS123 and any other job name containing a CICS prefix
  - CICS*05 matches CICSD05, CICS205, CICS1105, and so on
  - *05 matches any job name ending with 05
  - * alone matches all job names
- Lowercase letters are acceptable, but the AOF converts them to uppercase for event testing.

Initialization, Processing, and Termination Sections of EOM Rules

EOM rules follow the general guidelines for coding the initialization, processing, and termination sections and the various AOF tools that you can use.

More information:

AOF Rule Structure (see page 21)
AOF Rule Tools (see page 29)

RETURN Statements in the)PROC Section of an EOM Rule

The OPS/REXX RETURN statement has no special meaning in the processing section of an EOM rule. The return value has no effect on AOF processing.

Execution Considerations for EOM Rules

The processing section of a rule that responds to an EOM event executes in the z/OS master (*MASTER*) address space. Therefore, any type of logic that could possibly suspend the processing of an EOM rule should be performed by scheduling an OPS/REXX program to a CA OPS/MVS OSF TSO, TSL, or TSP server.

More information:

Code and Debug AOF Rules (see page 59)

OPS/REXX Host Environments in the)PROC Section of an EOM Rule

The)PROC section of an EOM rule has the following host environments with the following EOM rule characteristics. Specify the AOFDEFAULTADDRESS parameter for the default host environment for EOM rules.

ADDRESS AOF

Sent to CA OPS/MVS. Does not wait. Output is not returned.

ADDRESS AP

Sent to MSF and then forwarded to the CA Automation Point system. Does not wait. Output is not returned.

ADDRESS EPI

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS HWS

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS ISPEXEC

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS LXCON

The VM and Linux command requests sent to USS server for execution. Does not wait. Output is not returned. The List request runs inline and returns VM and Linux system data in stem variables.

ADDRESS MESSAGE

Sent as a WTO. The AOFDEST parameter specifies the destination.

ADDRESS MQ

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS NETMAN

Sent to a CA OPS/MVS internal CA Netman request queue to issue MGPT commands. Does not wait. Output is not returned.

ADDRESS NETMASTR

Sent to CA NetMaster NM for SNA on the local system. Does not wait. Output is not returned.

ADDRESS OPER

Sent to target console as specified through the OCCONSOLENAME parameters when no CONNAME operands are present. Output is not returned. Schedule an OPS/REXX program in a server when a command output interrogation is needed.

ADDRESS OPSCTL

Sent to a specified facility. If the facility is ECF or OSF, does not wait. If the facility is MSF, slight wait occurs. The external data queue returns the output.

Note: If the command is MSF LIST, no wait occurs.

ADDRESS OPSDYNAM

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS OSF

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSO servers.

ADDRESS OSFTSL

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSL servers.

ADDRESS OSFTSP

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSP servers.

ADDRESS SOF

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS SQL

Does not wait. Processed synchronously for requests that can be satisfied on the local system. Output is returned in stem variable. Error messages, if any, are returned to an external data queue.

ADDRESS SYSVIEWE

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS TSO

Sent to an OSF TSO server. Does not wait. Output is not returned.

ADDRESS USS

Sent to a USS server. Does not wait. Output is not returned. Schedule an OPS/REXX program in a server if a command output interrogation is needed.

ADDRESS WTO

Does not wait. Output that is sent to specified (or default) destination. When attempting a WTOR, host command is sent to a TSO server for execution. The response is returned to the server. Schedule an OPS/REXX program in a server if the WTOR response interrogation is needed.

More information:

Code and Debug AOF Rules (see page 59)

AOF Variables Available in an EOM Rule

You can use all AOF variable types in EOM rules. You can use the following unique AOF event variables in the)PROC section of a EOM rule. You can also manually interrogate the corresponding OPSLOG display field as an aid in debugging or implementing rule logic.

EOM.ABNORMAL

An integer value indicting whether the address space terminated abnormally. This value is derived from the SSENTYPE flag in the SSOB extension.

Data Type: Integer, read-only

Possible values are:

- 0 = normal termination
- 1 = abnormal termination.

Abnormal does *not* include all abends. S222 abends are not considered abnormal. Abend S069 reason code 4 for ASCRE failure is considered abnormal.

Sample Value: 0

EOM.ASID

The address space ID of the terminating address space, which is taken from the SSENASID field of the SSOB extension.

Data Type: 2-byte binary (unprintable), read-only

Sample Value: '003E'X

Notes:

- To convert this variable to printable hexadecimal characters, use the OPS/REXX C2X function.
- Using this variable, you can check to see if an address space is terminating. For example, suppose that an automation application keeps a list of critical address spaces and their ASIDs. You can write an EOM rule that compares the list entries against the ASID of the currently terminating address space. If a match is found, restarts the address space or takes another recovery action.

OPSLOG Browse Column: ASID

EOM.JOBNAME

The job name of the terminated address space, which is taken from one of these sources:

- TSO users: The OUCBUSRD (user ID) field of the OUCB; the value is the user ID for time sharing address spaces
- Started tasks: The OUCBTRXN (transaction name) field of the OUCB; the value is the started task name for started task address spaces

Data Type: Character, read-only

Sample Value: VTAM

Note: The EOM.JOBNAME variable determines which EOM rules execute for an EOM event.

OPSLOG Browse Column: JOBNAME

EOM.TEXT

Description of the terminated address space, which is taken from the OUCBSUBN (subsystem name) field of the OUCB and from the EOM.JOBNAME variable.

Data Type: Character, read-only

Sample Value: TSO USERA

Notes:

- The type of terminated address space is either TSO for a TSO user or STC for a started task.
- You can extract the first word of the EOM.TEXT string to determine the current address space type use the OPS/REXX WORD function.

OPSLOG Browse Column: Text is always displayed.

EOM.USER

An 8-byte variable providing communication between rules executing for the same EOM event. The variable can contain any installation data that these rules need, and it can store a character string displayable through OPSLOG Browse.

Data Type: User-defined, read/write

Notes:

- Before AOF processing, this variable is initialized to binary zeros. This variable is then passed to each rule that executes for the same EOM event. Each rule can look at or can change the variable contents before passing the variable to the next rule for the EOM event.
- The primary purpose for the USER variable is to provide a method to pass a small amount of data between the rules. This data can be binary or mixed case. The USER field can also be used for filtering in the OPSLOG. However, USER data that is used for OPSLOG filtering must be uppercase and displayable.

OPSLOG Browse Column: USER

More information:

AOF Rule Tools (see page 29)

Debug an EOM Rule

To debug an EOM rule

1. Set the CA OPS/MVS BROWSEEOM parameter to YES
2. Set the EOM event profile of your OPSLOG display to Y

   You can now view all EOM events.
3. With these parameters set, display the OPSLOG EVENT column

   This lets you see recorded EOM events.

Note: If OPSLOG is not recording EOM events, see Installation Requirements for EOM Rules (see page 165) in this chapter.

More information:

Code and Debug AOF Rules (see page 59)

Example: EOM Rule

The following is an example of a rule that responds to an EOM event. The rule detects a VTAM failure and displays a highlighted message. This would be a catchall rule to detect if VTAM ended; individual MSG rules would not be needed.

```
)EOM VTAM
)INIT
/* Initialize a flag variable to be used in the )PROC section     */
/* to determine if system is being shutdown.                      */

GLVTEMP1.SYSSHUT.STAT = 'N'

)PROC
/* Only do this if we are not shutting down the system. If we are  */
/* shutting down the system, then our shutdown procedures will set */
/* this flag glvtemp variable to Y so that this code won't fire    */
SHUTDOWN = OPSVALUE('GLVTEMP1.SYSSHUT.STAT','O')      /* get value */
if SHUTDOWN = 'Y' then return                         /* get out   */
address WTO             /* OPS/REXX Host environment to issue msgs */
 "MSGID(OPSAUT01) TEXT('VTAM HAS ENDED ABNORNALLY') HILITE Route(1)"
)END
```

End-of-Step Rules

End-of-step (EOS) rules provide the ability to monitor and optionally terminate a batch job based on any criteria that can be extracted from the SMF type 30 record that is produced during step termination of a batch job. Since the EOS rule is matched by a job name specification, it can be used to monitor step completion of jobs that produce no message traffic at step termination that would be visible to message rules. In addition, the option to cancel a job based on completion code, or any other data provided in the SMF type 30 record, is provided.

EOS and EOJ rules provide you with the option of replacing your assembler language IEFACTRT SMF exit with easier-to-maintain AOF OPS/REXX rules.

Installation Requirements for EOS Rules

Set the parameters INITSMF and EOSRULES to YES.

The installation IEFACTRT SMF exits must be implemented and SMF type 30 subtype 4 records must be generated.

Note: For more information, see the *CA OPS/MVS Parameter Reference*.

)EOS—Event Specifier of EOS Rules

The following is the format for coding the EOS event definition section:

```
)EOS jobnamespec
```

jobnamespec

Specifies the job name. Follow these guidelines when specifying the character string:

- Specify one to eight characters of the job name.
- The string cannot contain embedded blank spaces.
- You can use the wildcard (*) character. For example,
  - CICS* matches CICSA, CICSABC, CICS123 and any other job name containing a CICS prefix.
  - CICS*05 matches CICSD05, CICS205, CICS1105, and so on.
  - *05 matches any job name ending with 05.
  - * alone matches all job names.
- Lowercase letters are acceptable, but the AOF converts them to uppercase for event testing.

Initialization, Processing, and Termination Sections of EOS Rules

The general guidelines for coding the initialization, processing, and termination sections and the various AOF tools that you can use, apply to EOS rules.

More information:

AOF Rule Structure (see page 21)
AOF Rule Tools (see page 29)

RETURN Statements in the)PROC Section of an EOS Rule

The OPS/REXX RETURN statement specifies whether the job containing the ended step is allowed to continue or is cancelled. Valid values for a RETURN statement in the processing section of an EOS rule are as follows:

NORMAL

Allow the job or started task to continue processing.

CANCEL

Bypass *all* remaining job steps and terminate the job.

Default: RETURN NORMAL

Notes:

- The return values listed here are character constants rather than keywords.
- An unrecognized return value defaults to a value of NORMAL.
- Inadvertently coding a RETURN SUPPRESS will have the same meaning as RETURN CANCEL.

Execution Considerations for EOS Rules

The processing section of a rule that responds to an EOS event executes in the address space of the job or task whose step has ended. Therefore, any type of logic that could possibly suspend the processing of an EOS rule should be performed by scheduling an OPS/REXX program to a CA OPS/MVS OSF TSO, TSL, or TSP server.

The active JSCB in the ending address space is the region control task, or the initiator. This causes the ACCOUNT, EXECPGM, and MODULE operands of OPSINFO to return the values for the RCT or initiator program, rather than the application program that is ending.

Because SMF may generate more than one type 30 record for reach EOS event, only the first record containing all the job data that only occurs once is used to generate the EOS event. The additional records containing repeatable sections such as EXCP counts for every data set are not visible to the EOS rules.

More information:

Code and Debug AOF Rules (see page 59)

OPS/REXX Host Environments in the)PROC Section of an EOS Rule

The)PROC section of an EOS rule has the following host environments with the following EOS rule characteristics. Specify the AOFDEFAULTADDRESS parameter for the default host environment for EOS rules.

ADDRESS AOF

Sent to CA OPS/MVS. Does not wait. Output is not returned.

ADDRESS AP

Sent to MSF and then forwarded to the CA Automation Point system. Does not wait. Output is not returned.

ADDRESS EPI

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS HWS

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS ISPEXEC

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS LXCON

The VM and Linux command requests sent to USS server for execution. Does not wait. Output is not returned. The List request runs inline and returns VM and Linux system data in stem variables.

ADDRESS MESSAGE

Sent as a WTO. The AOFDEST parameter specifies the destination.

ADDRESS MQ

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS NETMAN

Sent to a CA OPS/MVS internal CA Netman request queue to issue MGPT commands. Does not wait. Output is not returned.

ADDRESS NETMASTR

Sent to CA NetMaster NM for SNA on the local system. Does not wait. Output is not returned.

ADDRESS OPER

Sent to target console as specified through the OCCONSOLENAME parameters when no CONNAME operands are present. Output is not returned. Schedule an OPS/REXX program in a server when a command output interrogation is needed.

ADDRESS OPSCTL

Sent to a specified facility. If the facility is ECF or OSF, does not wait. If the facility is MSF, slight wait occurs. The external data queue returns the output.

Note: If the command is MSF LIST, no wait occurs.

ADDRESS OPSDYNAM

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS OSF

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS servers.

ADDRESS OSFTSL

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSL servers.

ADDRESS OSFTSP

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSP servers.

ADDRESS SOF

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS SQL

Does not wait. Processed synchronously for requests that can be satisfied on the local system. Output is returned in stem variable. Error messages, if any, are returned to an external data queue.

ADDRESS SYSVIEWE

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS TSO

Sent to an OSF TSO server. Does not wait. Output is not returned.

ADDRESS USS

Sent to a USS server. Does not wait. Output is not returned. Schedule an OPS/REXX program in a server if a command output interrogation is needed.

ADDRESS WTO

Does not wait. Output is sent to the specified console. When attempting a WTOR, host command is sent to a TSO server for execution. The response is returned to the server. Schedule an OPS/REXX program in a server if the WTOR response interrogation is needed.

More information:

Code and Debug AOF Rules (see page 59)

AOF Variables Available in an EOS Rule

You can use all AOF variable types in EOS rules. You can use the following unique AOF event variables in the)PROC section of a EOS rule, and you can manually interrogate the corresponding OPSLOG display field as an aid in debugging or implementing rule logic.

EOS.ACCOUNT

The value of the EXEC statement account keyword or a null string if no EXEC accounting is specified. This field is a character string, where a comma separates each accounting field.

Data Type: Character, read-only

Sample Value: PROD,HR,,05210

EOS.COLOR

The color that the message text has in OPSLOG browse

Data Type: 1-byte binary (unprintable), read/write

Sample Value: '00'X

Note: Use the OPSCOLOR function of OPS/REXX to set the EOS.COLOR variable.

OPSLOG Browse Column: COLOR

EOS.CONDCODE

The condition code of the current step that has ended. The format is the same as EOS.MAXCC and may be the same value. The value for this field is derived from SMF30SCC in the type 30 SMF record.

Data Type: Character, read-only

Sample Value: 00008 (Condition code 8)

EOS.CPUSRB

The amount of CPU time, in hundredths of seconds, that was consumed by the step while running in SRB mode. This is roughly equivalent to the amount of CPU time to service I/O requests by the application. This value is field SMF30CPS in the type 30 SMF record.

Data Type: Integer, read-only

Sample Value: 20 (.2 seconds of SRB time consumed)

EOS.CPUTCB

The amount of CPU time, in hundredths of seconds, that was consumed by the step while running under a z/OS TCB. This is roughly equivalent to the CPU usage of the application program. This value is field SMF30CPT in the type 30 SMF record.

Data Type: Integer, read-only

Sample Value: 910 (9.1 seconds of CPU time consumed)

EOS.EXCPCNT

The total number of data blocks transferred from I/O channel program executions. This is a measure of the amount of I/O completed by the step. This value is field SMF30TEP in the type 30 SMF record.

Data Type: Integer, read-only

Sample Value: 400 (400 blocks transferred)

EOS.JOBCLASS

The JES job class for an initiated batch job. This value is field SMF30C18 in the type 30 SMF record.

Data Type: Character, read-only

Sample Value: A

EOS.JOB NAME

The name of the job or started task whose step has ended. JOB NAME is taken from the identification section of SMF type 30 record.

Data Type: Character, read-only

Sample Value: IBMUSER

OPSLOG Browse Column: JOB NAME

EOS.MAXCC

The maximum condition code of any step executed, up to and including the current step. This value is always a 5-character value. The condition code has the following formats:

- S0XXX-system hexadecimal abend code
- *Unnnn*-user decimal abend code
- *nnnnn*-normal decimal return code
- FLUSH-all steps of the job were flushed

System abends are considered the highest values, followed by user abends and normal return codes. FLUSH is only returned if all steps, up to and including the current step, were not executed.

Data Type: Character, read-only

Sample Value: S00C7 (System abend 0C7)

EOS.NONSPTAPE

The number of non-specific tape mounts for the step. This value is field SMF30PTM in the type 30 SMF record.

Data Type: Character, read-only

Sample Value: 2

EOS.OESUBSTEP

The Open MVS UNIX sub-step number that is incremented by 1 each time the OMVS EXEC function is invoked. This field is 0 for z/OS programs. This value is field SMF30SSN in the type 30 SMF record.

Data Type: Integer, read-only

Sample Value: 25

EOS.PGMNAME

The name of the job step program specified in the PGM keyword of the JCL EXEC statement. For an OMVS UNIX program, this field may be 16 characters in length and reflects the file name specified at the end of the UNIX path specification. This value is field SMF30PGM in the type 30 SMF record.

Data Type: Character, read-only

Sample Value: IEBGENER

EOS.PGMRNAME

The programmer name field from the JOB statement. This value is field SMF30USR in the type 30 SMF record.

Data Type: Character, read-only

Sample Value: JOHN DOE

EOS.PROCSTEP

If the EXEC statement invokes a catalogued procedure, this variable is the name of the EXEC statement. Otherwise, it is a null string. This value is field SMF30PSN in the type 30 SMF record.

Data Type: Character, read-only

Sample Value: ASMHCLG

EOS.REASCODE

If an abend occurs in the terminating step, the value passed in register 15 is sometimes a reason code for the abend. The value for this field is derived from SMF30ARC in the type 30 SMF record.

Data Type: Integer, read-only

Sample Value: 12 (Reason code 12)

EOS.RESGROUP

The WLM resource group name for the job if the system is using the z/OS Workload Manager for system management. This value is field SMF30GRN in the type 30 SMF record.

Data Type: Character, read-only

Sample Value: ALLCICS

EOS.SECGROUP

The security group ID taken from the ACEE. This value is field SMF30GRP in the type 30 SMF record.

Data Type: Character, read-only

Sample Value: OPERS

EOS.SECUSER

The security user ID taken from the ACEE. This value is field SMF30GRP in the type 30 SMF record.

Data Type: Character, read-only

Sample Value: OPER1

EOS.SERVCLAS

The WLM service class for the job if the system is using the z/OS Workload Manager for system management. This value is field SMF30SCN in the type 30 SMF record.

Data Type: Character, read-only

Sample Value: SPEEDY

EOS.SMF30AD

The address of the SMF type 30 record. This address may be used with the OPSTORE function of OPS/REXX to access any field in the type 30 record to obtain data not provided by the EOS event variables. The IBM macro IFASMFR (30) generates the assembler DSECT for the SMF type 30 record.

Data Type: 4-byte binary (unprintable), read-only

Sample Value: '00702C00'X

EOS.SPTAPE

The number of volume specific tape mounts for the step. This value is field SMF30TPR in the type 30 SMF record.

Data Type: Integer, read-only

Sample Value: 6

EOS.STARTDATE

The date on which the system began execution of this step

Data Type: Character, read-only

Sample Value: YYYY/MM/DD

EOS.STARTTIME

The time that the system begins execution of this job or started task step. The time value is measured in hundredths of seconds, starting from 12 a.m.

Data Type: Character, read-only

Sample Value: 3600000 (10AM)

EOS.STEPNAME

The step name of the job step that terminated. This value is field SMF30STM in the type 30 SMF record.

Data Type: Character, read-only

Sample Value: LKED

EOS.STEPNUMB

The step number of the job step that terminated. The first step has number 1, and so forth. This value is from field SMF30STN in the type 30 SMF record.

Data Type: Integer, read-only

Sample Value: 2

EOS.SUBSYS

The subsystem name of the job used by SMF for workload accounting. Subsystem names are defined in the SMFPRM*xx* member of parmlib and extracted from the OUCBSUBN field of the OUCB control block.

Data Type: Character, read-only

Sample Value: TSO

EOS.TERMNAME

The symbolic name of the TSO terminal for a TSO session. This value is field SMF30TSN in the type 30 SMF record.

Data Type: Character, read-only

Sample Value: OPSS1

OPSLOG Browse Column: TERMNAME

EOS.TEXT

The OPSLOG message text that describes the end of step event, including the maximum condition code

Data Type: Character, read-only

Sample Value: PAYJOB JOB00123 STEP 1 PAYPROC.STEP1 PGM=P

OPSLOG Browse Column: Text is always displayed.

EOS.USER

An 8-byte variable providing communication between rules executing for the same EOS event. The variable can contain any installation data that these rules need, and it can store a character string displayable through OPSLOG Browse.

Data Type: User-defined, read/write

Notes:

- Before AOF processing, this variable is initialized to binary zeros. It is then passed to each rule that executes for the same EOS event; each rule can look at or change the variable contents before passing the variable to the next rule for the EOS event.
- The primary purpose for the USER variable is to provide a method to pass a small amount of data between the rules. This data may be binary or mixed case. The USER field may also be used for filtering in the OPSLOG. However, USER data used for OPSLOG filtering must be uppercase and displayable.

OPSLOG Browse Column: USER

EOS.USERCOM

The value contained in the JMRUCOM of the JMR control block. This field is sometimes used to point to tables or control blocks used by installation SMF exits.

Data Type: 4-byte binary (unprintable), read-only

Sample Value: '0A002CFC'X

Note: Use the OPSTORE function of OPS/REXX to access any storage pointed to by EOS.USERCOM.

EOS.WORKLOAD

The WLM workload name of the job if the system is using the z/OS Workload Manager for system management. This value is field SMF30WLM in the type 30 SMF record.

Data Type: Character, read-only

Sample Value: PRODCICS

More information:

AOF Rule Tools (see page 29)

Debug an EOS Rule

To debug an EOS Rule

1. Set the CA OPS/MVS BROWSEEOS parameter to YES
2. Set the EOS event profile of your OPSLOG display to Y

   This lets you view all EOS events.
3. With these parameters set, display the OPSLOG EVENT column.

   This lets you see recorded EOS events.

Note: If OPSLOG is not recording EOS events, see Installation Requirements for EOS Rules (see page 172) in this chapter.

More information:

Code and Debug AOF Rules (see page 59)

Example: EOS Rule

Assume that the CICS on-line payroll databases may be reopened after STEP3 of the payroll job, called PAYLAST, completes successfully. An OPS/REXX program called OPENPAYB, running in an OSF server, reopens the databases.

```
)EOS PAYLAST
)PROC
if eos.jobclass = 'P' & ,   /* Only the production jobs */
eos.stepname = 'STEP3' & ,  /* Step3 is the signal      */
eos.condcode = '0000' then  /* Must complete normally   */
  address OSF "OI OPENPAYB"  /* Open the databases      */
)END
```

Global Variable Rules

A global variable (GLV) rule is triggered when the value of an OPS/REXX global variable, whose stem begins with GLVTEMP*x*., GLOBAL., or GLOBAL*x*., is changed. Depending on the logic of your automated CA OPS/MVS applications, you may need to react to the changing of a common OPS/REXX global variable that is being processed by many different OPS/REXX rules or OPS/REXX programs. A GLV rule allows you to centralize this type of processing.

Installation Requirements for GLV Rules

The parameters GLVCHAINMAX and GLVPENDINGMAX prevent the runaway recursion of global variable events. Depending on the logic of your applications that triggers the GLV rules, you can increase the default values of these parameters. To allow updates to CA CCS Common Variable Service (sysplex variables) to trigger the GLV rules, set the GLVNOTIFYRULES parameter to YES.

Note: For more information, see the *Parameter Reference*.

)GLV—Event Specifier of GLV Rules

The following is the format for coding the GLV event definition section:

```
)GLV glvnamespec
```

glvnamespec

Specifies the global variable name. Follow these guidelines when specifying the character string:

- Specify one to 50 characters of the complete GLVTEMP*x* or GLOBAL*x* variable name. The *x* can be a letter from A to Z.
- The string cannot contain embedded blank spaces.
- You can use the wildcard (*) character. For example,
  - GLVTEMPA* matches GLVTEMPA.CICS, GLVTEMPA.IMS.UPTIME, and any other OPS/REXX global variable name with a stem of GLVTEMPA.
  - GLVTEMP*.CICS matches GLVTEMPA.CICS, GLVTEMPB.CICS, and so on.
  - *.CICS matches any GLVTEMP*x* or GLOBAL*x* variable name ending with a tail name of CICS.
  - * alone matches all variables.
- Lowercase letters are acceptable, but the AOF converts them to uppercase for event testing.

Initialization, Processing, and Termination Sections of GLV Rules

The general guidelines for coding the initialization, processing, and termination sections and the various AOF tools that you can use, apply to GLV rules.

More information:

AOF Rule Structure (see page 21)
AOF Rule Tools (see page 29)

RETURN Statements in the)PROC Section of a GLV Rule

The OPS/REXX RETURN statement has no special meaning in the processing section of a global variable rule. The return value has no effect on AOF processing.

Execution Considerations for GLV Rules

The processing section of a rule that responds to a GLV event executes in the address space from which the global variable event originated. Therefore, any type of logic that could possibly suspend the processing of a GLV rule can be performed by scheduling an OPS/REXX program to a CA OPS/MVS OSF TSO, TSL, or TSP server. For more information see, Code and Debug AOF Rules (see page 59).

A change in the value of an OPS/REXX global variable triggers a GLV rule. However, a global variable event does *not* trigger a global variable rule under these conditions:

- Changing the value of a global variable triggers a global variable event unless that variable has a stem of GLOBAL*x*. or GLVTEMP*x*. (where *x* is a number from 0 to 9)
- During CA OPS/MVS startup or shutdown
- When a global variable in the initialization or termination section of the rule changes.
- When a global variable is deleted and the GLVDELETERULES parameter is set to NO, the default value.

If a global variable update occurs in the processing section of a rule, the rule can trigger itself. A global variable rule triggering itself is *not* true recursion. The AOF maintains such global variable events in a FIFO queue. The AOF extracts the events from the queue and triggers other rules when the rule that caused the recursion has finished executing. The GLVPENDINGMAX parameter determines size of the FIFO queue; the default queue size is 100.

To prevent infinite recursion, specify the GLVCHAINMAX parameter to limit the number of global variable events that can occur from in the original rule; the default limit is 1000 events.

Note: For more information about the GLVPENDINGMAX and GLVCHAINMAX parameters, see the *CA OPS/MVS Parameter Reference Guide*.

OPS/REXX Host Environments in the)PROC Section of a GLV Rule

The)PROC section of a GLV rule has the following host environments with the following GLV rule characteristics. Specify the AOFDEFAULTADDRESS parameter for the default host environment for EOS rules.

ADDRESS AOF

Sent to CA OPS/MVS. Does not wait. Output is not returned.

ADDRESS AP

Sent to MSF and then forwarded to the CA Automation Point system. Does not wait. Output is not returned.

ADDRESS EPI

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS HWS

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS ISPEXEC

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS LXCON

The VM and Linux command requests sent to USS server for execution. Does not wait. Output is not returned. The List request runs inline and returns VM and Linux system data in stem variables.

ADDRESS MESSAGE

Sent as a WTO. The AOFDEST parameter specifies the destination.

ADDRESS MQ

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS NETMAN

Sent to a CA OPS/MVS internal CA Netman request queue to issue MGPT commands. Does not wait. Output is not returned.

ADDRESS NETMASTR

Sent to CA NetMaster NM for SNA on the local system. Does not wait. Output is not returned.

ADDRESS OPER

Sent to target console as specified through the OCCONSOLENAME parameters when no CONNAME operands are present. Output is not returned. Schedule an OPS/REXX program in a server when a command output interrogation is needed.

ADDRESS OPSCTL

Sent to a specified facility. If the facility is ECF or OSF, does not wait. If the facility is MSF, slight wait occurs. The external data queue returns the output.

Note: If the command is MSF LIST, no wait occurs.

ADDRESS OPSDYNAM

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS OSF

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSO servers.

ADDRESS OSFTSL

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSL servers.

ADDRESS OSFTSP

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSP servers.

ADDRESS SOF

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS SQL

Does not wait. Processed synchronously for requests that can be satisfied on the local system. Output is returned in stem variable. Error messages, if any, are returned to an external data queue.

ADDRESS SYSVIEWE

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS TSO

Sent to an OSF TSO server. Does not wait. Output is not returned.

ADDRESS USS

Sent to a USS server. Does not wait. Output is not returned. Schedule an OPS/REXX program in a server if a command output interrogation is needed.

ADDRESS WTO

Does not wait. Output is sent to specified (or default) destination. When attempting a WTOR, host command is sent to a TSO server for execution. The response is returned to the server. Schedule an OPS/REXX program in a server if WTOR response interrogation is needed.

More information:

Code and Debug AOF Rules (see page 59)

AOF Variables Available in a GLV Rule

You can use all AOF variable types in GLV rules. You can use the following unique AOF event variables in the)PROC section of a GLV rule. You can also manually interrogate the corresponding OPSLOG display field as an aid in debugging or implementing rule logic.

GLV.FUNCTION

The OPSVALUE function code that caused the GLV rule to execute.

Data Type: Character, read-only

Notes:

- For global variable delete events, the parameter GLVDELETERULES must be set to YES.
- This variable must be used to distinguish between updates and deletes.
- The function codes for deletes are R (remove), 4 (variable mask delete), and 6 (single variable delete).
- For Sysplex variable events, the parameter GLVNOTIFYRULES must be set to YES. The function codes are one of: CREATE, UPDATE, or DELETE.

Sample Value: U

OPSLOG Browse Column: DSPNAME

GLV.JOBNAME

The job name of the address space that originated the GLV event.

Data Type: Character, read-only

Sample Value: OPSPROD

Note: For Sysplex variable events, the value of GLV.JOBNAME is the job name on the originating system on which the variable value change first occurred.

OPSLOG Browse Column: JOBNAME

GLV.MSFID

The Multi-System Facility or CAICCI ID of the system, either local or remote, that caused the global variable event to be invoked by setting a global variable to a value.

Data Type: Character, read-only

Sample Value: SYSA

Note: For Sysplex variable events, the value of GLV.MSFID is always the local system MSF or CCI ID.

OPSLOG Browse Column: MSFID

GLV.NAME

The 1- to 50-byte derived name of the global variable whose modification triggered this event.

Data Type: Character, read-only

Sample Value: GLOBAL.CICS.ACTIVE

OPSLOG Browse Column: Text is always displayed.

GLV.NEWVALUE

The value of the global variable after modification

Data Type: Character, read-only

Notes:

- The standard REXX definitions apply to variables that have never been referenced before or have been dropped.
- This value can or cannot be the current value of the variable, since a subsequent update can have changed its value again.

OPSLOG Browse Column: Text is always displayed.

GLV.OLDVALUE

The value that the global variable had before the global variable event modified it.

Data Type: Character, read-only

Notes:

- The standard REXX definitions apply to variables that have never been referenced before or have been dropped.
- For the Sysplex variables, the OLDVALUE is never provided or always null.

OPSLOG Browse Column: Text is always displayed.

GLV.PROGRAM

The name of the program or rule that triggered the current global variable event rule.

Data Type: Character, read-only

Sample Value: PROD.VTAM

Notes:

- If a REXX program running in a TSO address space triggered the global variable event, GLV.PROGRAM is the member name of the program. If a rule triggered the event, GLV.PROGRAM has the value *ruleset.rule*.
- For Sysplex variable events, the value of GLV.PROGRAM is set to blanks.

OPSLOG Browse Column: TEXT

GLV.SYNA

The name of the system on which the GLV event originated.

Data Type: Character, read-only

Sample Value: PRODS1

Note: For Sysplex variable events, the value of GLV.SYNA is the system name on which the variable value change first occurred.

OPSLOG Browse Column: SYSNAME

GLV.TEXT

The message text as seen in OPSLOG Browse, which is taken from these values of:

- GLV.NAME
- GLV.PROGRAM
- GLV.OLDVALUE
- GLV.NEWVALUE.

Data Type: Character, read-only

Notes:

- CA OPS/MVS truncates GLV.TEXT at 100 characters. So, in some cases this variable can contain only part of the GLV.OLDVALUE value and part or none of the GLV.NEWVALUE value of the variable.
- The GLV.TEXT variable is documented here for completeness. Do not use it for automation.

OPSLOG Browse Column: Text is always displayed.

GLV.USER

An 8-byte variable providing communication between rules executing for the same global variable event. The variable can contain any installation data that these rules need, and it can store a character string displayable through OPSLOG Browse.

Data Type: User-defined, read/write

Notes:

- Before AOF processing, this variable is initialized to binary zeros. This variable is then passed to each rule that executes for the same global variable event. Each rule can look at or can change the variable contents before passing the variable to the next rule for the global variable event.
- The primary purpose for the USER variable is to provide a method to pass a small amount of data between the rules. This data can be binary or mixed case. The USER field can also be used for filtering in the OPSLOG. However, USER data that is used for OPSLOG filtering must be uppercase and displayable.

OPSLOG Browse Column: USER

More information:

AOF Rule Tools (see page 29)

Debug a GLV Rule

To debug a GLV rule

1. Set the CA OPS/MVS BROWSEGLV parameter to YES
2. Set the GLV event profile of your OPSLOG display to Y

   This lets you view all GLV events.
3. With these parameters set, display the OPSLOG EVENT column.

   This lets you see recorded GLV events. This record will show the variable name and its old and new values.

More information:

Code and Debug AOF Rules (see page 59)

Example: GLV Rule

The following is an example of a rule that responds to a global variable event. The rule starts CICS if VTAM is active.

```
)GLV GLOBAL.STATUS.VTAM
)PROC
 if GLOBAL.STATUS.VTAM = 'ACTIVE' then
   do
     address OPER
     "COMMAND(S CICSPROD) NOOUTPUT"
     "COMMAND(S TSO) NOOUTPUT"
   end
```

This example shows a rule that responds to a global sysplex variable event. If VTAM becomes active on a system PLEXA, then this rule starts PLXBPROC on a system PLEXB.

```
)GLV GLVPLXTA.*
)PROC
if GLV.MSFID = ‘PLEXA’ then
 if GLV.NAME = 'GLVPLXTA.VTAM.STATUS' then
   if GLV.OLDVALUE = ‘DOWN’ & GLV.NEWVALUE = 'ACTIVE' then
   do
     address OPER
     "COMMAND(S PLXBPROC) NOOUTPUT"
   end
```

Message Rules

A message (MSG) event occurs when a system component sends a message to a console or to a system log. The AOF recognizes and responds to these types of messages:

- z/OS
- IMS
- CICS (Transient Data Queue messages*)*
- CA 7 Browse Log
- NetView alerts
- Log file directed messages (through the Generic Dataset Interface)
- Application generated WTOs (write-to-operator), WTORs (write-to-operator-with-reply), and WTLs (write-to-log)
- JES2/JES3

Installation Requirements for MSG Rules

Most system messages are broadcast on the subsystem interface (SSI) and no additional installation steps are necessary. However, depending on the product that is issuing the message and how it responds to messages on the SSI, you have to set the initialization parameter SSIMSG to YES. This parameter determines how CA OPS/MVS positions itself on the SSI.

Note: For more information, see the *Parameter Reference*.

The general rule to follow is: If the particular message that you are electing to automate using an MSG rule is not being broadcast on the SSI (not in OPSLOG), an additional CA OPS/MVS installation step is required, as noted on the following page.

IMS Messages

Implementation of the CA OPS/MVS IMS Operator Facility (IOF) lets you create MSG rules against IMS messages that are destined to the IMS log or IMS MTO terminal only-not in OPSLOG or SYSLOG.

Note: For more information, see the *Administration Guide*.

CICS TDQ Messages

Implementation of the CA OPS/MVS CICS Operator Facility (IOF) lets you create MSG rules against messages that are destined to unique CICS Transient Data Queues (TDQ) only-not in OPSLOG or SYSLOG.

Note: For more information, see the *Administration Guide*.

CA 7 Log Messages

Implementation of the CA OPS/MVS CA 7 Browse Log feature lets you create MSG rules against CA 7 messages that are destined to the CA 7 log only-not in OPSLOG or SYSLOG.

Note: For more information, see the *Administration Guide*.

NetView Alert Messages

Installing the CA OPS/MVS NetView Operator Facility (NOF) lets you create MSG rules against NetView alert messages.

Note: For more information, see the *Administration Guide*.

Reissued Messages

The AOFMESSAGES parameter controls whether reissued messages and DOMs are processed by AOF rules. This includes messages and DOMs that originated on another system and were transported and reissued on this system by MSF, CA MIC, or sysplex services.

Log File Directed Messages

You can direct output from data sets such as log files through the CA OPS/MVS generic data set interface (GDI).

Note: For more information, see the *Installation Guide*.

CA OPS/MVS Messages

In general, CA OPS/MVS messages cannot be processed by the AOF. However, CA OPS/MVS messages that have a severity code of O or J are exceptions to this rule; you can write message rules for these CA OPS/MVS messages only. Thus, you can specify an *msgidspec* string for any CA OPS/MVS message having a suffix of O or J. For a list of CA OPS/MVS messages that by default have a severity code of O or J, and information on changing the severity code of messages, see the *Messages Guide*.

)MSG—Event Specifier of MSG Rules

The following is the format for coding the MSG event definition section:

```
)MSG msgidspec  [MLWTO] [NOOPSLOG] [SUPPRESS]
```

msgidspec

Specifies the message ID. Follow these guidelines when specifying the character string:

- Specify one to ten characters of the message ID.
- The string cannot contain embedded blank spaces.

- If the MLWTO keyword is not specified, you can use the wildcard (*) character. For example,
  - IEC* matches IEC234, IECTL56, IEC6705, and any other event identifier containing an IEC prefix.
  - IEC*05 matches IEC05, IECD05, IECDE05, and so on.
  - *05 matches any message ending with 05.* alone matches all messages.
- Lowercase letters are acceptable, but the AOF converts them to uppercase for event testing.

MLWTO

(Optional) To be specified when interrogating a multi-line message and each data line is needed to be stored in stemmed msg.text.n variables for further manipulation. Rule will process after the end-line of the multi-line message is received. Wildcarding is not allowed if utilizing this keyword. Therefore the msgidspec must be that of the complete message id of the multi-line message. For more information see Execution Considerations for Msg Rules (see page 197) for details on processing Multi-line messages.

NOOPSLOG

(Optional) Prevents a message (specified by *msgidspec*) from appearing in the OPSLOG. Specify the NOOPSLOG option before you enable a rule.

SUPPRESS

(Optional) Only valid if the MLWTO optional keyword is utilized, and causes the multi-line message to be suppressed.

More information:

Code and Debug AOF Rules (see page 59)

How to Use the NOOPSLOG Option

You cannot use the NOOPSLOG option to remove *all* message event records from the OPSLOG. For example, specifying MSG * NOOPSLOG causes AOF to ignore the NOOPSLOG option. The NOOPSLOG option is ignored if the *msgidspec* contains an imbedded wildcard (*) character (for example, IST*I). The NOOPSLOG option is acknowledged only for complete *msgidspecs* (for example, IST123I) or for a prefix *msgidspec* (for example, IST*).

You can monitor and control the use of the NOOPSLOG option using these methods:

- To display the current number of NOOPSLOG messages, check the value of the display-only parameter MSGNOOPSLOG.
- To tell the AOF to ignore the NOOPSLOG option, set the AOFNOOPSLOG parameter to NO. The default is YES.
- To determine whether the NOOPSLOG option has been specified in enabled rules, issue the ADDRESS AOF LIST command. For more information about the ADDRESS AOF commands, see the *Command and Function Reference*.

WARNING! Using the NOOPSLOG option in conjunction with a RETURN “DELETE” in the)PROC section eliminates the message from both the OPSLOG and SYSLOG.

Initialization, Processing, and Termination Sections of MSG Rules

MSG rules follow the general guidelines for coding the initialization, processing, and termination sections and the various AOF tools that you can use.

More information:

AOF Rule Structure (see page 21)
AOF Rule Tools (see page 29)

RETURN Statements in the)PROC Section of an MSG Rule

The OPS/REXX RETURN statement specifies the final disposition of a message. The RETURN statement can:

- Allow the operating system to route a message as usual (The message appears both on the console and in the SYSLOG log file).
- Prevent a message from appearing on the console.
- Prevent a message from appearing in the SYSLOG log file.
- Delete a message (Prevent a message from appearing on both, the console and the SYSLOG log file).

Valid values for a RETURN statement in the processing section of a message rule are:

NORMAL

Allows z/OS to route a message as usual.

SUPPRESS

Prevents a message from appearing on the console. The message appears in the system log.

DISPLAY

Prevents a message from appearing in the system log. The message appears on the console.

DELETE

Suppresses a message entirely. The message appears neither on the console nor in the SYSLOG log file.

FORCENORMAL

Allows z/OS to route a message normally, but overrides any return value that was specified by a prior rule.

The default is RETURN NORMAL. The return values listed here are character *constants* rather than keywords. An unrecognized return value (for example, a misspelled value), defaults to a value of NORMAL.

Other RETURN Statement Considerations

Consider the following when specifying the RETURN statement in the processing section of a message rule:

- If multiple rules respond to a single message event, the AOF uses the highest-precedence return value; the order of precedence is:
  - DELETE (highest precedence)
  - DISPLAY
  - SUPPRESS
  - NORMAL (lowest precedence)
- The DELETE and DISPLAY return values work as described only if the AOFDELETE parameter is set to YES (the default setting). If the AOFDELETE parameter is set to NO, the rule processes a message as though the DELETE return value is SUPPRESS and the DISPLAY return value is NORMAL.

For information about the AOFDELETE parameter, see the *Parameter Reference*.

Execution Considerations for MSG Rules

The processing section of a rule that responds to a message event executes in the address space from where the message originated. Therefore, any type of logic that could possibly suspend the processing of an MSG rule must be performed by scheduling an OPS/REXX program to a CA OPS/MVS OSF TSO, TSL, or TSP server. For more information, see Code and Debug AOF Rules (see page 59).

MSG rules execute on both single-line messages (WTOs) and multiple line messages (MLWTOs). Single-line WTOs are processed once, meaning they enter the rule, and then exit. A)MSG rule with the MLWTO option specified executes upon the issuing of the end-line of the multiline message.

Note: While processing a multi-line message, if the MLWTO keyword is omitted, the rule logic executes for each line of the MLWTO. Meaning, that the primary line is processed first by the rule logic, and then each subsequent data line and the end-line is processed. Thus a four line multi-line message would cause the rule to execute four times.

Adhere to the following guidelines when attempting to perform automation on true MLWTOs:

1. Using the OPSFLAGS display field in the OPSLOG, verify that the message is a true MLWTO (primary line, data lines, end-line). Some applications internally issue multiple single WTOs, making them appear as one MLWTO. For details on this field, see the discussion of the event variable MSG.FLAGS in AOF Variables Available in an MSG Rule. Single-line WTOs have an eight in the first byte of this field and MLWTOs have a 2 or a 3 in the first byte.

2. Identify the complete message specifier for the)MSG rule. To see exactly what CA OPS/MVS sees as this message ID, display the MSGID column in the OPSLOG for the MLWTO. Specify the complete message ID. Wildcarding is not allowed when using the MLWTO keyword. While processing a multi-line message, if the MLWTO keyword is omitted, the rule logic executes for each line of the MLWTO. Meaning, that the primary line is processed first by the rule logic, and then each subsequent data line and the end-line is processed. Thus a four line multi-line message would cause the rule to fire four times.

   ```
   )MSG IST663I MLWTO or )MSG IST663I MLWTO SUPPRESS
   ```

   Add the SUPPRESS keyword if the suppression of the MLWTO is desired.

   Note: If you are attempting to modify the route or descriptor codes of a multi-line message, then this logic cannot be accomplished when specifying the MLWTO optional keyword. This is because the MLWTO keyword causes the rule to process after the end-line has been issued, and disposition alteration logic (change route or desc codes) must be performed at the time the primary line of the multi-line message is issued.

3. The following MSG event variables are available to manipulate multiline messages when the MLWTO optional keyword is included on the message specifier:

   msg.text.0

   Shows the number of lines available in the MLWTO.

   msg.text.n

   Text of individual lines. For example, msg.text.1 is the first line, msg.text.2 is the second line.

   msg.linetype.0

   Shows number of lines available in the MLWTO.

   msg.linetype.n

   The line type of each line in the MLWTO. For example, msg.linetype.1 is the line type of the first line, msg.linetype.2 is the line type of the second line.

   C

   Control line

   L

   Label line

   D

   Data line

   E

   End line

   DE

   Data and end line

   Suppose you have the following VTAM multi-line message:

```
IST663I CDINIT REQUEST FROM A55X99 FAILED, SENSE=08570002
IST664I REAL OLU=USILDA02.A13IOML0 REAL DLU=USILDA01
IST314I END
```

   Using this test rule to dump these environmental variables for this message:

```
)MSG IST663I MLWTO SUPPRESS
)Proc
say '**Total Number of lines in message='msg.text.0
do l = 1 to msg.text.0
say ' **Line 'l' contents =>'msg.text.l
say ' **Line type of line 'l'=>'msg.linetype.l
do w = 1 to WORDS(msg.text.l)
say ' **Word 'w 'of line 'l' =>'WORD(msg.text.l,w)
end
say ''
end
```

Resulting SAY output from test rule:

```
*Total Number of lines in message=3
**Line 1 contents =>IST663I CDINIT REQUEST
                 FROM A55X99 FAILED, SENSE=08570002
**Line type of line 1=>D
**Word 1 of line 1 =>IST663I
**Word 2 of line 1 =>CDINIT
**Word 3 of line 1 =>REQUEST
**Word 4 of line 1 =>FROM
**Word 5 of line 1 =>A55X99
**Word 6 of line 1 =>FAILED,
**Word 7 of line 1 =>SENSE=08570002


**Line 2 contents =>IST664I REAL OLU=USILDA02.A13OML0
                 REAL DLU=USILDA01
**Line type of line 2=>D
**Word 1 of line 2 =>IST664I
**Word 2 of line 2 =>REAL
**Word 3 of line 2 =>OLU=USILDA02.A13IOML0
**Word 4 of line 2 =>REAL
**Word 5 of line 2 =>DLU=USILDA01


**Line 3 contents =>IST314I END
**Line type of line 3=>DE
**Word 1 of line 3 =>IST314I
**Word 2 of line 3 =>END
```

4. If logic is needed to pass the complete contents of a multi-line message to an OPS/REXX program in order to perform asynchronous processing that cannot be performed in an MSG rule. Then the corresponding msg.text.n variables can be stored in unique GLVTEMP variables and then obtained within the OPS/REXX program. For example, writing the message to some data set. Refer to the sample rule member MLWTO of the opsmvshlq.CCLXRULS data set for specific coding details on passing the contents of a MLWTO to an OPS/REXX program.

   For more information, about a specific sample rule that utilizes the MLWTO option in addition to the MLWTOx samples of the opsmvshlq.CCLXRULS library, see MSG Rules Examples (see page 236).

   These samples outline various coding techniques that can be used to process MLWTOs including altering message disposition.

OPS/REXX Host Environments in the)PROC Section of an MSG Rule

The)PROC section of an MSG rule has the following host environments with the following MSG rule characteristics. The AOFDEFAULTADDRESS parameter specifies the default host environment for MSG rules.

ADDRESS AOF

Sent to CA OPS/MVS. Does not wait. Output is not returned.

ADDRESS AP

Sent to MSF and then forwarded to the CA Automation Point system. Does not wait. Output is not returned.

ADDRESS EPI

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS HWS

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS ISPEXEC

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS LXCON

The VM and Linux command requests sent to USS server for execution. Does not wait. Output is not returned. The List request runs inline and returns VM and Linux system data in stem variables.

ADDRESS MESSAGE

Sent as a WTO to the destination the AOFDEST parameter specifies.

ADDRESS MQ

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS NETMAN

Sent to a CA OPS/MVS internal CA Netman request queue to issue MGPT commands. Does not wait. Output is not returned.

ADDRESS NETMASTR

Sent to CA NetMaster NM for SNA on the local system. Does not wait. Output is not returned.

ADDRESS OPER

Sent to target console as specified through the OCCONSOLENAME parameters when no CONNAME operands are present. Output is not returned. Schedule an OPS/REXX program in a server when a command output interrogation is needed.

ADDRESS OPSCTL

Sent to a specified facility. If the facility is ECF or OSF, does not wait. If the facility is MSF, slight wait occurs. External data queue returns the output.

Note: If the command is MSF LIST, no wait occurs.

ADDRESS OPSDYNAM

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS OSF

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSO servers.

ADDRESS OSFTSL

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSL servers.

ADDRESS OSFTSP

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSP servers.

ADDRESS SOF

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS SQL

Does not wait. Processed synchronously for requests that can be satisfied on the local system. Output is returned in stem variable. Error messages, if any, are returned to an external data queue.

ADDRESS SYSVIEWE

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS TSO

Sent to an OSF TSO server. Does not wait. Output is not returned.

ADDRESS USS

Sent to a USS server. Does not wait. Output is not returned. Schedule an OPS/REXX program in a server if the command output interrogation is needed.

ADDRESS WTO

Does not wait. Output is sent to specified (or default) destination. When attempting a WTOR, host command is sent to a TSO server for execution. The response is then returned to the server. Schedule an OPS/REXX program in a server if the WTOR response interrogation is needed.

More information:

Code and Debug AOF Rules (see page 59)

AOF Variables Available in MSG Rules

The following unique AOF event variables appear in the)PROC section of an MSG rule. Use all variables as an aid in debugging or implementing rule logic. The corresponding OPSLOG display fields are listed. For more information, see AOF Rule Tools (see page 29).

MSG.AMRF

Displays a value indicating whether the message is retained in the Action Message Retention Facility (AMRF).

Data Type

Integer, read/write

Values

0

Do not retain this message.

1

Retain this message.

Example

1

Notes:

- A message must have one of the AMRF descriptor codes in order to be considered for AMRF. See the OPSBITS OPS/REXX function for the following AMRF eligible descriptor code values: IMEDACTN, EVENACTN, CRITEVET. Unless the message has one of these descriptor codes, setting MSG.AMRF to 1 has no significant effect on the AMRF retention status of the message. For more information about AMRF eligible descriptor codes values, see the OPSBIT OPS/REXX function in the CA OPS/MVS Command and Reference Guide.
- For messages originally issued with any of the AMRF descriptor codes, setting MSG.AMRF to 0 prevents AMRF from retaining the message. Even when the descriptor codes are not changed in the AOF rule.
- Suppressing or deleting a message using the AOF rule RETURN statement has no effect on the AMRF retention status of a message.

Example

The following two lines of code in an MSG rule results in the AMRF retaining the message:

```
MSG.DESC = OPSBITS("IMEDACTN")
MSG.AMRF=1
```

OPSLOG Browse Column: Fourth bit of AFLAGS

MSG.AUTOFLAG

The NetView automation flag, which is taken from the WQESAUT bit flag. In addition, the WQEAUTO bit flag is also used.

Data Type

Integer, read/write

Example

1, if automation is specified.

Notes:

- Modifying the MSG.AUTOFLAG variable is equivalent to dynamically modifying the AUTO(YES|NO) parameter in the appropriate MPFLSTxx member of the Logical Parmlib Concatenation. For more information, see the IBM documentation.
- Modifying this variable determines whether NetView processes the message. Using a rule, you can change the value of the variable to 0 (equivalent to AUTO(NO)) or non-zero (equivalent to AUTO(YES)).

OPSLOG Browse Column: The high-order bit in the AFLAGS (AF) column

MSG.AUTOTOKN

The NetView automation token, which is taken from the WQEAUTOT flag in the WQE.

Data Type

8-byte character, read/write

Example

Parm1

Notes:

- Modifying the MSG.AUTOTOKN variable is equivalent to modifying the AUTO(token) parameter in the appropriate MPFLSTxx member of the Logical Parmlib Concatenation. For more information, see the IBM documentation.
- Only the first 8 bytes of the modified value are passed to NetView.
- When setting fewer than 8 bytes of the value, the remainder of the 8-byte field is padded with nulls.

OPSLOG Browse Column: AUTOTOKN

MSG.BEWTO

A value indicating whether the current message was issued as a branch entry WTO.

Data Type

Integer, read-only

Values

0

Message is not a branch entry WTO.

1

Message is a branch entry WTO.

Usage Notes:

This variable can be used to determine whether MSG.JOBNAME or MSG.OJOBNAME should be used in your automation procedure.

Example

The following code can be used to determine which variable contains the appropriate JOBNAME:

```
IF MSG.BEWTO = 1 THEN
  jobname = MSG.OJOBNAME
ELSE
  jobname = MSG.JOBNAME
```

MSG.CMDRESPONSE

A value indicating whether the current message is a command response.

Data Type

Integer, read-only

Values

0

Message is not a command response.

1

Message is a command response

Example

0

OPSLOG Browse Column: Third bit of AFLAGS

MSG.COLOR

The color that the message has in OPSLOG Browse and, optionally, on the console.

Data Type

1-byte binary (unprintable), read/write

Note: Write takes effect only if the value of the PROPAGATEATTR parameter is YES.

Example

'00'X

Notes:

- Use the OPSCOLOR function of OPS/REXX to set the MSG.COLOR variable.
- IMS messages use the colors set in IMSnCOLOR parameters.
- MCS descriptor codes 1 and 11 generate red messages; code 2 generates a white message.
- When using multiple rules, set the MSG.COLOR variable for the same message, CA OPS/MVS uses the last value set. To determine the color of each rule set, set the CA OPS/MVS RULETRACE parameter to ON. For a description of the RULETRACE parameter, see the *CA OPS/MVS Parameter Reference*.
- For messages from the generic data set interface, MSG.COLOR contains the color that is specified with the optional third parameter in the SUBSYS JCL keyword. If you do not specify this parameter, the variable contains the text string NONE.

OPSLOG Browse Column: COLOR

MSG.CONSNAME

The console name that is used to get messages from the subsystem interface:

- For messages originating in the subsystem interface, MSG.CONSNAME contains the console name.
- For messages originating in the JES3 IATUX31 exit, MSG.CONSNAME contains the JES3 DSP name.
- For messages originating in the generic data set interface or in the OMEGAMON interface, MSG.CONSNAME contains the report ID.

Data Type

Character, read/write

Example

MSTCONS

OPSLOG Browse Column: CONSNAME

MSG.CONTROLLN

A value indicating whether the current message is a control line in a multiline message (MLWTO).

Data Type

Character, read-only

Values

0

Not a control line.

1

A control line.

Example

1

OPSLOG Browse Column: Fourth bit of OPSFLAGS

MSG.DATALN

A value indicating whether the current message is a data line in a multiline message (MLWTO).

Data Type

Character, read-only

Values

0

Not a data line.

1

A data line.

Example

1

OPSLOG Browse Column: Sixth bit of OPSFLAGS

MSG.DATESTMP

The date when the message was issued.

Data Type

Character, read-only

Example

20050908

Notes:

- CA OPS/MVS sets this variable only for messages where OPSINFO("EXITTYPE") returns a value of either MVS or NIP.
- The format of the date value is the same as the format for the REXX standard date function - DATE("S") - which is yyyymmdd.
- MSG.DATESTMP can return a date one day earlier than the REXX DATE("S") function in a message rule. This early date occurs when a message is issued a fraction of a second before midnight. Any such difference reflects the difference in time between when the operating system constructed the message control block and when CA OPS/MVS intercepted the message.

Recommended

Use the REXX DATE("S") function because that date always matches the date for the message in OPSLOG Browse. MSG.DATESTMP is provided for completeness.

MSG.DESC

The descriptor codes of the message, which are taken from the WQEDESCD field of the WQE control block.

Data Type

2-byte binary (unprintable), read/write

Example

'0400'X

Note: Use the OPSBITS function of OPS/REXX to set the MSG.DESC variable. For a description of message descriptor codes, see the IBM documentation.

OPSLOG Browse Column: ROUTE or ROUTEX

MSG.DISP

The current disposition of the message that is set previously in executed rules. For example, SUPPRESS.

Data Type

Integer, read-only

Example

4

Note: CA OPS/MVS sets this variable automatically after each rule processes a message; this variable cannot be set manually. The next rule always gets the highest return code that is set in all of the previous rules. The following list shows the correspondence between the MSG.DISP value and the AOF RETURN values:

0

NORMAL

4

SUPPRESS

8

DISPLAY

12

DELETE

OPSLOG Browse Column: DISP

Note: This column uses descriptive character strings to display the final, highest return code.

MSG.ENDLN

A value indicating whether the current message is the end line in a multiline message (MLWTO).

Data Type

Character, read-only

Values

0

Not an end line.

1

An end line.

Example

1

OPSLOG Browse Column: Seventh bit of OPSFLAGS

MSG.FLAGS

The message type is taken from CA OPS/MVS message flags. For example, MLWTO.

Data Type

2-byte binary (unprintable), read-only

Example

'0010'X

Message Flags

Many of the bits represented in MSG.FLAGS are also represented by bit-specific MSG variables. We recommend using the bit-specific variables wherever possible. In all cases, the bit-specific variable takes on a value of 1 or 0 to show whether the corresponding bit is set.

'8000'X

Single-line message flag.

Single-bit equivalent: MSG.SINGLELN

'4000'X

WTOR message flag.

Single-bit equivalent: MSG.WTOR

'2000'X

Multiline message (MLWTO) flag.

Single-bit equivalent: MSG.MULTILN

'1000'X

Control line of an MLWTO flag.

Single-bit equivalent: MSG.CONTROLLN

'0800'X

Label line of an MLWTO flag.

Single-bit equivalent: MSG.LABELLN

'0400'X

Data line of an MLWTO flag.

Single-bit equivalent: MSG.DATALN

'0200'X

End line of an MLWTO flag. See note.

Single-bit equivalent: MSG.ENDLN

'0100'X

Last command output message flag. CA OPS/MVS sets this variable to mark the last output line.

'0080'X

Urgent attention message flag. Set when the message has descriptor code 1 or 11 set.

Single-bit equivalent: MSG.URGENT

'0040'X

Immediate action message flag, Set when the message has descriptor code 2 is set or is a WTOR.

Single-bit equivalent: MSG.IMMEDACT

'0020'X

Message is from a local JES3.

'0010'X

Current message is a z/OS (WTO/SVC 35) message from a global JES3.

'0002'X

Current message is a WTL (SVC 36) rather than a WTO. CA OPS/MVS intercepts WTL messages only if its SSIWTL parameter is set to YES.

'0008'X

Current message is from an authorized task.

'0004'X

Current message was originally issued on a different system and is now being reissued on the current system.

Single-bit equivalent: MSG.REISSUE

'0001'X

MPF has suppressed the current message.

Single-bit equivalent: MSG.MPFSUPP

Note: The '0200'X bit is set for the last line of an MLWTO, and usually marks the end of output from a single command. However, HSM, JES2, and other products can issue multiple MLWTOs in response to a single command, so this bit is not a reliable indication that output has been completed.

OPSLOG Browse Column: OPSFLAGS

MSG.FULLTEXT

The COF feature of the product generates message rule events from the CICS Transient Data Queue. This variable contains up to 256 characters of the TD queue message text. When the MSG.TEXT variable is reworded, it replaces the MSG.FULLTEXT value once the message event is fully processed by the message rules. Because the maximum length of MSG.TEXT is shorter than MSG.FULLTEXT, truncation of the original message text value can occur. For all other message rule event types, this variable is the same as MSG.TEXT.

Da**ta Type**

256-byte character, read-only

Example

```
DFHTD0101I applid Transient Data initialization has ended.
```

MSG.ID

The message identifier, usually the first token, or the first blank delimited word of the message text. This variable has a maximum length of ten characters. If this token exceeds ten characters in length, this variable contains the leftmost ten characters of the token.

Data Type

Character, read-only

Example

IEF125I

Note: The MSG.ID variable value determines which message rules execute for the current message event. This variable never contains any special screen characters or leading or trailing blanks.

OPSLOG Browse Column: MSGID

MSG.IMMEDACT

A value indicating whether the current message is an immediate action message. An immediate action message is one that has descriptor code 2 set.

Data Type

Character, read-only

Values

0

Not an urgent attention message.

1

An urgent attention message.

Example

1

OPSLOG Browse Column: Tenth bit of OPSFLAGS

MSG.IMSID

The IMS ID of the associated IMS control region; or, for non-IMS messages, the value of the IMSNONE parameter. The default for the IMSNONE parameter is NONE.

Data Type

Character, read-only

Example

IMSA

Notes:

- The IMS ID is set for all z/OS messages that any IMS-related address space issues; for example, messages from the IMS control and message processing regions.
- The MSG.IMSID variable is available only when the IOF is licensed, installed, and active at your site. The INITIMS parameter must be YES.

OPSLOG Browse Column: IMSID

MSG.JOBID

The identifier that JES2 or JES3 assigned to the message issuer.

Data Type

Character, read-only

Examples

- T12345 for a TSO user
- J12345 for a job
- S12345 for a started task
- ACF2 for a started task that is started with SUB=MSTR.

When JOBIDs greater than 100,000 are supported and activated in z/OS, the examples are in the format T0012345, J0012345, and S0012345.

Note: The contents of this variable depend on where the current address space was created:

- When the address space was created using JES2 or JES3 and does not have a job ID, the value of the variable is a modified job identifier. The job identifier begins with a character identifying the address space type: T for TSO, J for a batch job, S for a started task.
- When the address space was created using the MSTR subsystem, the variable value is the first five characters of the job name.

This value can be different than the value OPSINFO('JOBID') returns when used in a message rule. MSG.JOBID contains the job ID explicitly specified by the message issuer or determined by z/OS. Messages that are issued by JES2 in response to JES2 commands specify the job ID of the address space to which the message relates rather than the job ID of JES2. When the WQE does not contain an explicit job ID. CA OPS/MVS uses the job ID of the address space that issued the message.

OPSLOG Browse Column: JOBID

MSG.JOBLOGSUP

A value indicating whether the message should appear in the JES job log of the job that issued the message or on whose behalf the message was issued.

Data Type

Integer, read/write

Values

0

Display the message in the JES job log.

1

Suppress the message from the JES job log.

Example

1

Notes:

- For example, use the OPSPRM function to change the severity of message OPS1000 to J. Keep the result of SAY statements from rules out of the JES job log by setting the MSG.JOBLOGSUP variable to 1 in a message rule for OPS1000J.
- The value of the MSG.JOBLOGSUP variable takes effect only if the value of the SSIMSG parameter is YES.

OPSLOG Browse Column: Sixth bit of AFLAGS

MSG.JOBNAME

The job name of the message issuer, which is taken from:

- The WQE, for IMS or z/OS messages that are reissues of messages that originated on another system.
- Data areas the ASCB points to, for all other IMS and z/OS messages.
- The message text prefix areas, for most JES3 messages.

Note: The MSG.JOBNAME value is blank for some JES3 messages.

Data Type

Character, read-only

Example

VTAM

OPSLOG Browse Column: JOBNAME

MSG.JOBNM

The name of the job that is associated with this line of output, which is taken from the WQEJOBNM field of the WQE.

Data Type

Character, read-only

Example

VTAM

Notes:

- Set only for z/OS messages, the MSG.JOBNM variable contains the data that is displayed on an MCS console using the MFORM=J keyword of the Control command. For more information, see the IBM documentation.
- Usually, the MSG.JOBNM value is a job name string, but is set to a job number for output from some JES2 commands. For example, MSG.JOBNM contains job numbers for each output line from the JES2 command $DA.
- When the CA OPS/MVS SSIMSG parameter is set to YES at startup. JES2 does not put a job name in MSG.JOBNM because this parameter setting dictates that <projectname> receives messages before the subsystem.

OPSLOG Browse Column: JOBNM

MSG.LABELLN

A value indicating whether the current message is a label line in a multiline message (MLWTO).

Data Type

Character, read-only

Values

0

Not a label line.

1

A label line.

Example

1

OPSLOG Browse Column: Fifth bit of OPSFLAGS

MSG.LINETYPE.0

Shows number of lines available in the MLWTO. Only valid if the MLWTO option is specified on the message specifier.

Data Type

Character, read-only

MSG.LINETYPE. n

The line type of each line in the MLWTO (msg.linetype.1 is the line type of the first line, msg.linetype.2 is the line type of the second line,etc). Only valid if the MLWTO option is specified on the message specifier.

Data Type

Character, read-only

Values

C

Control line

L

Label line

D

Data line

E

End line

DE

Data and end line

MSG.MCSFG

Internal z/OS, IMS, or MCS message flags, which are taken from:

- z/OS messages-The WQEMCSF field of WQE
- JES3 messages-The MSGMASK field of MESSAGE macro PLIST
- IMS messages-2 flag bytes of UEHB or DFSAOE0, and the 1-byte entry code
  - UEHBFLG1/UEHBFLG2 for IMS DFSAOUE0 (type 1) exit
  - AOE0FLG1/AOE0FLG2 for IMS DFSAOE00 (type 2) exit
- COF messages-The CICS start code.

Data Type

3-byte binary (unprintable), read-only

Example

'0E0000'X

Note: For information about WQE flags, see the IBM documentation.

OPSLOG Browse Column: FLAGS

MSG.MIC

A value indicating whether the current message is a message that was imported and reissued on this system by CA MIC.

Data Type

Character, read-only

Values

0

CA MIC did not reissue this message.

1

CA MIC reissued this message.

Example

1

Note: When the AOFMESSAGES parameter is set to MVSGLOBAL, CA MIC reissued messages are eligible for processing by the AOF. Even response messages which are returned to this system as a result of a cross-system command that is issued on this system through CA MIC.

MSG.MLWTOMIN

When the message is a minor line of an MLWTO message, the value is 1. When the message is the major line of an MLWTO message, or the only line of a WTO message, the variable 0.

Data Type

Character, read-only

OPSLOG Browse Column: Second bit of AFLAGS

MSG.MPFSUPP

A value indicating whether the z/OS Message Processing Facility (MPF) has suppressed the current message.

Data Type

Character, read-only

Values

0

MRF did not suppress this message.

1

MPF suppressed this message.

Example

1

Note: MPF processes messages before they are passed to the subsystem interface where CA OPS/MVS processes them.

OPSLOG Browse Column: 16th bit of OPSFLAGS

MSG.MSFID

The system ID of the system where the message originated, supplied by the CA OPS/MVS Multi-System Facility (MSF).

Data Type

Character, read-only

Example

SYSA

Note: The MSF ID of a message is the local SYSID, for a message that is created on the current system. For a remote system, the MSFID is the SYSID of that system.

OPSLOG Browse Column: MSFID

MSG.MULTILN

A value indicating whether the current message is a part of a multiline message (MLWTO).

Data Type

Character, read-only

Values

0

Not part of a multiline message.

1

Part of a multiline message.

Example

1

Third bit of OPSFLAGS

MSG.OASID

The original ASID associated with the message.

When the message is:

- Issued asynchronously from the CONSOLE address space
- Reissued for some reason

The value differs from OPSINFO("ASID") and represents the ASID in which the message was intercepted.

In the following cases, the value of MSG.OASID is always the same as OPSINFO("ASID"):

- AOFTEST
- The EXITTYPE is not MVS

Data Type

2-byte binary (unprintable), read-only

Example

'003E'X

Notes:

- Use the C2X function of OPS/REXX to convert the value of this variable to hexadecimal characters.
- Use MSG.ASID to determine the ASID from which a message was originally issued.

MSG.ODESC

The original descriptor codes that are assigned to a message.

Data Type

2-byte binary (unprintable), read-only

Example

'0400'X

Note: The MSG.ODESC variable is equivalent to the Automate &DESC environmental variable.

MSG.OJOBNAME

The original job name that is associated with the message.

When the message is:

- Issued asynchronously from the CONSOLE address space
- Reissued for some reason

This value differs from MSG.JOBNAME.

In the following cases, the value of MSG.OJOBNAME is always the same as MSG.JOBNAME:

- AOFTEST
- The EXITTYPE is not MVS

Data Type

Character, read-only

Example

VTAM

MSG.OMAJORTEXT

The *original* message text. The value of MSG.OMAJORTEXT does not change from a rule to a rule, even if a rule rewords the message text.

Data Type

Character, read-only

Values

For a WTO message, the value is the original message text; for an MLWTO message, the value is the possibly updated text from the first line of the message.

Notes:

- The MSG.OMAJORTEXT variable is equivalent to the Automate &MSG environmental variable.
- For example, if there are two MSG rules processing a message the first MSG rule modifies MSG.TEXT, the second rule sees the changed MSG.TEXT but MSG.OMAJORTEXT remains unchanged and contains the original text of the message. When an MSG rule changes the text of the first line of an MLWTO, rules for the minor lines see the changed text in MSG.OMAJORTEXT.

MSG.OROUTE

The original routing codes that are assigned to a message.

Data Type

16-byte binary (unprintable), read-only

Example

'10000000000000000000000000000000'X

Note: The MSG.OROUTE variable is equivalent to the Automate &ROUTCDE environmental variable.

MSG.OTEXT

The complete text of a *secondary* line of a multiline WTO message; this *value* is null for a message that is not a multiline WTO. The secondary line is a line other than the first or primary line.

Data Type

Character, read-only

Note: The MSG.OTEXT variable is equivalent to the Automate &MLMSG environmental variable.

MSG.REISSUE

A value indicating whether the current message is a message that was originally issued on another system and has been transported and then reissued on this system.

Data Type

Character, read-only

Values

0

Not a reissued message.

1

A reissued message.

Example

1

Note: Messages can be reissued as a result of sysplex console processing, MSF, CA MIC, and possibly other software products. Reissued messages are only eligible to be processed by the AOF when the AOFMESSAGES parameter is set to MVSGLOBAL.

14th bit of OPSFLAGS

MSG.REPLYID

The reply number that is associated with a WTOR message, which is taken from the OREID field of the ORE.

Data Type

Character, read-only

Example

01

Notes:

- This variable is valid only for WTORs.
- The MSG.REPLYID variable field is the defined size by RMAX in the DEFAULT statement of the appropriate CONSOLxx member, which is in the Logical Parmlib Concatenation.

OPSLOG Browse Column: First part of the text field when the message is a WTOR.

MSG.REPORTID

The report ID associated with the message. If the message comes from the generic data set interface, MSG.REPORTID contains the subsystem report ID parameter.

Data Type

Character, read-only

Example

PAYLOG1

OPSLOG Browse Column: DSPNAME

MSG.ROUTE

The routing codes of the message, that is taken from the WQE control block.

Data Type

16-byte binary (unprintable), read/write

Example

'10000000000000000000000000000000'X

Notes:

Use the OPSBITS function of OPS/REXX to set this variable.

For a description of routing codes, see the IBM documentation.

OPSLOG Browse Column: ROUTE or ROUTEX

MSG.SINGLELN

A value indicating whether the current message is a single-line message.

Data Type

Character, read-only

Values

0

Not a single-line message.

1

A single-line message.

Example

1

OPSLOG Browse Column: First bit of OPSFLAGS

MSG.SPCHR

The message special screen character. For example, the + preceding problem CA OPS/MVS messages.

Data Type

Character, read-only

Example

+

Note: This variable often is blank. For more information about special screen characters, see the IBM documentation.

OPSLOG Browse Column: SPECIAL

MSG.SUBSMOD

A value indicating whether the current message is eligible for subsystem modification. The value of the SUBSMOD parameter that is specified for the WTO message determines message eligibility.

Data Type

Integer, read-only

Values

0

Message is not eligible for subsystem modification.

1

Message is eligible for subsystem modification.

Example

1

OPSLOG Browse Column: Fifth bit of AFLAGS

MSG.SYNA

The system name of the system issuing the message.

Data Type

Character, read-only

Example

MVS34

Notes:

The system name is derived from the SYSNAME parameter specified in the appropriate IEASYSxx member of the Logical Parmlib Concatenation.

For messages imported through the CA MIC product, this variable contains the name of the system from which the message originated.

OPSLOG Browse Column: SYSNAME or SYNA

MSG.SYSID

The system ID of the system where the message was issued, usually the SMF ID. For JES3 messages, the SYSID value derives from the MPNAME field of the Active Main Processor Control Table. For JES2 messages, the SYSID value derives from the SMF ID string.

Data Type

Character, read-only

Example

S000

Note: The OPSLOG Browse column displays two characters of this variable, not the complete field. The CA OPS/MVS BROWSEIDFORMAT parameter determines which characters are displayed. For a description of the BROWSEIDFORMAT parameter, see the *CA OPS/MVS Parameter Reference*.

OPSLOG Browse Column: SYSID

MSG.TERMNAME

The name of the terminal that is associated with the message:

- For z/OS messages, the variable has the value NONE unless the message originated from a TSO address space.
- For IMS messages, the variable contains the name of the IMS LTERM.
- For JES3 messages, the variable contains the JES3 console name.
- For COF messages, the variable contains the CICS intra-partition queue name.
- For messages from the generic data set interface, the variable contains the ddname that is related to the subsystem data set from which the message was obtained.

Data Type

Character, read-only

Example

OPSS1

OPSLOG Browse Column: TERMNAME

MSG.TEXT

The message text:

- For z/OS messages, which are taken from the WQE text field and excludes special characters and blanks.
- For JES3 messages, which are taken from a parameter list that is passed to the IATUX31 exit.
- For IMS messages, which are taken from the AOI exit parameter list.

Data Type

Character, read/write

Example

IEF125I USERA - LOGGED ON - TIME=06.49.05

OPSLOG Browse Column: Text is always displayed.

MSG.TEXT.0

Shows the number of lines available in a multiline message. Only valid if the MLWTO option is specified on the message specifier.

Data Type

Character, read only

MSG.TEXT.n

Text of each data line of a multiline message. Only valid if the MLWTO option is specified on the message specifier.

Data Type

Character, read only

MSG.TIMESTMP

The time when the message was issued, as it would appear on an MCS console through MFORM=T.

Data Type

Character, read-only

Example

12.43.14

Notes:

- CA OPS/MVS sets this variable only for messages where OPSINFO("EXITTYPE") returns a value of either MVS or NIP.
- The format of the timestamp value (hh.mm.ss) is slightly different from the format for the REXX normal time function - TIME("N") - which is hh:mm:ss. The timestamp format uses a period as a delimiter, whereas the TIME function uses a colon as a delimiter.
- This variable is related to the MSG.DATESTMP variable in that it represents the point of view of the operating system as to the date and time the message was issued. MSG.TIMESTMP can return a time slightly earlier than the REXX TIME("N") function in a message rule. Any such difference reflects the difference in time between when the operating system constructed the message control block and when CA OPS/MVS intercepted the message.

Recommended

Use the REXX TIME("N") function because that time always matches the time for the message in OPSLOG Browse. MSG.TIMESTMP is provided for completeness.

OPSLOG Browse Column: TIMESTMP

MSG.TOKEN

The variable token set when the MSF receives the message and the originating system specified OPSSEND('W' ,...).

Data Type

4-byte binary (unprintable), read-only

Example

'00004C94'

Notes:

- MSG.TOKEN has the same value as the MSG.WTOID variable on the originating system.
- Using MSG.TOKEN, a DOM rule on the receiving system can translate the DOM.WTOID variable by comparing its value to that of MSG.TOKEN. When the DOM.WTOID and MSG.TOKEN values match, the DOM has been issued for MSG.WTOID.

OPSLOG Browse Column: TOKEN

MSG.URGENT

A value indicating whether the current message is an urgent attention message. An urgent attention message is one that has descriptor codes 1 or 11 set.

Data Type

Character, read-only

Values

0

Not an urgent attention message.

1

An urgent attention message.

Example

1

OPSLOG Browse Column: Ninth bit of OPSFLAGS

MSG.USER

An 8-byte variable providing communication between rules that execute for the same message event. The variable can contain any installation data that these rules need, and it can store a character string displayable through OPSLOG Browse.

Data Type

User-defined, read/write

Notes:

- Before AOF processing, this variable is initialized to binary zeros. The variable is then passed to each rule that executes for the same message event. Each rule can look at or can change the variable contents before passing the variable to the next rule for the message event.
- The primary purpose for the USER variable is to provide a method to pass a small amount of data between the rules. This data can be binary or mixed case. The USER field can also be used for filtering in the OPSLOG. However, USER data that is used for OPSLOG filtering must be uppercase and displayable.

OPSLOG Browse Column: USER

MSG.USERID

The user ID for the security product on your system. This value is usually the CA ACF2 user ID from the ACFASVT or the RACF user ID from the current ACEE. For IMS, the user ID from the CTB is valid only if IMS is generated with enhanced security.

Data Type

Character, read-only

Example

TSOID01

Note: The MSG.USERID value can have the same value as the JOBNAME which is typical for a TSO address space. These values need not match; for instance, the user IDs and the job names for batch jobs or started tasks can be different.

OPSLOG Browse Column: USERID

MSG.WTOID

The internal WTO sequence number, which is taken from the WQERTCT field of the WQE delimited word of the message text.

Data Type

4-byte binary (unprintable), read-only

Example

'00004C94'X

Notes:

- The MSG.WTOID value provides information that the OPSSEND function of OPS/REXX uses in transmitting message events to other systems.
- Your system assigns sequence numbers to WTO or WTOR messages. Use these sequence numbers to delete (using DOM) highlighted, nonscrollable messages. Use the sequence of a message as its unique token identifier, but do not use the contents of the sequence number itself.
- To store a WTO ID, use the C2X REXX function. To use a WTO ID, such as with the OPSDOM built-in function, use the X2C REXX function.

OPSLOG Browse Column: WTOID

MSG.WTOR

A value indicating whether the current message is a WTOR.

Data Type

Character, read-only

Values

0

This message is not a WTOR.

1

This message is a WTOR.

Example

1

OPSLOG Browse Column: Second bit of OPSFLAGS

MSG.WTP

A value indicating whether the message is set to Write to Programmer (WTP). WTP messages appear in the JESYSMSG data set. The JESYSMSG data that is set. These data set allocated data set messages appear and are different from the job log of the job that issued the message or on whose behalf the message was issued. If the issuing address space is a TSO address space, a WTP message is written (TPUT) to the TSO screen.

Data Type

Integer, read/write

Values

1

Issue the WTP

0

Do not issue the WTP

Example

1

For example, Use the OPSPRM function to change the severity of message OPS1000 to J. Keep the results of SAY statements from rules out of the JESYSMSG data set by setting the MSG.WTP variable to 0 in a message rule for OPS1000J.

By default, messages that are issued with a ROUTE code of 11 have MSG.WTP set to 1.

OPSLOG Browse Column: Seventh bit of AFLAGS

Debug an MSG Rule

To debug an MSG rule

In cases where the)MSG is not executing, verify that you have the correct message specifier defined. You can do this by displaying the MSGID column in the OPSLOG for the problem message. The message specifier of the)MSG rule should be the same as the value of this column.

More information:

Code and Debug AOF Rules (see page 59)

MSG Rules Examples

Example 1

Demonstrates how MSG rules can respond to WTOR events as they are generated:

```
)MSG $HASP426
)INIT
/********************************************************/
/* Verify rule is only enabled on our development system */

/********************************************************/
if OPSINFO('SMFID') ¬= 'SYST' then
  return 'REJECT'
)PROC
/********************************************************/
/* Reply COLD to the JES2 initialization WTOR message    */
/* MSGTXT - IDNUM $HASP426 SPECIFY OPTIONS - SYST        */
/********************************************************/
ID = MSG.REPLYID  /* Get REPLYID from  an event variable */
address OPER      /* Set environment to issue cmds       */
"C(R "ID",COLD) NOOUTPUT"   /* Issues a z/OS REPLY command*/
```

Example 2

Demonstrates how to incorporate REXX tools and OPS/REXX tools to make various automated decisions about a particular event:

```
)MSG IEF450I
/***************************************************************/
/* Manipulate JOB ABEND messages using the following criteria   */
/* -Suppress all IEF450I except those prefixed with P (PROD)    */
/* -Hilite the abend message if JOBNAME = PMNTHEND              */
/* -Invoke ACCTRECV OPS/REXX program for all PACCT* JOBS        */
/* -Start DRECOVER JOB if PDAILY1 ABENDS with S000 & U0004      */
/* IEF450I AMAJA03 CATSO CATSO - ABEND=S000 U0004 REASON=0000   */
/*   TIME=08.00.18                                              */
/***************************************************************/
)PROC
if MSG.MLWTOMIN =1 then return  /* No need to look at 2nd line  */
JOB = MSG.JOBNAME               /* Get the JOBNAME that abended */
if SUBSTR(JOB,1,1) ¬= 'P' then  /* suppress all non prod jobs   */
   return 'SUPPRESS'
/***************************************************************/
```

```
/* Further manipulate the abending production job              */
/*****************************************************************/
select
 / *Hilite message if PMNTHEND abended, setting the descriptor  */
 /* code environmental variable using the OPS/REXX OPSBITS      */
 /* function.                                                   */
 when JOB = 'PMNTHEND' then
   do
     MSG.DESC=OPSBITS('HILITE')
   end       /*END OF PMNTHEND CHECK*/
 /* If PDAILY1 ABENDs, start DRECOVER JOB only if ABEND code    */
 /* in message is 'S000' with a user code of 'U0004'.           */
 /* Use the REXX PARSE instruction to break down the message.   */
when JOB = 'PDAILY1' then
   do
     parse var MSG.TEXT . 'ABEND=' ACODE UCODE .
     if ACODE = 'S000' & UCODE = 'U0004' then
       do
         address OPER
         "COMMAND(S DRECOVER) NOOUTPUT"
       end                              /*end of code checks   */
   end                                 /*end of pdaily1 check */
 /*****************************************************************/
 /* Schedule the ACCTRECV OPS/REXX program to a server if        */
 /* this is a production accounting job (PACCT*). Pass the job   */
 /* to the EXEC. We have to schedule the EXEC to run in a server */
 /* since it will be issuing WTORs to operations and will        */
 /* wait for the operator responses.                             */
 /* Remember that waiting in MSG rules is not allowed!!!         */
 /*****************************************************************/
when SUBSTR(JOB,1,5) = 'PACCT' then
   do
     address OSF                        /* Ship to server     */
     "OI P(ACCTRECV) ARG("JOB")"
   end                                  /* End of PACCT check */
 otherwise RETURN 'NORMAL'              /* NOT A SPECIAL CASE */
end                                     /* END OF SELECT      */
```

OMEGAMON Rules

An OMEGAMON (OMG) rule allows you to trigger automation from exception analysis data that is generated by IBM® Tivoli® OMEGAMON® XE on z/OS products and reported to an OMEGAMON terminal.

Installation Requirements for OMG Rules

Establishing an interface between CA OPS/MVS and OMEGAMON lets you create OMG rules against messages generated by the exception analysis feature of IBM® Tivoli® OMEGAMON® XE on z/OS products.

Note: For more information, see the *CA OPS/MVS Administration Guide*.

)OMG—Event Specifier of OMG Rules

The following is the format for coding the OMG-event definition section:

)OMG *omgexspec*

omgexspec

Specifies the exception label. Follow these guidelines when specifying the character string:

- Specify one to four characters of the exception label.
- The string cannot contain embedded blank spaces.
- You can use the wildcard (*) character. For example,
  - N* matches NVSC or any exception label beginning with an N.
  - *C matches any exception label ending with a C.
  - * alone matches all exception labels.
- Lowercase letters are acceptable, but the AOF converts them to uppercase for event testing.

Initialization, Processing, and Termination Sections of OMG Rules

The general guidelines for coding the initialization, processing, and termination sections and the various AOF tools that you can use, apply to OMG rules.

More information:

AOF Rule Structure (see page 21)
AOF Rule Tools (see page 29)

RETURN Statements in the)PROC Section of an OMG Rule

The OPS/REXX RETURN statement has no special meaning in the processing section of an OMEGAMON rule. The return value has no effect on AOF processing.

Execution Considerations for OMG Rules

The processing section of a rule that responds to an OMEGAMON exception event executes in the OMEGAMON address space that generated the exception. Therefore, any type of logic that could possibly suspend the processing of an OMG rule should be performed by scheduling an OPS/REXX program to a CA OPS/MVS OSF TSO, TSL, or TSP server.

The AOF execution limits apply to the processing section of a rule that responds to an OMEGAMON event.

More information:

Code and Debug AOF Rules (see page 59)

OPS/REXX Host Environments in the)PROC Section of an OMG Rule

The)PROC section of an OMG rule has the following host environments with the following OMG rule characteristics. Specify the AOFDEFAULTADDRESS parameter for the default host environment for ARM rules.

ADDRESS AOF

Sent to CA OPS/MVS. Does not wait. Output is not returned.

ADDRESS AP

Sent to MSF and then forwarded to the CA Automation Point system. Does not wait. Output is not returned.

ADDRESS EPI

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS HWS

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS ISPEXEC

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS LXCON

The VM and Linux command requests sent to USS server for execution. Does not wait. Output is not returned. The List request runs inline and returns VM and Linux system data in stem variables.

ADDRESS MESSAGE

Sent as a WTO. The AOFDEST parameter specifies the destination.

ADDRESS MQ

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS NETMAN

Sent to a CA OPS/MVS internal CA Netman request queue to issue MGPT commands. Does not wait. Output is not returned.

ADDRESS NETMASTR

Sent to CA NetMaster NM for SNA on the local system. Does not wait. Output is not returned.

ADDRESS OPER

Sent to target console as specified through the OCCONSOLENAME parameters when no CONNAME operands are present. Output is not returned. Schedule an OPS/REXX program in a server when a command output interrogation is needed.

ADDRESS OPSCTL

Sent to a specified facility. If the facility is ECF or OSF, does not wait. If the facility is MSF, slight wait occurs. The external data queue returns the output.

Note: If the command is MSF LIST, no wait occurs.

ADDRESS OPSDYNAM

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS OSF

Schedule TSO commands, CLISTs, or REXX EXECs to <CA OPS/MVS OSF TSO servers.

ADDRESS OSFTSL

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSL servers.

ADDRESS OSFTSP

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSP servers.

ADDRESS SOF

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS SQL

Does not wait. Processed synchronously for requests that can be satisfied on the local system. Output is returned in stem variable. Error messages, if any, are returned to an external data queue.

ADDRESS SYSVIEWE

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS TSO

Sent to an OSF TSO server. Does not wait. Output is not returned.

ADDRESS USS

Sent to a USS server. Does not wait. Output is not returned. Schedule an OPS/REXX program in a server if a command output interrogation is needed.

ADDRESS WTO

Does not wait. Output is sent to the specified console. When attempting a WTOR, host command is sent to a TSO server for execution. The response is returned to the server. Schedule an OPS/REXX program in a server if the WTOR response interrogation is needed.

More information:

Code and Debug AOF Rules (see page 59)

AOF Variables Available in an OMG Rule

You can use all AOF variable types in OMG rules, as described in the chapter "AOF Rule Tools (see page 29)." You can use the following unique AOF event variables in the)PROC section of a OMG rule, and you can manually interrogate the corresponding OPSLOG display field as an aid in debugging or implementing rule logic.

OMG.COLOR

The color a message line is used in OPSLOG Browse

Data Type: 1-byte binary (unprintable), read/write

Sample Value: '00'X

Notes:

- Use the OPSCOLOR function of OPS/REXX to set the OMG.COLOR variable.
- To change the default display colors for OMEGAMON messages, reset the CA OPS/MVS parameters whose names begin with OMG. For descriptions of these parameters, see the *Parameter Reference*.
- If multiple rules set OMG.COLOR for a single OMEGAMON message, CA OPS/MVS uses only the last value. To trace the color set by each rule, set the CA OPS/MVS RULETRACE parameter to ON. For a description of the RULETRACE parameter, see the *Parameter Reference*.

OPSLOG Browse Column: COLOR

OMG.DDNAME

The 8-character ddname associated with the OMEGAMON report file

Data Type: Character, read-only

Sample Value: OMREPORT

Note: Use this variable to identify the source of an OMEGAMON exception event.

OPSLOG Browse Column: TERMNAME

OMG.JOBNAME

The name of the OMEGAMON address space that caused the rule to execute

Data Type: Character, read-only

Sample Value: OMEGTASK

OPSLOG Browse Column: JOBNAME

OMG.NAME

The name of the OMEGAMON exception (for instance, DNSR for DASD Not Responding). Usually, this is a 4-character code.

Data Type: Character, read-only

Sample Value: XREP

Note: The OMG.NAME variable value determines which OMEGAMON rules execute for the current OMEGAMON event. For descriptions of the OMEGAMON exception codes, see your OMEGAMON manuals. For information on setting up the CA OPS/MVS interface to OMEGAMON, see the *Administration Guide.*

OPSLOG Browse Column: MSGID

OMG.REPORTID

A unique report ID you can use to identify the source of an OMEGAMON message. This is taken from the SUBSYS parameter (if specified).

Data Type: Character, read-only

Sample Value: CICSA

Note: Knowing where OMEGAMON events are coming from can be useful when you monitor multiple CICS systems from a single address space. The OMG.REPORTID variable enables you to tell to which CICS system the event refers.

OPSLOG Browse Column: DSPNAME

OMG.SYSID

The ID of the system where OMEGAMON is running. For JES3 messages, the SYSID value derives from the MPNAME field of the Active Main Processor Control Table. For JES2 messages, the SYSID value derives from the SMF ID string.

Data Type: Character, read-only

Sample Value: S000

OPSLOG Browse Column: SYSID

Note: The OPSLOG Browse column displays two characters of this variable, not the complete field. The CA OPS/MVS BROWSEIDFORMAT parameter determines which characters are displayed. For a description of the BROWSEIDFORMAT parameter, see the *Parameter Reference*.

OMG.TEXT

The text of the exception message

Data Type: Character, read-only

Sample Value: 'XREP Number of Outstanding Replies = 4'

Notes:

- Some OMEGAMON exceptions generate several lines of output; when this occurs, the OMG.TEXT variable contains all of these lines concatenated together.
- All sequences of multiple blanks are compressed to a single blank.
- If the exception produces more output lines than the OMG.TEXT variable can hold, CA OPS/MVS truncates the lines that do not fit.

OPSLOG Browse Column: Text is always displayed.

OMG.USER

An 8-byte variable providing communication between rules that execute for the same OMEGAMON event. The variable can contain any installation data that these rules need, and it can store a character string displayable through OPSLOG Browse.

Data Type: User-defined, read/write

Notes:

- Before AOF processing, this variable is initialized to binary zeros. It is then passed to each rule that executes for the same OMEGAMON event; each rule can look at or change the variable contents before passing the variable to the next rule for the OMEGAMON event.
- The primary purpose of the USER variable is to provide a method to pass a small amount of data between the rules. This data may be binary or mixed case. The USER field may also be used for filtering in the OPSLOG. However, USER data used for OPSLOG filtering must be uppercase and displayable.

OPSLOG Browse Column: USER

Debug an OMG Rule

To debug an OMG rule

1. Set the CA OPS/MVS BROWSEOMG parameter to YES
2. Set the OMG event profile of your OPSLOG display to Y.

   This lets you view all OMG events.
3. With these parameters set, display the OPSLOG EVENT column

   This lets you see recorded OMG events. This record will contain details for each OMEGAMON exception that is generated.

More information

Code and Debug AOF Rules (see page 59)

Example: OMG Rule

The following is an example of a rule that responds to an OMEGAMON event. The rule cancels or stops any job that exceeds the exception limit.

```
)OMG NVSC
 )PROC
 /* Extract the type of address space that generated the */
 /* NVSC exception and the associated JOBNNAME. Issue    */
 /* the appropriate command to terminate the address     */
 /* space based on the type of address space.            */

 ADDRTYPE = WORD(OMG.TXT,2)
 JOBN = WORD(OMG.TXT,3)
 address OPER            /* Send Host commands to OPER env */
 select
  when (ADDRTYPE = 'STC') then
   'P 'JOBN
  when (ADDRTYPE ='BAT') then
   'C 'JOB
  otherwise
   'C U='JOBN
 end
 )END
```

Request Rules

A request (REQ) rule provides an arbitrary method in which end users (primarily TSO users), can invoke AOF rule processing. REQ rules allow you to perform any task that you would typically perform with a standard OPS/REXX program, with these added benefits:

- Because an AOF rule is not subject to security processing, you can distribute a normally authorized task to unauthorized users (think of a request rule as a kind of authorized OPS/REXX program library).
- Because an AOF rule is preloaded (usually at CA OPS/MVS startup), it begins executing much sooner than a standard OPS/REXX program.
- Because an AOF rule is precompiled, you can enhance the performance of a user-written, interactive application by writing an OPS/REXX program that uses one or more request rules.
- Use ADDRESS ISPEXEC services.
- Get the output of any command from the OPS/REXX external data queue.
- Easily control accessibility by enabling or disabling the rule.

Installation Requirements for REQ Rules

The REQ rule is triggered whenever the OPSREQ command processor executes.

REQ rules can also be triggered through the System State Manager EVRULE and RULE actions.

Note: For more information, see the *Command and Function Reference*.

)REQ—Event Specifier of REQ Rules

The following is the format for coding the REQ-event definition section:

`)REQ` *`requestcode`*

requestcode

Specifies the request code. Follow these guidelines when specifying the character string:

- Specify 1 to 10 characters of the request code specifier.
- The string cannot contain embedded blank spaces.
- You can use the wildcard (*) character. For example,
  - CANCEL* matches CANCELJOB, CANCELUSER, and any other request code beginning with CANCEL.
  - * alone matches all request codes.
- Lowercase letters are acceptable, but the AOF converts them to uppercase for event testing.

Initialization, Processing, and Termination Sections of REQ Rules

REQ rules follow the general guidelines for coding the initialization, processing, and termination sections and the various AOF tools that you can use.

More information:

AOF Rule Structure (see page 21)
AOF Rule Tools (see page 29)

RETURN Statements in the)PROC Section of a REQ Rule

The return value has no effect on AOF processing unless the request rule is invoked as the System State Manager EVRULE action keyword.

Note: For more information, see the chapter “Using System State Manager” in the *User* Guide.

Execution Considerations for REQ Rules

The processing section of a rule that responds to a request event executes in the address space from which the request originated, which is usually that of a TSO user that invokes the OPSREQ command processor. Automation that requires waiting, such as file allocation, issuing WTORs and manipulating the reply, and issuing system commands and interrogating the output can be performed. Therefore, this type of automation suspends the address space that triggered the REQ rule.

Note: Creating this type of automation in REQ rules triggered through the System State Manager EVRULE and RULE actions suspends the SSM task.

The OPS/REXX batch execution limits as controlled through the REXX* parameters apply to the processing section of a rule that responds to a request event.

Note: For more information, see the chapters “Using OPS/REXX” and “Using System State Manager” in the *User Guide*.

OPS/REXX Host Environments in the)PROC Section of a REQ Rule

The)PROC section of an REQ rule has the following host environments with the following REQ rule characteristics. The REXXDEFAULTADDRESS parameter specifies the default host environment for REQ rules.

ADDRESS AOF

Runs inline. Waits for output in an external data queue.

ADDRESS AP

Sent to MSF and then forwarded to the CA Automation Point system. Does not wait. Output is not returned.

ADDRESS EPI

Runs inline. Waits for output in an external data queue.

ADDRESS HWS

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS ISPEXEC

Runs inline. Waits for output in an external data queue.

ADDRESS LXCON

Runs inline. Waits for output that is returned in stem variables.

ADDRESS MESSAGE

Sent to a TSO user when invoked from a foreground TSO session. Otherwise, it is sent as a WTO.

ADDRESS MQ

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS NETMAN

Sent to a CA OPS/MVS internal CA Netman request queue to issue MGPT commands. Waits for a command completion. External data queue returns output.

ADDRESS NETMASTR

Sent to CA NetMaster NM for SNA on the local system. Does not wait. Output is not returned.

ADDRESS OPER

Sent to target console as specified through the OCCONSOLENAME parameters when no CONNAME operands are present. Output is not returned. Schedule an OPS/REXX program in a server when a command output interrogation is needed.

ADDRESS OPSCTL

Sent to a specified facility. If the facility is ECF or OSF, does not wait. If the facility is MSF, slight wait occurs. External data queue returns output.

Note: If the command is MSF LIST, no wait occurs.

ADDRESS OPSDYNAM

Runs inline. Returns output in variables.

ADDRESS OSF

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSO servers.

ADDRESS OSFTSL

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSL servers.

ADDRESS OSFTSP

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSP servers.

ADDRESS SOF

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS SQL

Does not wait. Processed synchronously for requests that can be satisfied on the local system. Output is returned in stem variable. Error messages, if any, are returned to an external data queue.

ADDRESS SYSVIEWE

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS TSO

Runs inline. Waits for output in an external data queue.

ADDRESS USS

Runs inline. Waits for output that is returned in stem variables.

ADDRESS WTO

Does not wait. Output is sent to the specified console. If you attempt a WTOR, runs inline and waits for response in an external data queue.

More information:

Code and Debug AOF Rules (see page 59)

AOF Variables Available in REQ Rules

You can use all AOF variable types in REQ rules, as described in the chapter "AOF Rule Tools (see page 29)." You can use the following unique AOF event variables in the)PROC section of a REQ rule, and you can manually interrogate the corresponding OPSLOG display field as an aid in debugging or implementing rule logic.

REQ.CODE

The current OPSREQ event specifier, taken from the CODE keyword of the OPSREQ command

Data Type: Character, read-only

Sample Value: CANTSO

Note: You can enter the CODE value of OPSREQ either as a keyword (for example, CODE(CANTSO)) or as a positional parameter. The CODE value determines which request rules execute for the current event.

OPSLOG Browse Column: MSGID

REQ.TEXT

A description of the current request

Data Type: Character, read-only

Sample Value: CANTSO USERA

Note: CA OPS/MVS creates the REQ.TEXT variable by concatenating the value of the CODE clause and the value of the TEXT clause in an OPSREQ command. A blank separates the two values.

OPSLOG Browse Column: Text is always displayed.

REQ.USER

An 8-byte variable providing communication between rules that execute for the same request event. The variable can contain any installation data that these rules need, and it can store a character string displayable through OPSLOG Browse.

Data Type: User-defined, read/write

Note: Before AOF processing, this variable is initialized to binary zeros. It is then passed to each rule that executes for the same request event; each rule can look at or change the variable contents before passing the variable to the next rule for the request event.

OPSLOG Browse Column: USER

Debug an REQ Rule

To debug an REQ rule

1. Set the CA OPS/MVS BROWSEREQ parameter to YES
2. Set the REQ event profile of your OPSLOG display to Y.

   This lets you view all REQ events.
3. With these parameters set, display the OPSLOG EVENT column

   This lets you see recorded REQ events. This record shows the request rule that was invoked and you can display the OPSLOG JOBNAME column to see the TSO user that invoked the rule.

More information:

Code and Debug AOF Rules (see page 59)

Examples: REQ Rules

The following examples show the response to request events.

- Example 1: The following is an example of a rule that responds to a request event. The rule allows a TSO user to cancel any other TSO user who has the same user ID prefix.

```
)REQ CANTSO
)PROC
/*************************************************************/
/* Allow users in the same group to cancel each other's      */
/* TSO ids. Invoke this REQ rule from TSO by entering:      */
/*  OPSREQ CANTSO userid from any TSO command line          */
/*************************************************************/
```

```
USERID = TRANSLATE(WORD(REQ.TEXT,2))       /* passed userid */

/* See if this users TSO id is the same as the passed userid*/

if SUBSTR(USERID,1,4) = SUBSTR(OPSINFO(JOBNAME),1,4) then
  do
    address OPER
    'COMMAND(C U='USERID') NOOUTPUT'         /* OK to cancel */
  end
else
  say 'You are not authorized to cancel user 'USERID
```

- Example 2: The following is an example of a rule that responds to a request event. The rule obtains and displays various system-related data.

```
)REQ SYSINFO
)PROC
/***************************************************************/
/* TSO users can invoke this REQ rule by entering:             */
/*  OPSREQ SYSINFO from any TSO command line                   */
/* This rule will collect and display various system info      */
/* by invoking various OPS/REXX functions.                     */
/***************************************************************/
 SYSPLEX_INFO = OPSYSPLX('I','S')       /* SYSPLEX information */
 say '***Number of systems in PLEX = 'SYSPLEX_INFO
 do while QUEUED() > 0
   pull EDQ
   say 'PLEX info='EDQ
 end

CMD = OPSIPL('IEASYS','CMD')      /* COMMNDxx used in last IPL */
 say 'COMMNDxx members used for this IPL='cmd

say 'CPUID   = 'OPSINFO('CPUID')             /* CPUID          */
say 'DFSMSVERSION  = 'OPSINFO('DFSMSVERSION')/* DFSMSVERSION   */
say 'SYSTEM IPL DATE = 'OPSINFO('IPLDATE')   /* IPL DATE       */
say 'IPL VOLSER NAME = 'OPSINFO('IPLVOLSER') /* IPL VOLSER     */
say 'IPL TYPE  = 'OPSINFO('IPLTYPE')         /* IPL TYPE       */
say 'SYSTEM IPL TIME = 'OPSINFO('IPLTIME')   /* IPL TIME       */
```

Screen Rules

A screen rule (SCR) is triggered when a screen change occurs on an External Product Interface (EPI) virtual terminal. The EPI permits CA OPS/MVS to communicate with any VTAM application that supports IBM 3270 type virtual terminals. For any EPI terminal defined to a VTAM application that can be set up to generate automatic screen updates, an SCR can be created to process these screen changes.

Do not design SCR rules that trigger from programmatic keystrokes entered to an EPI terminal by the ADDRESS EPI OPS/REXX environment. The ADDRESS EPI keystroke tools have mechanisms to trap and return the screen image so that you can programmatically manipulate the screen contents in the OPS/REXX program that issued the keystroke.

Note: For more information, see the *Command and Function Reference*.

Installation Requirements for SCR Rules

The External Product Interface (EPI) requires the use of VTAM virtual terminals, in addition to the issuing of a series of ADDRESS EPI host environment commands that define and activate the terminals.

Note: For information on EPI terminal implementation, see the *Administration Guide* and for information on controlling EPI terminals, see the *User Guide*.

)SCR—Event Specifier of SCR Rules

Use this format when coding the screen-event definition section:

```
)SCR termspec
[screencond1]
[screencond2]
[screencond3]
       .
       .
       .
[screencond10]
```

termspec

Terminal name specifier:

- Specify one to eight characters of the EPI virtual terminal name.
- The string cannot contain embedded blank spaces.
- You can use the wildcard (*) character.
- Lowercase letters are acceptable, but the AOF converts them to uppercase for event testing.

screencond1* through *screencond10

Compares the screen condition of a specified string with a portion of the virtual terminal screen. If you do not specify one or more screen conditions (*screencond1* through *screencond10*), a screen rule executes when the terminal:

- Becomes enabled or disabled.
- Logs on or off.
- Receives an instruction from a host application to unlock its keyboard.
- Receives data from a host application that causes a screen change.

If you specify one or more screen conditions, a screen rule executes only when:

- At least *one* of the conditions is satisfied when the screen event occurs.
- A non-screen update (ENABLE, DISABLE, KBUNLOCK, LOGON, or LOGOFF) condition occurs.

Use this format when coding a screen condition:

```
row column [string]
```

row* and *column

Specify any of the following values:

- A positive integer.
- A range, specified by a pair of integers separated by a colon. For example, 5:15.
- The wildcard (*) character (equivalent to the range 1:255).

string

(Optional) The value of *string* has these characteristics:

- It is a character string enclosed in double or single quotes.
- It defaults to a null string if you do not specify a value.
- Only strings wholly contained in one row of the terminal screen can match a specified *string* value. If a screen string wraps to the next line, it cannot match any screen condition that you specify.

How Screen Rules Are Triggered

The following information concerns how screen rules are triggered:

- Screen update events and keyboard unlocking events occur after the host application sends a VTAM Request Unit (RU) chain.
  - Some applications (such as ISPF) send a full screen of data using multiple RUs
  - Other applications (such as native TSO in READY mode) send one line of data per RU.
- A screen rule executes once per screen event. For example, a rule triggered when the word READY appears in a screen update executes only once even if READY appears three times on the screen.
- A rule triggered by *any* change, either to an entire screen or to a portion of a screen, executes only once.
- The string contained in the SCR.TEXT event-related variable indicates the first position (row and column) of the change and the new text found at that location.
- If your rule is concerned with specific screen updates, have the rule check the SCR.TYPE event-related variable for the SCRUPDATE screen-update value.
- Because SCR rules allow multiple spec lines, they must be followed by a section header before comments can be used; either a)INIT card must be inserted prior to the comment card or the comment must be moved after the)PROC card.

Initialization, Processing, and Termination Sections of SCR Rules

The general guidelines for coding the initialization, processing, and termination sections and the various AOF tools that you can use, apply to SCR rules.

More information:

AOF Rule Structure (see page 21)
AOF Rule Tools (see page 29)

RETURN Statements in the)PROC Section of an SCR Rule

The RETURN statement has no special meaning in the processing section of a SCR rule.

Execution Considerations for SCR Rules

The processing section of a SCR rule executes in the CA OPS/MVS main address space. Therefore, any complex logic or interactive logic that may cause a wait to occur should be done in an OPS/REXX program that gets scheduled to an OSF TSO server on behalf of the SCR rule. For a discussion of dispatching OPS/REXX programs to OSF TSO, TSL, or TSP servers, see the chapter "Code and Debug AOF Rules (see page 59)."

The AOF execution limits apply to the processing section of a rule that responds to a screen event.

More information:

Building and Controlling AOF Rules (see page 49)

OPS/REXX Host Environments in the)PROC Section of an SCR Rule

The)PROC section of an SCR rule has the following host environments with the following SCR rule characteristics. The AOFDEFAULTADDRESS parameter specifies the default host environment for SCR rules.

ADDRESS AOF

Sent to CA OPS/MVS. Does not wait. Output is not returned.

ADDRESS AP

Sent to MSF and then forwarded to the CA Automation Point system. Does not wait. Output is not returned.

ADDRESS EPI

Runs inline. Waits for output in an external data queue.

ADDRESS HWS

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS ISPEXEC

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS LXCON

Runs inline. Waits for output that is returned in stem variables.

ADDRESS MESSAGE

Sent as a WTO. The AOFDEST parameter specifies the destination.

ADDRESS MQ

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS NETMAN

Sent to a CA OPS/MVS internal CA Netman request queue to issue MGPT commands. Does not wait. Output is not returned.

ADDRESS NETMASTR

Sent to CA NetMaster NM for SNA on the local system. Does not wait. Output is not returned.

ADDRESS OPER

Sent to target console as specified through the OCCONSOLENAME parameters when no CONNAME operands are present. Output is not returned. Schedule an OPS/REXX program in a server when a command output interrogation is needed.

ADDRESS OPSCTL

Sent to a specified facility. If the facility is ECF or OSF, does not wait. If the facility is MSF, slight wait occurs. The external data queue returns the output.

Note: If the command is MSF LIST, no wait occurs.

ADDRESS OPSDYNAM

Runs Inline. Returns output in variables.

Note: For possible implications, see Execution Considerations for SCR Rules (see page 255) in this chapter.

ADDRESS OSF

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSO servers.

ADDRESS OSFTSL

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSL servers.

ADDRESS OSFTSP

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSP servers.

ADDRESS SOF

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS SQL

Does not wait. Processed synchronously for requests that can be satisfied on the local system. Output is returned in stem variable. Error messages, if any, are returned to an external data queue.

ADDRESS SYSVIEWE

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS TSO

Sent to the server. Does not wait. Output is not returned.

ADDRESS USS

Runs inline. Waits for output in stem variables.

Note: For possible implications, see Execution Considerations for SCR Rules (see page 255).

ADDRESS WTO

Does not wait. Output is sent to the specified console. If you attempt a WTOR, runs inline and waits for response in the external data queue.

Note: For possible implications, see Execution Considerations for SCR Rules (see page 255).

More information:

Code and Debug AOF Rules (see page 59)

AOF Variables Available in an SCR Rule

You can use all AOF variable types in SCR rules, as described in the chapter "AOF Rule Tools (see page 29)." You can use the following unique AOF event variables in the)PROC section of a SCR rule, and you can manually interrogate the corresponding OPSLOG display field as an aid in debugging or implementing rule logic.

SCR.ASID

The ASID of the address space that caused this screen event taken from either the address space of the issuer of an EPI command or, in all other cases, from the CA OPS/MVS main address space

Data Type: 2-byte binary (unprintable), read-only

Sample Value: '003E'X

Notes:

- Use the C2X function of OPS/REXX to convert the value of this variable to hexadecimal characters.
- You can use SCR.ASID to check whether a specific address space is issuing an EPI command.

OPSLOG Browse Column: ASID

SCR.JOBNAME

The job name of the address space that caused the current screen event. This name can be one of the following:

- The name of the job which issued an EPI command
- The name of the address space of the CA OPS/MVS product

Data Type: Character, read-only

Sample Value: OPSMAIN

OPSLOG Browse Column: JOBNAME

SCR.TERMNAME

The name of the virtual terminal associated with this screen event; taken from the DEFINE command used to define the terminal to EPI

Data Type: Character, read-only

Sample Value: IMSTERM

Note: The SCR.TERMNAME variable determines which screen rules execute for the current screen event.

OPSLOG Browse Column: TERMNAME

SCR.TEXT

A description of the screen event.

Possible Values:

- ENABLE-The virtual terminal has been enabled.
- DISABLE-The virtual terminal has been disabled.
- LOGON-The virtual terminal has logged on to an application.
- LOGOFF-The virtual terminal has logged off from an application.
- CHANGE-The virtual terminal characteristics have been changed (the EPI CHANGE, SETMODEL, and SETUNAME commands only).
- SCRUPDATE *row col string*-The virtual terminal screen has been updated at the named row and column. The string reflects the new text on the screen.
- MVCURSOR *row col*-The virtual terminal cursor has been moved to the specified row and column.
- KBUNLOCK-The virtual terminal keyboard has been unlocked.

Data Type: Character, read-only

Sample Value: SCRUPDATE 3 2 READY

Note: The first word of the SCR.TEXT variable always matches the value of the SCR.TYPE variable.

OPSLOG Browse Column: Text is always displayed.

SCR.TYPE

A description of the screen event.

Possible Values:

- ENABLE-The virtual terminal has been enabled
- DISABLE-The virtual terminal has been disabled
- LOGON-The virtual terminal has logged on to an application
- LOGOFF-The virtual terminal has logged off of an application
- CHANGE-The virtual terminal characteristics have been changed (the EPI CHANGE, SETMODEL, and SETUNAME commands only)
- SCRUPDATE-The virtual terminal screen has been updated
- MVCURSOR-The virtual terminal cursor has been moved
- KBUNLOCK-The virtual terminal keyboard has been unlocked

Data Type: Character, read-only

Sample Value: SCRUPDATE

Note: The value of SCR.TYPE always matches the first word of the SCR.TEXT variable.

OPSLOG Browse Column: MSGID

SCR.USER

An 8-byte variable providing communication between rules that execute for the same screen event. The variable can contain any installation data that these rules need, and it can store a character string displayable through OPSLOG Browse.

Data Type: User-defined, read/write

Notes:

- Before AOF processing, this variable is initialized to binary zeros. It is then passed to each rule that executes for the same screen event; each rule can look at or change the variable contents before passing the variable to the next rule for the screen event.
- You can store Dialog Manager attribute characters in this variable to control the color of the OPSLOG Browse column.

OPSLOG Browse Column: USER

Debug an SCR Rule

To debug an SCR rule

1. Set the CA OPS/MVS BROWSESCR parameter to YES
2. Set the SCR event profile of your OPSLOG display to Y.

   This lets you view all SCR events.
3. With these parameters set, display the OPSLOG EVENT column

   This lets you see recorded SCR events. The record shows the screen type that triggered the rule, indicated by the RULE OPSLOG column.

More information:

Code and Debug AOF Rules (see page 59)

Examples: SCR Rules

The following example screen-event definitions show how to specify screen conditions. In the examples, the rule is triggered when:

- Example 1: A change occurs anywhere on the screen:

  ```
  )SCR TSOTERM
  * *
  ```

  or

  ```
  )SCR TSOTERM
  * * ""
  ```

  or

  ```
  )SCR TSOTERM
  ```
- Example 2: READY appears in row 1, column 2:

  ```
  )SCR TSOTERM
  1 2 "READY"
  ```
- Example 3: READY appears in column 2 of any row, on any terminal with a name beginning with TSO:

  ```
  )SCR TSO*
  * 2 "READY"
  ```
- Example 4: READY appears anywhere on the screen:

  ```
  )SCR TSOTERM
  * * "READY"
  ```

- Example 5: READY appears anywhere in row 1:

  ```
  )SCR TSOTERM
  1 * "READY"
  ```

- Example 6: ALERT appears anywhere in rows 3 through 24 in columns 2 through 72:

  ```
  )SCR TSOTERM
  3:24 2:72 "ALERT"
  ```

Security Rules

A security rule responds to a security event and provides an easy-to-use method for protecting the many CA OPS/MVS facilities. Unlike an authorization exit written in assembler language, a security (SEC) rule is easy to write, implement, and update.

Installation Requirements for SEC Rules

Use the SECURITYRULESET parameter to allocate a rule set containing only security rules, allowing you to easily enable and disable security-rules.

To prevent accidental or unauthorized editing of security rules, use the installation security product to allow read/write access to only those users that maintain security rules.

For example, consider these parameter settings:

- RULEPREFIX = 'OPSMVS.PROD'
- RULESUFFIX = 'RULES'
- SECURITYRULESET = 'SEC'

Using these examples, your security rule set is the PDS named OPSMVS.PROD.SEC.RULES and is the only rule set from which the AOF allows a SEC type rule to be enabled.

Note: For more information about the SECURITYRULESET parameter and other security parameters that affect rule security such as SECRULEFAILURE and AOFINITOPEN, see the *Parameter Reference.*

)SEC—Event Specifier of SEC Rules

Use this format for coding the security-event definition section:

```
)SEC facility||eventqualifier
```

facility

Specifies a CA OPS/MVS facility. This security-event specifier is one of the following character strings:

AP

The ADDRESS AP host environment, used to restrict commands sent to CA Automation Point.

OPSAOF

An ADDRESS AOF host command issued in an OPS/REXX program.

OPSBRW

The OPSBRW command processor, used to view entries in the OPSLOG Browse facility.

OPSCMD

The OPSCMD command processor or OPS/REXX ADDRESS OPER command, used to issue operator commands.

OPSCTL

The ADDRESS OPSCTL host environment, used to control the Multi-System Facility (MSF).

OPSDOM

The OPSDOM command processor, used to delete an outstanding message.

OPSEPI

An ADDRESS EPI host command issued from in an OPS/REXX program.

OPSGLOBAL

An OPS/REXX global variable that is accessed or updated.

OPSHFI

The OPSHFI command or REXX function, used to read, write, or delete variable records from the shared VSAM file supporting global variables.

OPSLOG

A CA OPS/MVS API request (processed by the CA OPS/MVS Automation Analyzer).

OPSOSF

An ADDRESS OSF, ADDRESS OSFTSL, or ADDRESS OSFTSP host command issued from within an OPS/REXX program.

OPSPARM

The OPSPARM command processor or the OPSPRM OPS/REXX function, used to change CA OPS/MVS parameter values.

OPSREPLY

The OPSREPLY command processor, used to reply to WTOR messages.

OPSREQ

The OPSREQ command processor, used to invoke AOF request rules.

OPSRMT

The OPSRMT command processor, used to issue a command to a remote system.

OPSSMTBL

The OPSSMTBL command processor, used to maintain the directory table that System State Manager uses to manage tables containing system resource information.

OPSVIEW

The OPSVIEW command processor, used to invoke
the CA OPS/MVS OPSVIEW interface.

OPSWTO

The OPSWTO command processor or the ADDRESS WTO host environment, used to send WTO or WTOR messages.

SOF

The ADDRESS SOF host environment, used to access the Switch Operations Facility (SOF) for managing the I/O configuration.

SQL

The OPSQL command processor or the ADDRESS SQL host environment, used to issue Structured Query Language commands.

SUBSYSDSN

A CA OPS/MVS subsystem data set that is opened.

USS

An ADDRESS USS host command issued from within an OPS/REXX program.

eventqualifier

Specifies a subset of the *facility* security-event specifier. The value is a character string.

Follow these guidelines when specifying the *eventqualifier* value:

- Concatenate the string with the *facility* string (no blanks between them).
- For all *facility* values except OPSCMD and OPSGLOBAL, you can only specify a wildcard (*) character. For example,

  ```
  )SEC OPSBRW*
  ```

  or

  ```
  )SEC OPSAOF*
  ```

- For OPSCMD, you can specify OPSCMD* to execute on all OPSCMDs or optionally specify a full command verb to be more specific. For example,

  ```
  )SEC OPSCMDMODIFY
  ```

  This *facility* and *eventqualifier* combination in this security event definition would execute for any z/OS MODIFY (or short form F) command issued through the OPSCMD command processor.

- For OPSGLOBAL, you can specify OPSGLOBAL* to execute on all OPS/REXX global variable updates, or optionally specify up to 41 characters of the global variable name, followed by the wildcard (*) character. For example,

  ```
  )SEC OPSGLOBALGLOBAL1.VAR*
  ```

  The *facility* and *eventqualifier* combination in this security event definition matches all global variables beginning with the GLOBAL1.VAR prefix.

- Lowercase letters are acceptable, but the AOF converts them to uppercase for event testing.
- You can specify an * for the event definition to process all facilities.

Initialization, Processing, and Termination Sections of SEC Rules

SEC rules follow the general guidelines for coding the initialization, processing, and termination sections and the various AOF tools that you can use.

More information:

AOF Rule Structure (see page 21)
AOF Rule Tools (see page 29)

RETURN Statements in the)PROC Section of an SEC Rule

The OPS/REXX RETURN statement specifies the final disposition of a security event. The RETURN statement can:

- Refer a security event to the OPUSEX security exit.
- Force CA OPS/MVS to deny access to a requested facility.
- Allow access to a requested facility.

Valid values for a RETURN statement in the processing section of a security rule are as follows:

NOACTION

Allows the event to occur with no intervention from the AOF. CA OPS/MVS passes the event to the OPUSEX security exit after AOF processing (if any). This is the default.

ACCEPT

Allows access to the requested facility and does not call the OPUSEX security exit.

REJECT

Denies access to the requested CA OPS/MVS facility and does not call the OPUSEX security exit.

Note: The return values listed here are character constants rather than keywords. An unrecognized return value, a misspelled value for example, defaults to a value of NOACTION.

If multiple rules respond to a single security event, the AOF uses the highest-precedence return value. The order of precedence is:

- REJECT (highest)
- ACCEPT
- NOACTION (lowest)

Execution Considerations for SEC Rules

The processing section of a rule that responds to a security event executes in the TSO users or batch address space that is attempting to invoke the specified CA OPS/MVS facility. Any type of logic that could possibly suspend the processing of an SEC rule, such as issuing a command and interrogating the output, or allocating and manipulating data sets is not practical and therefore is not allowed. The primary logic that should be incorporated in all security rules is to allow/disallow access to CA OPS/MVS facilities based on a check against the requester (TSO user, or possibly a batch job) of the facility.

Note: Security rules do not process facilities that are invoked from within other AOF rules.

The AOF execution limits apply to the processing section of a rule that responds to a security event.

More information:

Code and Debug AOF Rules (see page 59)

OPS/REXX Host Environments in the)PROC Section of an SEC Rule

The)PROC section of an SEC rule has the following host environments with the following SEC rule characteristics. The default host environment for SEC rules is specified by the AOFDEFAULTADDRESS parameter.

ADDRESS AOF

Sent to CA OPS/MVS. Does not wait. Output is not returned.

ADDRESS AP

Sent to MSF and then forwarded to the CA Automation Point system. Does not wait. Output is not returned.

ADDRESS EPI

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

Note: For details, see Code and Debug AOF Rules (see page 59).

ADDRESS HWS

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS ISPEXEC

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS MESSAGE

Sent to a TSO user if invoked from a foreground TSO session. Otherwise, it is sent as a WTO.

ADDRESS MQ

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS NETMAN

Sent to a CA OPS/MVS internal CA Netman request queue to issue MGPT commands. Does not wait. No output is returned.

ADDRESS NETMASTR

Sent to CA NetMaster on the local system. Does not wait. No output is returned.

ADDRESS OPER

Sent to target console as specified through the OCCONSOLENAME parameters when no CONNAME operands are present. Output is not returned. Schedule an OPS/REXX program in a server when a command output interrogation is needed.

ADDRESS OPSCTL

Sent to a specified facility. If the facility is ECF or OSF, does not wait. If the facility is MSF, slight wait occurs. Output is returned to an external data queue.

Note: If the command is MSF LIST, no wait occurs.

ADDRESS OPSDYNAM

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS OSF

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSO servers.

ADDRESS OSFTSL

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSL servers.

ADDRESS OSFTSP

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSP servers.

ADDRESS SOF

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS SQL

Does not wait. Processed synchronously for requests that can be satisfied on the local system. Output is returned in stem variable. Error messages, if any, are returned to an external data queue.

ADDRESS SYSVIEWE

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS TSO

Sent to an OSF TSO server. Does not wait. No output is returned.

ADDRESS USS

Sent to a USS server. Does not wait. No output is returned. Schedule an OPS/REXX program in a server if a command output interrogation is needed.

ADDRESS WTO

Does not wait. Output is sent to specified (or default) destination. If you attempt a WTOR, host command is sent to a TSO server for execution, and the response is returned to a server. Schedule an OPS/REXX program in a server if a WTOR response interrogation is needed.

More information:

Code and Debug AOF Rules (see page 59)

AOF Variables Available in an SEC Rule

You can use all AOF variable types in SEC rules, as described in the chapter "AOF Rule Tools (see page 29)." You can use the following unique AOF event variables in the)PROC section of a SEC rule, and you can manually interrogate the corresponding OPSLOG display field as an aid in debugging or implementing rule logic.

SEC.TEXT

A description of the current security event. Taken from the full security event text of the CA OPS/MVS component that the user is trying to use, the description consists of:

```
eventtype||de-aliased verb|| commandtext
```

- The first word of SEC.TEXT is the value that determines which security rules execute for the current security event
- The *de-aliased verb* is present only for OPSCMD events
- The *commandtext* is present only for OPSCMD, OPSCTL, OPSRMT, OPSAOF, and OPSEPI events
- The *eventtype* is always one of the following:
  - OPSAOF-An OPS/REXX program addressed commands to the AOF
  - OPSBRW-Someone used OPSLOG Browse
  - OPSCMD-Someone issued the OPSCMD command processor, or an OPS/REXX program issued ADDRESS OPER commands
  - OPSCTL-Someone used the MSF
  - OPSEPI - An OPS/REXX program addressed commands to the EPI
  - OPSGLOBAL - Global variables were accessed or updated
  - OPSLOG - Someone used the CA OPS/MVS Automation Analyzer to process OPSLOG
  - OPSOSF - An OPS/REXX program addressed commands to the OSF
  - OPSPARM - Someone issued the OPSPARM TSO command or invoked the OPSPRM() REXX function to change CA OPS/MVS parameters
  - OPSREPLY - Someone used the OPSREPLY command processor to respond to a WTOR
  - OPSRMT - Someone used the OPSRMT command processor to issue a command to a remote machine
  - OPSSMTBL - Someone used the OPSSMTBL command processor
  - OPSWTO - Someone used the OPSWTO command processor to issue a WTO message

- SOF - An OPS/REXX program addressed commands to the SOF
- SQL - Someone issued the OPSQL command processor, or an OPS/REXX program issued ADDRESS SQL commands
- SUBSYSDN - Someone opened a CA OPS/MVS subsystem data set
- OPSVIEW - Someone invoked the OPSVIEW application

Data Type: Character, read-only

Sample Value: OPSCMDSTOP P OPSS

OPSLOG Browse Column: Text is always displayed.

SEC.TYPE

The SEC.TYPE variable determines which rules execute for the current security event.

The type of security request from the CA OPS/MVS component that the user is trying to access. See the event types listed in the description of the SEC.TEXT variable.

Data Type: Character, read-only

Sample Value: OPSPARM

Notes:

- If the event type is OPSCMD, CA OPS/MVS appends the command verb to the event type (for instance, OPSCMDSTOP). But, because the variable can contain no more than ten characters, some of the verb may be truncated.
- If the event type is OPSGLOBAL, the SEC.TYPE variable may have the first character of the global variable name appended to OPSGLOBAL when you append the name of a global variable prefix on the)SEC rule specifier. For example, if you specify)SEC OPSGLOBALGLOBAL1.*, the SEC.TYPE variable has the value OPSGLOBALG.

OPSLOG Browse Column: MSGID

SEC.USER

The primary purpose of the USER variable is to provide a method to pass a small amount of data between the rules. This data may be binary or mixed case. The USER field may also be used for filtering in the OPSLOG. However, USER data used for OPSLOG filtering must be uppercase and displayable.

An 8-byte variable providing communication between rules that execute for the same security event. The variable can contain any installation data that these rules need, and it can store a character string displayable through OPSLOG Browse.

Data Type: User-defined, read/write

Note: Before AOF processing, this variable is initialized to binary zeros. It is then passed to each rule that executes for the same security event. Each rule can look at or change the variable contents before passing the variable to the next rule for the security event.

OPSLOG Browse Column: USER

OPAU Control Block Variables

The)PROC section of security rules can also contain variables set to the value of fields in the OPAU control block that CA OPS/MVS passes to the user exit. Some OPAU variables are available for all security events, but most are valid for only one type of security event.

OPAU control block variables have these characteristics:

- OPSLOG Browse does *not* display them.
- OPAU variables containing bit data become OPS/REXX compound symbols that are either true (indicated by 1) or false (indicated by 0).
- OPAU variables containing integers become OPS/REXX compound symbols containing numeric data. When modifying one of these variables, take its significance or its sign into account.
- If an OPAU variable containing character data is set to a value larger than the size of an OPAU field, the variable is truncated upon return. If the variable value is shorter than an OPAU field, CA OPS/MVS adds trailing blanks to that value.
- You can change the SEC.OPAURQRC variable only by setting the return value through the OPS/REXX RETURN statement.

OPAU Variables for All Security Events

The following variables are available for all types of security events:

SEC.OPAUAUSR

The authorization string of the CA OPS/MVS installation

Data Type: Character, read-only

Source: The value of the CA OPS/MVS AUTHSTRING parameter (described in the *Parameter Reference*)

Sample Value: TRIAL

SEC.OPAUBYSC

Indicates whether security processing should be bypassed.

Data Type: Bit, read-only

Source: The CA OPS/MVS component indicates whether security is bypassed.

Sample Value: 1

Note: This flag is true for the CA OPS/MVS main address space and for AOF rules.

SEC.OPAUECJB

Indicates whether the current address space is an ECF user.

Data Type: Bit, read-only

Source: The CA OPS/MVS component indicates the address space type.

Sample Value: 1

SEC.OPAUERMG

The error message text

Data Type: Character, read/write

Source: Other security rules

Sample Value: 'You are not allowed to use OPSCMD'

Notes:

- Security rules can set this variable to create and send short error messages to the current user.
- CA OPS/MVS passes this variable to each rule that a security event triggers. Each rule can examine or reset this variable before passing it to the next rules for the current event.
- If multiple security rules set this variable, CA OPS/MVS uses only the value set last.

SEC.OPAUGNER

Indicates that the CA OPS/MVS authorization routine (OPAUCK) can produce error messages.

Data Type: Bit, read-only

Source: The CA OPS/MVS component that calls OPAUCK.

Sample Value: 1

SEC.OPAUJBNA

The current job name that the ASCB points to

Data Type: Character, read-only

Source: The ASCBJBNI field for batch jobs, or the ASCBJBNS field for other address spaces

Sample Value: USERA

SEC.OPAUOPJB

Indicates whether the current address space is the main CA OPS/MVS product address space.

Data Type: Bit, read-only

Source: The CA OPS/MVS component indicates the address space type.

Sample Value: 1

SEC.OPAUOSJB

Indicates whether the current address space is an OSF server.

Data Type: Bit, read-only

Source: The CA OPS/MVS component indicates the address space type.

Sample Value: 1

SEC.OPAURQRC

The return code from the current access request

Data Type: Integer, read-only

Source: Set by authorization components

Possible Values:

- 0-Request approved
- 8-Request failed
- 12-Request abended

Sample Value: 8

SEC.OPAURQTX

The type of access request

Data Type: 1 to 10 characters, right-justified, read-only, padded with blanks

Source: The CA OPS/MVS component that calls OPAUCK.

Possible Values:

AP

ADDRESS AP Host command

OPSAOF

ADDRESS AOF command

OPSBRW

OPSLOG Browse request

OPSCMD

OPSCMD/ADDRESS OPER (MVS, VM, JES3, IMS CMD)

OPSCTL

ADDRESS OPSCTL (MSF, OSF, ECF) request

OPSDOM

OPSDOM (DOM A MESSAGE) request

OPSEPI

ADDRESS EPI command or EPI request

OPSGLOBAL

Global or Sysplex variable access/update request

OPSHFI

SHARED file I/O request

OPSLOG

OPSLOG API request

OPSOSF

OPSOSF request (OSF command request)

OPSPARM

OPSPARM (SET PARAMETERS) request

OPSREPLY

OPSREPLY (WTO/WTOR) request

OPSREQ

Attempt to execute a REQUEST Rule.

OPSRMT

SEND a command to a server request

OPSSMTBL

STATETBL request

OPSVIEW

OPSVIEW request

OPSWTO

OPSWTO/ADDRESS WTO (WTO, WTP, WTOR) request

SOF

ADDRESS SOF request

SQL

SQL/RDF request

SUBSYSDSN

Subsystem data set open request

USS

ADDRESS USS command

Sample Value: SQL. When OPAUQRTY is blank, OPAUTRTX must be checked for a valid security type.

SEC.OPAURQTY

The type of access request

Data Type: Character, read-only

Source: The CA OPS/MVS component that calls OPAUCK.

Possible Values:

<blank>

ADDRESS AP Host Command

A

OPSVIEW request

B

OPSLOG Browse request

E

OPSEPI request

F

Automated Operations Facility request

H

OPSOSF request

J

OPSDOM request

K

OPSCTL request

L

OPSLOG API request

M

OPSRMT request

N

USS request

O

OPSCMD request

P

OPSPARM request

Q

OPSREQ request

R

OPSREPLY request

U

SQL request

V

Global and Sysplex variable access/update request

W

OPSWTO request

X

Subsystem data set open request

Y

OPSSMTBL request

Z

OPSHFI request

Sample Value: A

Notes:

- This variable will be removed with the next release of CA OPS/MVS.
- In some cases where SEC.OPAURQTX contains a value, this variable contains a single blank.

SEC.OPAUSSNA

The CA OPS/MVS subsystem name to which the request was directed.

Data Type: Character, read-only

Source: Subsystem to which the request was directed.

Sample Value: OPSS

Note: You can use this variable to create a security rule that works on multiple subsystems (for example, a production and a test system).

SEC.OPAUUSID

The CA ACF2/CA Top Secret/RACF logon ID for this request

Data Type: Character, read-only

Source: The SAF user ID associated with the current task or address space

Sample Value: USERA

OPAU Variables for OPSAOF Security Events

The following variables are available for OPSAOF security events:

SEC.AUAOBULN

The command buffer length

Data Type: Integer, read-only

Source: The ADDRESS AOF host command

Sample Value: 4

SEC.AUAOCMBU

The complete ADDRESS AOF command buffer string

Data Type: Character, read-only

Source: The ADDRESS AOF host command

Sample Value: LIST IPL

SEC.AUAODSNA

The current verb data set name string

Data Type: Character, read-only

Source: The ADDRESS AOF host command

Sample Value: SYS1.OPS.CCLXRULS

Note: This variable contains between 1 and 44 characters.

SEC.AUAORLNA

The current security rule name string

Data Type: Character, read-only

Source: The ADDRESS AOF host command

Sample Value: IEF404I

SEC.AUAORQTY

The type of ADDRESS AOF request

Data Type: Character, read-only

Source: The ADDRESS AOF host command string

Possible Values:

- A-Set Auto-Enable flag
- D-Disable rule or rules
- E-Enable rule or rules
- I-List DSNAME index

- J-List in-storage rules
- L-List rule set or rule
- M-List rule source text
- S-Set or reset subsystem string
- V-Compile a rule or rules
- W-Delete a compiled rule or rules
- Y-List a compiled rule or rules
- Z-Reset auto-enable flag

Sample Value: A

SEC.AUAORSNA

The current rule set name string

Data Type: Character, read-only

Source: The ADDRESS AOF host command

Sample Value: IPL

SEC.AUAORSSC

The current security rule set name string

Data Type: Character, read-only

Source: The CA OPS/MVS SECURITYRULESET parameter

Sample Value: SEC

SEC.AUAOSCOP

Indicates that the AOF operation uses the security rule set. Check this variable if your site imposes more restrictive security for the security rule set.

Data Type: Bit, read-only

Source: The ADDRESS AOF command processor

Sample Value: 1

SEC.AUAOSYNA

The remote/local system ID of the system on which the AOF request is issued

Data Type: Character, read-only

Source: The value entered using the ADDRESS AOF SYSTEM keyword

Sample Value: SYS4

SEC.AUAOVBSR

The current verb string

Data Type: Character, read-only

Source: The first blank delimited word of the ADDRESS AOF host command

Sample Value: LIST

Note: CA OPS/MVS converts the verb to uppercase letters.

OPAU Variables for OPSBRW Security Events

The following variables are available for OPSBRW security events:

SEC.AUBODBCD

The browse database ID code

Data Type: Character, read-only

Source: The OPSLOG Browse command

Possible Values: O-MVS/JES OPSLOG Browse database

Sample Value: O

SEC.AUBODBLN

The length of the database string

Data Type: Integer, read-only

Source: The OPSLOG Browse command

Sample Value: 6

SEC.AUBODBSR

The database string, which provides a character description of the database code

Data Type: Character, read-only

Source: The OPSLOG Browse command

Sample Value: OPSLOG

SEC.AUBOSSLN

The length of the CA OPS/MVS subsystem name

Data Type: Integer, read-only

Source: The OPSLOG Browse command

Sample Value: 4

SEC.AUBOSSNA

The CA OPS/MVS subsystem name

Data Type: Character, read-only

Source: The OPSLOG Browse command

Sample Value: OPST

SEC.AUBOSYNA

The MSF system name of the remote OPSLOG that a user is trying to access

Data Type: Character, read-only

Source: The OPSLOG Browse SYSTEM primary command

Sample Value: MSIC

OPAU Variables for OPSCMD and ADDRESS OPER Security Events

The following variables are available for OPSCMD and ADDRESS OPER security events:

SEC.AUOCBULN

The command buffer length

Data Type: Integer, read-only

Sample Value: 6

SEC.AUOCCMBU

The command buffer string

Data Type: Character, read/write

Source: Command text from OPSCMD or ADDRESS OPER

Sample Value: P OPSS

Notes:

- Security rules can set this variable. The modified command text buffer then executes the actual command.
- CA OPS/MVS initially sets this variable from the command buffer that the user entered.
- Each rule triggered by the same security event can examine and reset this variable before passing it on to the next rule executed for this event.
- If multiple security rules set this variable, CA OPS/MVS uses the value set last.

SEC.AUOCCMLN

The current command verb length

Data Type: Integer, read-only

Source: The de-aliased verb string type

Sample Value: 7

SEC.AUOCCMSR

The current command verb

Data Type: Character, read-only

Source: The de-aliased command verb from OPSCMD or ADDRESS OPER

Sample Value: STOP

SEC.AUOCCNNM

Console name, or blank if no console name is specified

Data Type: Character, read-only

Source: The NAME or CONNAME keywords from OPSCMD or ADDRESS OPER

Sample Value: MASTER

SEC.AUOCDLTM

The delay time in seconds

Data Type: Integer, read-only

Source: The contents of the DELAY keyword on the OPSCMD or ADDRESS OPER command

Sample Value: 10

Note: The SEC.AUOCDLTM value will be a number between 1 and 300 seconds.

SEC.AUOCORSY

The MSFID of the system where the command originated

Data Type: Character, read-only

Sample Value: MFSSYSA

Notes:

- If the command originated in the local system, then this variable contains a null value (zero length).
- If the command was sent to this system through MSF as a result of the SYSTEM keyword of OPSCMD or ADDRESS OPER, or if the OPSCMD was imbedded in an OPSRMT command that was issued on another system, then this variable contains the name of the origin system.
- If OPSRMT is used to run a REXX EXEC on another system, and that EXEC uses ADDRESS OPER or OPSCMD, then the command is considered to have originated on the system where the OPSRMT was issued; hence this variable contains the name of the origin system.

SEC.AUOCRQTY

The type of command OPSCMD or ADDRESS OPER is processing

Data Type: Character, read-only

Source: OPSCMD or ADDRESS OPER determines the command type as follows:

- The command is an IMS command if the IMSID keyword was entered or if the command character matches the IMS command character
- If JES3 is running and the command starts with an asterisk (*) or the number eight, the command is a JES3 command
- The command is a VM command if it begins with #CP
- If the command meets none of the above criteria, it is treated as a z/OS command

Possible Values:

- M-MVS command entered
- 3-JES3 command entered
- V-VM command entered
- I-IMS command entered

Sample Value: M

SEC.AUOCSYID

The remote/local system ID string

Data Type: Character, read-only

Source: The value entered using the OPSCMD or ADDRESS OPER SYSID keyword

Sample Value: SYSA

OPAU Variables for OPSCTL Security Events

The following variables are available for OPSCTL security events:

SEC.AUCTBULN

The command buffer length

Data Type: Integer, read-only

Source: The ADDRESS OPSCTL host command

Sample Value: 10

SEC.AUCTCMBU

The command buffer string

Data Type: Character, read-only

Source: The ADDRESS OPSCTL host command

Sample Value: MSF ACTIVATE(SYSA)

SEC.AUCTCMLN

The length of the current command verb

Data Type: Integer, read-only

Source: The ADDRESS OPSCTL host command

Sample Value: 2

SEC.AUCTCMSR

The current command verb

Data Type: Character, read-only

Source: The ADDRESS OPSCTL host command

Sample Value: OK

SEC.AUCTRQTY

The ADDRESS OPSCTL request type

Data Type: Character, read-only

Source: The ADDRESS OPSCTL host command

Possible Values:

- B-OPSLOG request
- C-COF request
- E-ECF request
- M-MSF request
- O-OSF request

SEC.AUCTSYNA

One of the following values:

The MSF system name to which the OPSCTL host command will be routed for execution

ALL if the command will be routed to all active MSF-defined systems including the local system

EXT if the command will be routed to all active, remote MSF-defined systems

Data Type: Character, read-only

Source: The ADDRESS OPSCTL host command

Sample Value: MSIX

OPAU Variables for OPSEPI Security Events

The following variables are available for OPSEPI security events:

SEC.AUEPAPID

The application ID (if any) used in the command, usually specified with the APPLID() keyword and containing 0 to 8 characters

Data Type: Character, read-only

Source: The ADDRESS EPI host command

Sample Value: TSO

SEC.AUEPBULN

The length of the command buffer

Data Type: Integer, read-only

Source: The ADDRESS EPI host command

Sample Value: 4

SEC.AUEPCMBU

The complete ADDRESS EPI command buffer string. This variable can contain 0 to 256 characters.

Data Type: Character, read-only

Source: The ADDRESS EPI host command

Sample Value: 'DEFINE OPSS0001 APPLID(TSO) LOGMODE(T3278M3)'

SEC.AUEPRQTY

The request type

Data Type: Character, read-only

Source: The ADDRESS EPI host command string

Possible Values:

- '25'X-BIND command
- '01'X-CHANGE command
- '02'X-DEBUG command
- '03'X-DEFINE command
- '04'X-DELETE command
- '05'X-DEQ command
- '06'X-DISABLE command
- '07'X-ENABLE command
- '08'X-ENQ command
- '0b'X-INQINPUT command
- '0c'X-LIST command
- '0d'X-LOGON command
- '0e'X-LOGOFF command
- '23'X-MVCURSOR command
- '10'X-PEEK command
- '11'X-POKE command

- '12'X-RDCURSOR command
- '13'X-RDSCREEN command
- '14'X-RDSCRROW command
- '16'X-SETMODEL command
- '17'X-SETUNAME command
- '1c'X-TRACE command
- '1e'X-TYPE command
- '1f'X-TYPESEC command
- '20'X-TYPETEST command
- '26'X-UNBIND command

SEC.AUEPTMID

The terminal ID (if any) used in the command, up to 8 characters in length

Data Type: Character, read-only

Source: The ADDRESS EPI host command

Sample Value: OPSS0001

Note: CA OPS/MVS converts the verb to uppercase letters.

SEC.AUEPTMPW

The terminal password (if any) used in the command, usually specified with the PASSWORD() keyword and containing 0 to 8 characters

Data Type: Character, read-only

Source: The ADDRESS EPI host command

Sample Value: SECRET

SEC.AUEPVBSR

The current verb string

Data Type: Character, read-only

Source: The ADDRESS EPI host command

Sample Value: LIST

Note: CA OPS/MVS converts the verb to uppercase letters.

OPAU Variables for OPSGLOBAL Security Events

The following variables are available for OPSGLOBAL security events:

SEC.AUGLDELN

The global variable derived name length

Data Type: Integer, read-only

Source: The global variable access/update routine

Sample Value: 24

SEC.AUGLDENA

The global or sysplex variable derived name string

Data Type: Character, read-only

Source: The global or sysplex variable access/update routine

Sample Value: GLOBAL.MSG.IEF450I.COUNT

Notes:

- Global variable derived names are 1 through 84 characters long after symbol substitution.
- Sysplex variable names begin with the prefix GLVPLXTx. and can be up to 128 characters in length. A 32-byte printable hex token value can be substituted for a variable name in some sysplex variable functions. Additionally the PURGE and QUERY functions of sysplex variables do not allow variable name operands. For these cases, a constant value replaces the variable name.

Function	Value
PURGE	CAVARSRV.PURGE
QUERY	CAVARSRV.QUERY
Token name	CAVARSRV.TOKEN

SEC.AUGLOPCH

The global variable access option byte, which corresponds to the function byte operand of the OPSVALUE function of OPS/REXX.

Data Type: Character, read-only

Source: The global variable access/update routine

Possible Values:

A

Add an option value.

B

Compare/update without executing a global variable rules option value.

C

Compare/update option value.

D

Drop an option value.

E

Exist option value.

F

Exist/obtain option value.

H

High-level security option value.

I

Info an option value.

L

List an option value.

N

Obtain or return a null option value.

O

Obtain an option value.

R

Remove an option value.

S

Subtree an option value.

T

Subtree info option value.

U

Update an option value.

V

Value an option value.

Z

Update without executing a global variable rules option value.

0

Obtain or return zero option value.

1

Obtain or return null with an update token option value.

2

Compare/update with a token option value.

3

Compare/update with a token without executing a global variable rules option value.

4

Remove with the name mask option value.

5

Obtain variable names with the name mask option value.

6

Remove a single variable option value.

Sample Value: C

SEC.AUGLRQTY

The request type. You can use this variable in place of the SEC.AUGLOPCH variable.

Data Type: Character, read-only

Source: The global variable access/update routine

Possible Values:

A

Access a global variable

U

Update a global variable

Sample Value: A

SEC.AUGLSYNA

The MSF system name of the remote system to which an OPSVALUE request has been targeted.

Data Type: Character, read-only

Source: The ADDRESS OPSCTL MSF DEFAULT SYSTEM(*sysname*) command

Sample Value: SYSA

OPAU Variables for OPSHFI Security Events

The following variables are available for OPSHFI security events:

SEC.AUSHDENA

The variable name or name mask specified in the shared file request. The plus sign (+) is a single wildcard character. A trailing asterisk (*) means match all subsequent characters.

Data Type: Character, read-only

Source: The value specified for the *variablename* operand of the OPSHFI command

Sample Value: GLOBAL0.TEST.*

SEC.AUSHFUCD

The type of file request operation

Data Type: Character, read-only

Source: The function value (DELETE, READ, or WRITE) specified for the OPSHFI command

Possible Values:

- D-Delete request
- R-Read request
- W-Write request

SEC.AUSHSFID

The SMF ID when the scope value is I

Data Type: Character, read-only

Source: The value specified for the SMFID keyword of the OPSHFI command

Sample Value: TST1

SEC.AUSHSYID

The system name list on which to perform the shared file request. System names are separated by a single blank for REXX parsing.

Data Type: Character, read-only

Source: The value specified for the SYSTEM keyword of the OPSHFI command

Sample Value: SYS1 PROD06 TESTSYS

SEC.AUSHVATY

The scope of the variable records to be read, written, or deleted

Data Type: Character, read-only

Source: The OPSHFI command

Possible Values:

- B-Both local and shared variables
- I-Variables with a specific SMF ID
- L-Local variables only
- S-Shared variables only

OPAU Variables for SOF Security Events

The following variables are available for SOF security events:

SEC.AUIOCMND

Contains the entire command line as passed to the ADDRESS SOF processor.

Data Type: Character, read-only

Source: The entire ADDRESS SOF command string

Sample Value: QUERY CONTROLUNITS LOCAL SYSNAME(D44ENF9)

SEC.AUIOCTYP

Contains the SOF command.

Data Type: Character, read-only

Source: The specific command type

Sample Value: QUERY

Possible Values for PPRC:

- P-PPRCCMD DISPLAY command
- U-PPRCCMD SETUP/DELETE/FREEZE/RUN command

SEC.AUIOQTYP

Contains the target of a QUERY command.

Data Type: Character, read-only

Source: The keyword specification of the QUERY command

Sample Value: CONTROLUNITS

Note: Not used for PPRC.

SEC.OPMOUSID

Contains the TSO user ID issuing ADDRESS SOF or the submitting jobname.

Data Type: Character, read-only

Source: System provided userid or Jobname

Sample Value: JEDFR03

OPAU Variables for OPSLOG Security Events

The following variables are available for OPSLOG security events:

SEC.AULGFUCD

The function type

Data Type: Character, read-only

Source: The OPSLOG interface routine

Possible Values:

- END-Terminates access to OPSLOG
- IDENTIFY-Identifies the OPSLOG to be accessed
- RETRIEVE-Fetches data from OPSLOG
- WINDOW-Indicates that a window to carry part of the OPSLOG has been created

Sample Value: RETRIEVE

SEC.AULGLGNA

The log name of the OPSLOG to be accessed

Data Type: Character, read-only

Source: The OPSLOG interface routine

Sample Value: OPSLOG1

SEC.AULGMXSC

The maximum number of OPSLOG records that will be scanned to satisfy a retrieve request

Data Type: Integer, read/write

Source: The OPSLOG interface routine

Sample Value: 100

Note: A value of 0 means that an unlimited number of records will be scanned.

SEC.AULGSFCD

The subfunction requested

Data Type: Character, read-only

Source: The OPSLOG interface routine

Sample Value: SUBSYS

SEC.AULGSSNA

The subsystem ID of the CA OPS/MVS OPSLOG to be accessed

Data Type: Character, read-only

Source: The OPSLOG interface routine

Sample Value: OPSS

OPAU Variables for OPSOSF Security Events

The following variables are available for OPSOSF security events:

SEC.AUOSBULN

The command buffer length

Data Type: Integer, read-only

Source: The ADDRESS OSF host command

Sample Value: 5

SEC.AUOSCMBU

The complete ADDRESS OSF command buffer string

Data Type: Character, read-only

Source: The ADDRESS OSF host command

Sample Value: LISTA STATUS

SEC.AUOSHOEV

The name specified on the host command

Data Type: Character, read-only

Source: The host command environment name

Possible Values:

- OSF
- OSFTSL
- OSFTSP

Sample Value: OSF

SEC.AUOSVBSR

The function type

Data Type: Character, read-only

Source: The ADDRESS OSF host command

Sample Value: LISTA

OPAU Variables for OPSPARM and OPSPRM Security Events

The following variables are available for OPSPARM and OPSPRM security events:

SEC.AUPAPANA

The name of the parameter requested for change or display

Data Type: Character, read-only

Source: The contents of the SET or SHOW keyword clauses type

Sample Value: AOFMESSAGES

SEC.AUPARQTY

The type of OPSPARM or OPSPRM request

Data Type: Character, read-only

Source: A modification request, if the SET keyword is specified; or a display request, if the SHOW keyword is specified

Possible Values:

- D-The request includes the SHOW keyword, so the request is a display request
- M-The request includes the SET keyword, so the request is a modification request

Sample Value: D

SEC.AUPASYNA

One of the following values:

- The MSF system name to which the OPSPRM function or OPSPARM command will be routed for execution
- ALL-if the command will be routed to all active MSF-defined systems including the local system
- EXT-if the command will be routed to all active, remote MSF-defined systems

Data Type: Character, read-only

Source: The ADDRESS OPSCTL "MSF DEFAULT SYSTEM(*sysname*)" command

Sample Value: SYSA

OPAU Variables for OPSREPLY Security Events

The following variables are available for OPSREPLY security events:

SEC.AURPDLTM

The delay time in seconds

Data Type: Integer, read-only

Source: The contents of the DELAY keyword on the OPSREPLY command processor

Sample Value: 10

Note: The value of the variable will be a number between 1 and 300 seconds.

SEC.AURPFUCD

The reply function code byte

Data Type: Character, read-only

Source: The OPSREPLY command processor

Possible Values:

- R - Reply to an outstanding WTOR message
- T - Test for an outstanding WTOR message

Sample Value: R

SEC.AURPIMID

The IMS ID string

Data Type: Character, read-only

Source: The contents of the IMSID keyword on the OPSREPLY command processor

Sample Value: IMSA

SEC.AURPIMKY

Indicates that the IMSID keyword was entered

Data Type: Bit, read-only

Source: The OPSREPLY command processor

Sample Value: 1

SEC.AURPJBKY

Indicates that the JOBNAME keyword was entered

Data Type: Bit, read-only

Source: The OPSREPLY command processor

Sample Value: 1

SEC.AURPJBNA

The job name string

Data Type: Character, read-only

Source: The contents of the JOBNAME keyword on the OPSREPLY command processor

Sample Value: VTAM

SEC.AURPJNKY

Indicates that the JOBNUMBER keyword was entered

Data Type: Bit, read-only

Source: The OPSREPLY command processor

Sample Value: 1

SEC.AURPJNNM

The JES job number of the WTOR to be replied to

Data Type: Integer, read-only

Source: The contents of the JOBNUMBER keyword on the OPSREPLY command processor

Sample Value: 1406

SEC.AURPMGID

The value of the MSGID or MSGTEXT keyword string used to select matching WTORs. The SEC.AURPMGID or SEC.AURPTXKY variable determines which keyword was used.

Data Type: Character, read-only

Source: The contents of the MSGID or MSGTEXT keyword on the OPSREPLY command processor

Sample Values: DFS996I (MSGID keyword; maximum length is ten) and DSI803A A44IM (MSGTEXT keyword; maximum length is 124)

SEC.AURPMGKY

Indicates that the MSGID keyword was entered

Data Type: Bit, read-only

Source: The OPSREPLY command processor

Sample Value: 1

SEC.AURPRIID

The WTOR reply ID to be replied to

Data Type: Integer, read-only

Source: The contents of the REPLYID keyword on the OPSREPLY command processor

Sample Value: 106

SEC.AURPRIKY

Indicates that the REPLYID keyword was entered

Data Type: Bit, read-only

Source: The OPSREPLY command processor

Sample Value: 1

SEC.AURPRPKY

Indicates that the TEXT keyword was entered

Data Type: Bit, read-only

Source: The OPSREPLY command processor

Sample Value: 1

SEC.AURPRPTX

The text that will be used to reply to an outstanding WTOR message

Data Type: Character, read-only

Source: The contents of the TEXT keyword on the OPSREPLY command processor

Sample Value: /DIS A

Note: The maximum length is 119 bytes.

SEC.AURPSLCN

The number of criteria a WTOR must match to be eligible for further processing by the OPSREPLY command processor

Data Type: Integer, read-only

Source: The count of the keywords IMSID, JOBNAME, JOBNUMBER, MSGID, or MSGTEXT, REPLYID, STEPNAME, and SYSNAME on an OPSREPLY command processor

Sample Value: 3

SEC.AURPSPKY

Indicates that the STEPNAME keyword was entered

Data Type: Bit, read-only

Source: The OPSREPLY command processor

Sample Value: 1

SEC.AURPSPNA

The step name string

Data Type: Character, read-only

Source: The contents of the STEPNAME keyword on the OPSREPLY command processor

Sample Value: NET

SEC.AURPSYKY

Indicates that the SYSNAME keyword was entered

Data Type: Bit, read-only

Source: The OPSREPLY command processor

Sample Value: 1

SEC.AURPSYNA

The system name on which the WTOR to be replied to was issued. This variable is primarily used in a sysplex.

Data Type: Character, read-only

Source: The z/OS system name specified on the SYSNAME keyword of the OPSREPLY command processor

Sample Value: SYSA

SEC.AURPTXKY

Indicates that the MSGTEXT keyword was entered or that the MSGID keyword was entered with a text string value greater than ten characters

Data Type: Bit, read-only

Source: The OPSREPLY command processor

Sample Value: 1

SEC.AURPWAKY

Indicates that the WAIT keyword was entered

Data Type: Bit, read-only

Source: The OPSREPLY command processor

Sample Value: 1

SEC.AURPWATM

The wait time in seconds

Data Type: Integer, read-only

Source: The contents of the WAIT keyword on the OPSREPLY command processor

Sample Value: 30

The maximum value is 3600 seconds.

OPAU Variables for OPSREQ Security Events

The following variables are available for OPSREQ security events:

SEC.AURQFUCD

The 1 to 10-character request code

Data Type: Alphabetic, national characters, or both, read-only

Source: The CODE keyword on the OPSREQ command processor

Sample Value: CANUSER

SEC.AURQRQTX

The request text (in most cases, the length of this string cannot exceed 117 characters)

Data Type: Character, read-only

Source: The TEXT keyword on the OPSREQ command processor

Sample Value: TSOID1

OPAU Variables for OPSRMT Security Events

The following variables are available for OPSRMT security events:

SEC.AURMBULN

The command buffer length

Data Type: Integer, read-only

Source: The length of the command verb from OPSRMT

Sample Value: 1

SEC.AURMCMBU

The command buffer string

Data Type: Character, read-only

Source: Command text from OPSRMT

Sample Value: P OPSS

SEC.AURMCMLN

The current command verb length

Data Type: Integer, read-only

Source: The command verb length from OPSRMT type

Sample Value: 1

SEC.AURMCMSR

The current command verb

Data Type: Character, read-only

Source: The command verb from OPSRMT. OPSRMT *does not* strip aliases from command verbs

Sample Value: P

SEC.AURMDETY

The destination of the command OPSRMT is processing

Data Type: Character, read-only

Source: If the value of the SYSID keyword (representing the target system) matches the SYSID for the local system, the destination is the local system. Otherwise, the destination is a remote system.

Possible Values:

- L-Command destination is the local system
- R-Command destination is a remote system
- **Sample Value:** L

SEC.AURMSYID

The remote/local system ID list that will receive the remote command

Data Type: Character, read-only

Source: The value entered using the OPSRMT SYSID keyword

Sample Value: SYSA SYSB

OPAU Variables for OPSWTO Security Events

The following variables are available for OPSWTO security events:

SEC.AUWTCIKY

Indicates that the CNNAME keyword was entered on the OPSWTO command processor or ADDRESS WTO host command

Data Type: Bit, read-only

Source: OPSWTO command processor or ADDRESS WTO host command

Sample Value: 3

SEC.AUWTCNKY

Indicates that the CNID keyword was entered

Data Type: Bit, read-only

Source: The OPSWTO command processor or the ADDRESS WTO host command

Sample Value: 1

SEC.AUWTCNNM

The contents of the CNNAME keyword on the OPSWTO command processor or ADDRESS WTO host command

Data Type: Character, read-only

Source: The CNNAME keyword on the OPSWTO command processor or ADDRESS WTO host command

Sample Value: MASTER

SEC.AUWTDCCD

The WTO or WTOR descriptor codes

Data Type: Binary, read-only

Source: The contents of the DESC keyword on the OPSWTO command processor or ADDRESS WTO host command

Sample Value: '1000'X

Note: This variable contains as many as 16 bytes. To format the variable for printing, use the C2X function in OPS/REXX.

SEC.AUWTDCKY

Indicates that the DESC keyword was entered

Data Type: Bit, read-only

Source: The OPSWTO command processor or the ADDRESS WTO host command

Sample Value: 1

SEC.AUWTDLTM

The delay time in seconds

Data Type: Integer, read-only

Source: The contents of the DELAY keyword on the OPSWTO command processor or ADDRESS WTO host command

Sample Value: 10

Note: This value of the variable will be between 1 and 300 seconds.

SEC.AUWTFUCD

The WTO or WTOR function code byte

Data Type: Character, read-only

Source: The presence of the REPLY keyword determines how OPSWTO or ADDRESS WTO sets this byte

Possible Values:

- R-A WTOR is requested
- W-A WTO is requested

Sample Value: R

SEC.AUWTHIKY

Indicates that the HILITE keyword was entered

Data Type: Bit, read-only

Source: The OPSWTO command processor or the ADDRESS WTO host command

Sample Value: 1

SEC.AUWTLOKY

Indicates that the LOWLITE keyword was entered

Data Type: Bit, read-only

Source: The OPSWTO command processor or the ADDRESS WTO host command

Sample Value: 1

SEC.AUWTMCFG

The WTO or WTOR MCS flags

Data Type: Binary, read-only

Source: The contents of the MCSFLAGS keyword on the OPSWTO command processor or ADDRESS WTO host command

Sample Value: '1000'X

Note: This variable contains as many as 16 bytes. To format the variable for printing, use the C2X function in OPS/REXX.

SEC.AUWTMCKY

Indicates that the MCSFLAGS keyword was entered

Data Type: Bit, read-only

Source: The OPSWTO command processor or the ADDRESS WTO host command

Sample Value: 1

SEC.AUWTMGID

The WTO or WTOR message ID string

Data Type: Character, read-only

Source: The contents of the MSGID keyword on the OPSWTO command processor or ADDRESS WTO host command

Sample Value: ZRX011

Note: The maximum length cannot exceed 10 characters.

SEC.AUWTMGKY

Indicates that the MSGID keyword was entered

Data Type: Bit, read-only

Source: The OPSWTO command processor or the ADDRESS WTO host command

Sample Value: 1

SEC.AUWTROCD

The WTO or WTOR route codes

Data Type: Binary, read-only

Source: The contents of the ROUTE keyword on the OPSWTO command processor or the ADDRESS WTO host command

Sample Value: '10000000000000000000000000000000'X

Note: This variable contains as many as 16 bytes. To format the variable for printing, use the C2X function in OPS/REXX.

SEC.AUWTROKY

Indicates that the ROUTE keyword was entered

Data Type: Bit, read-only

Source: The OPSWTO command processor or the ADDRESS WTO host command

Sample Value: 1

SEC.AUWTRPKY

Indicates that the REPLY keyword was entered

Data Type: Bit, read-only

Source: The OPSWTO command processor or the ADDRESS WTO host command

Sample Value: 1

SEC.AUWTSYNA

The remote/local system ID of the system on which the WTO request will be issued

Data Type: Character, read-only

Source: The value entered using the OPSWTO SYSTEM keyword

Sample Value: SYSA

SEC.AUWTTXKY

Indicates that the TEXT keyword was entered

Data Type: Bit, read-only

Source: The OPSWTO command processor or the ADDRESS WTO host command

Sample Value: 1

SEC.AUWTTXSR

The WTO or WTOR text string

Data Type: Character, read-only

Source: The contents of the TEXT keyword on the OPSWTO command processor or the ADDRESS WTO host command

Sample Value: 'This is a test message'

SEC.AUWTWAKY

Indicates that the WAIT keyword was entered

Data Type: Bit, read-only

Source: The OPSWTO command processor or the ADDRESS WTO host command

Sample Value: 1

SEC.AUWTWATM

The wait time in seconds

Data Type: Integer, read-only

Source: The contents of the WAIT keyword on the OPSWTO command processor or the ADDRESS WTO host command

Sample Value: 60

OPAU Variables for SQL Security Events

The Relational Data Framework, System State Manager, and all other SQL-related components of CA OPS/MVS use the standard CA OPS/MVS security exit and AOF security rules. The following variables are available in security rules:

SEC.AUSQCAID

A single-character code that represents the environment where the SQL request originated

Data Type: Character, read-only

Source: The environment from which the SQL request originated

Possible Values:

- A-ADDRESS SQL (OPS/REXX)
- C-TSO/E CLIST
- D-Remote Automate system
- E-RDF table editor
- O-OPSQL command processor (or SQL alias)
- R-TSO/E REXX
- T-OPSSMTBL command processor (or STATETBL alias)

Sample Value: A

SEC.AUSQFTXT

The complete text of the SQL statement. The maximum size of an SQL statement in CA OPS/MVS is currently 2048 characters.

Data Type: Character, read-only

Source: The SQL statement

Sample Value: SELECT * FROM STCTBL WHERE NAME='VTAM'

SEC.AUSQFUCD

The function code returned by the OPSQL command processor. The function code should always be S for SQL.

Data Type: Character, read-only

SEC.AUSQHVCN

The count of host variable references that occur within the SQL statement. This variable can be used as a loop count limit for reading host variable names and values using the SEC.AUSQHVIX facility for obtaining a specific host variable name and value.

Data Type: Integer, read-only

Source: The SQL statement control block created by the SQL syntax compiler.

Sample Value: 3

SEC.AUSQHVIX

The host variable reference sequence number for obtaining a host variable name or value using the SEC.AUSQHVNA and SEC.AUSQHVVL variables. The maximum usable value is the value of SEC.AUSQHVCN.

The initial value is 0.

Data Type: Integer, read/write

Source: Set by user for host variable name and value retrieval

Sample Value: 2

SEC.AUSQHVNA

The host variable name of the host variable pointed to by the SEC.AUSQHVIX index number. The SQL syntax compiler creates a table of host variable names in the order they occur in the statement. The index number points to the name entry to retrieve.

Data Type: Character, read-only

Source: The SQL statement control block created by the SQL syntax compiler.

Sample Value: VNAME

SEC.AUSQHVVL

The host variable value of the host variable pointed to by the SEC.AUSQHVIX index number. Values for host variables are provided for host variable names in the SQL statement in a separate table at the time that the SQL statement is executed. In an SQL cursor statement, the values are supplied when the OPEN cursor statement is executed. The DECLARE cursor statement has no values for the host variables in the SQL statement.

Data Type: Character/Numeric, read-only

Source: The SQL host variable input value table (VIL) provided at SQL statement execution.

Sample Value: CICSAOR

SEC.AUSQRQTY

The SQL request type.

Data Type: Character, read-only.

Source: The SQL verb of the SQL statement.

Possible Values:

- A-Access a SQL table
- U-Update a SQL table

Sample Value: A

SEC.AUSQSMTY

The SQL statement type.

Data Type: Character, read-only.

Source: The SQL verb of the SQL statement.

Possible Values:

- CA-Alter table add column; AUSQRQTY=U
- CD-Alter table drop column; AUSQRQTY=U
- CL-Close cursor; AUSQRQTY=A
- CT-Create table; AUSQRQTY=U
- DC-Declare cursor; AUSQRQTY=A
- DE-Delete rows; AUSQRQTY=U
- DT-Drop table; AUSQRQTY=U
- FE-Fetch rows; AUSQRQTY=A
- IN-Insert row; AUSQRQTY=U
- OP-Open cursor; AUSQRQTY=A
- SE-Select rows; AUSQRQTY=A
- UP-Update rows; AUSQRQTY=U

Sample Value: SE

SEC.AUSQSQST

The first 128 characters of an SQL statement

Data Type: Character, read-only

Source: The SQL statement

Sample Value: DROP TABLE TABLE

SEC.AUSQSYNA

The list of system names to which the request is being sent. If the request is being sent to the local system, it may be empty or it may contain the words ALL or EXT.

Data Type: Character, read-only

Source: The system keyword on the SQL request

Sample Value: SYSA

SEC.AUSQTBLS

A word delimited list of table names referenced in the SQL statement. Host variable table names are resolved to their values. Most simple SQL statements have only one table reference that is usually a literal value. More complex SQL statements such as joins and sub-select clauses will reference more than one table.

Data Type: Character, read-only

Source: The SQL statement control block created by the SQL syntax compiler and the host variable value table.

Sample Value: STCTBL DB2TBL

Example: Read Host Variable Names and Values into Stem Variable Array

The following sample code shows how all the host variable names and values in a SQL statement can be read into a stem variable array for use in:

- Security authorization decisions
- Monitoring of critical table changes from sources outside the OPS/MVS address space.

The host variable names can also be created as simple REXX variables with the correct values.

```
hvnam.0=SEC.AUSQHVCN               /* Count of host variables */
Do ix=1 To hvnam.0                 /* Find all the host variables */
   SEC.AUSQHVIX=ix                 /* Set the host variable index */
   hvnam.ix=SEC.AUSQHVNA           /* Get the host variable name */
   hvval.ix=SEC.AUSQHVVL           /* Get the host variable value */
   vrc=VALUE(hvnam.ix,hvval.ix)  /* Create the Rexx variable */
End
```

OPAU Variables for SUBSYSDSN Security Events

The following variables are available for SUBSYSDSN security events:

SEC.AUSSDDNA

The ddname associated with the subsystem data set

Data Type: Character, read-only

Source: The ddname from the JCL

Sample Value: OMREPORT

SEC.AUSSPA01

The CA OPS/MVS subsystem name

Data Type: Character, read-only

Source: The first SUBSYS parameter from the JCL

Sample Value: OPSS

SEC.AUSSPA02

Identifies the interface that is opening the subsystem data set

Data Type: Character, read-only

Source: The second SUBSYS parameter from the JCL

Possible Values:

- OMEGAMON-OMEGAMON log data set interface
- OPSDSN-CA OPS/MVS product generic data set interface

Sample Value: OPSDSN

SEC.AUSSPA03

The characteristics of subsystem messages

Data Type: Character, read-only

Source: The third SUBSYS parameter from the JCL

Possible Values for OPSDSN:

- BLUE-OPSLOG messages are blue
- GREEN-OPSLOG messages are green
- PINK-OPSLOG messages are pink
- RED-OPSLOG messages are red
- TURQ-OPSLOG messages are turquoise
- WHITE-OPSLOG messages are white
- YELLOW-OPSLOG messages are yellow

Possible Values for OMEGAMON:

- CICS-Exceptions originated from OMEGAMON/CICS
- DB2-Exceptions originated from OMEGAMON/DB2
- IMS-Exceptions originated from OMEGAMON/IMS
- MVS-Exceptions originated from OMEGAMON/MVS

Sample Value: RED

SEC.AUSSPA04

A user-defined identifier to uniquely identify the source of messages

Data Type: Character, read-only

Source: The fourth SUBSYS parameter from the JCL

Sample Value: CICSA

OPAU Variables for System State Manager Security Events

The CA OPS/MVS OPSSMTBL command processor invokes security rules that have the following variables defined:

SEC.AUSTACTB

The name of the action table specified on the ACTION keyword of the OPSSMTBL command processor

Data Type: Character, read-only

Sample Value: TSOACTNS

SEC.AUSTFUCD

The function code returned by the OPSSMTBL command processor, which is one of the following:

- A (Add)
- C (Change)
- D (Delete)
- L (List)
- P (Post)

Data Type: Character, read-only

Sample Value: A

SEC.AUSTMDTB

The first letter of the mode of the table specified through the MODE keyword of the OPSSMTBL command processor. Mode types are INACTIVE, PASSIVE, ACTIVE, and NOPREREQ.

Data Type: Character, read-only

Sample Value: P (Passive)

SEC.AUSTNADW

The DOWN state of the table to be operated on, specified on the DOWN keyword of the OPSSMTBL command processor

Data Type: Character, read-only

Sample Value: DOWNSTAT1

SEC.AUSTNATB

The name of the table to be operated, specified on the ADD, CHANGE, DELETE, or LIST keyword of the OPSSMTBL command processor

Data Type: Character, read-only

Sample Value: MYTABLE

SEC.AUSTNAUN

The UNKNOWN state of the table to be operated on, specified on the UNKNOWN keyword of the OPSSMTBL command processor

Data Type: Character, read-only

Sample Value: UNKNOWN1

SEC.AUSTNAUP

The UP state of the table to be operated on, specified on the UP keyword of the OPSSMTBL command processor

Data Type: Character, read-only

Sample Value: UPSTATE1

SEC.AUSTOTCR

The name of the table created, if the OPSSMTBL command processor specified the name of a non-existent table in conjunction with an ADD operation

Data Type: Bit, read-only

Sample Value: 1 (create option specified)

SEC.AUSTSTTB

The name of the System State Manager resource directory table to be read or updated

Data Type: Character, read-only

Sample Value: SSM_MANAGED_TABLES

SEC.AUSTSYNA

The MSF system name to which the OPSSMTBL command is to be routed for execution

Data Type: Character, read-only

Sample Value: MSIX

OPAU Variables for OPSVIEW Security Events

The following variables are available for OPSVIEW security events:

SEC.AUSYCMSR

The OPSVIEW option 6 command string

Data Type: Character, read-only

Source: The contents of the COMMAND keyword of the OPSVIEW command

Sample Value: 'D TS,L'

SEC.AUSYFUCD

The OPSVIEW function code byte

Data Type: Character, read-only

Source: The OPSVIEW command

Note: This variable is not defined at this time.

SEC.AUSYJSNA

The primary JES name string

Data Type: Character, read-only

Source: The JESPJESN field of the JESCT

Sample Value: JES3

SEC.AUSYOTSR

The OPSVIEW option string

Data Type: Character, read-only

Source: The option entered with the OPSVIEW command

Sample Value: 3.2

SEC.AUSYSSNA

The subsystem name string

Data Type: Character, read-only

Source: The contents of the SUBSYS keyword of the OPSVIEW command

Sample Value: OPST

SEC.AUSYSYID

The remote system ID string

Data Type: Character, read-only

Source: The contents of the SYSID keyword of the OPSVIEW command

Sample Value: SYSA

OPAU Variables for USS Security Events

The following variables are available for ADDRESS USS security events:

SEC.AUUNFUCD

The server class of the request. Currently, the function code is always U for USS.

Data Type: Character, read-only

Source: The ADDRESS USS host command

SEC.AUUNUNCM

The first 255 bytes of the UNIX command string or CCS for z/OS API request keyword syntax

Data Type: Character, read-only

Source: The ADDRESS USS host command

Sample Values:

- UNIX Command: ps -a
- CCS for z/OS API Command: Node(UNIPC1) Text('Message for TNG pc')

Note: The case of the text can be mixed.

SEC.AUUNVERB

The verb name that indicates whether a UNIX command or a CCS for z/OS API command is sent to the OSF server

Data Type: Character, read-only

Source: The ADDRESS USS host command

Possible Values:

- USSCMD-UNIX System Services command
- WTO-Framework write-to-operator API
- WTOR-Framework write-to-operator with reply API
- REPLY-Framework reply to write-to-operator with reply API
- CMD-Framework command API
- DOM-Framework acknowledge message API
- PING-Framework ping network node API
- LOGOFF-Command to USS server to shut down
- TRACE-Command to USS server to toggle the command tracing parameter

Sample Value: USSCMD

OPAU Variables for CA Automation Point Security Events

The following variables are available for ADDRESS AP security events:

SEC.AUAPVERB

The ADDRESS AP command verb

Data Type: Character, read-only

Source: The ADDRESS AP host command

Possible Values:

- REXX
- NMFIND
- PPQ WRITE

Sample Value: REXX

SEC.AUAPCOMM

The first 255 bytes of the command text (truncated)

Data Type: Character, read-only

Source: The ADDRESS AP host command

Sample Value: REXX SYSTEM(APSYS1) PROGRAM(REXXPGM1)

SEC.AUAPSYSN

The 8-byte CA Automation Point system name as defined in MSF

Data Type: Character, read-only

Source: AP address environment command

Sample Value: MVSSY01

A security rule is required for all CA Automation Point function calls from CA OPS/MVS. For example,

```
)SEC AP*
)PROC
 CmdVerb = SEC.AUAPVERB
 Command = SEC.AUAPCOMM
 Uid = SEC.OPAUUSID

 say "Hello” Uid “using AP command verb: "CmdVerb
 say "Command is:" Command
 return “ACCEPT”
```

Debug an SEC Rule

To debug an SCR rule

1. Set the CA OPS/MVS BROWSESEC parameter to YES
2. Set the SEC event profile of your OPSLOG display to Y.

   This lets you view all SEC events.
3. With these parameters set, display the OPSLOG EVENT column.

   This lets you see recorded SEC events. This record will contain details for each security rule and you can use additional OPSLOG display columns to further interrogate the event.

More information:

Code and Debug AOF Rules (see page 59)

Examples: SEC Rules

The following security rule example executes for any TSO user attempting to access a RDF table. The rule has logic to only allow specific users to access specific tables.

```
)SEC SQL*
)INIT
/* The purpose of this security rule is to allow only specific users */
/* to update specific RDF tables especially SSM type tables.         */
/* Optionally, a GLOBAL.XX variable such as GLOBAL.SSM.USERS could   */
/* contain a list of authorized SSM users. This variable could be    */
/* used in all SSM SEC type rules and can be simply updated via      */
/* OPSVIEW option 4.8.            */
USERS = 'TSOUSR1 TSOUSR2 TSOUSR3 TSOUSR4'   /* List of allowed users */
TABLES = 'STCTBL DASD_TBL MAJORNODES_TBL'
)PROC
if POS(SEC.OPAUUSID,USERS) = 0 then
  do i = 1 to WORDS(TABLES)
    TBL = WORD(TABLES,I)
    if POS(TBL,SEC.AUSQSQST) > 0 then
      return 'REJECT'

  end
```

The following security rule demonstrates how to limit access to specific CA OPS/MVS facilities:

```
)SEC *
)PROC
/*********************************************************************/
/* Variable definitions :                                            */
/* TSOID - Set to current requestor (SEC.OPAUUSID)                   */
/* AUTHUSERS - Set to authorized users as set in global variable     */
/* OPSREQUEST - Set to the attempting OPSMAIN request                */
/*********************************************************************/

TSOID = SEC.OPAUUSID
AUTHUSERS = OPSVALUE('GLOBAL1.OPSMAINP.USERS','O')
OPSREQUEST = SEC.TYPE

/*********************************************************************/
/* Set the security error message variable and reject any user not   */
/* in the list of authorized users attempting to perform all         */
/* OPSMAIN requests EXCEPT viewing the OPSLOG.                       */
/*********************************************************************/

if POS(TSOID,AUTHUSERS) = 0 & OPSREQUEST ¬= 'OPSBRW' then
  do
    ERRMSG = 'Unauthorized to issue OPSMAIN request -'OPSREQUEST
    SEC.OPAUERMG = ERRMSG
    return 'REJECT'
  end
```

Time Limit-Exceeding Rules

Time limit-exceeding (TLM) rules provide the ability to intercept events that exceed a time limit when either processor usage or continuous wait time limits for a batch job or address space are exceeded. An extension to the exceeded time limit may be granted to prevent the usual termination of the job or address space. The major benefit of TLM rules is the ability to write the equivalent of an IEFUTL SMF exit in OPS/REXX, rather than assembly language, and to use the power of OPS/REXX facilities for making time limit extension decisions.

Example: TLM Rule

Assume that a privileged set of TSO users are not subject to the SMF continuous wait time limit when they leave their sessions unattended. A TLM rule may be installed to extend their session limits each time the wait time limit is exceeded up to a maximum of ten times.

```
)TLM *
)INIT
/* Create a STATIC variable with a list of privileged TSO users.*/
/* This variable will be compared against the TLM.JOBNAME TLM   */
/* environmental variable which indicates who just exceeded the */
/* limit.                                                       */
GoodGuys = 'OPER1 SYSPRG1 NETPRG1'
)PROC
/* Verify data from tlm event variables that this is a TSO user */
/* that has exceeded the defined wait limits. If it is, then    */
/* check to see if it is one of the users in the good_guys list */
/* and increase limit if we haven't done it 10 times yet.       */
if TLM.LIMIT = 'WAIT' & TLM.SUBSYS = 'TSO' & ,
 WORDPOS(TLM.JOBNAME,GoodGuys) > 0 & ,
 TLM.WAITCOUNT < 10 then ,
  do
    TLM.EXTEND = 3600                /* Extend session 1 hour  */
    return "EXTEND"
  end
return "NORMAL"                      /* Let system cancel them */
)END
```

Installation Requirements for TLM Rules

Set the parameters INITSMF and TLMRULES to YES.

The installation IEFUTL SMF exit must be implemented.

Note: For more information, see the *CA OPS/MVS Parameter Reference*.

)TLM—Event Specifier of TLM Rules

The following is the format for coding the TLM event definition section:

```
)TLM jobnamespec
```

jobnamespec

Specifies the job name. Follow these guidelines when specifying the character string:

- Specify one to eight characters of the job name.
- The string cannot contain embedded blank spaces. You can use the wildcard (*) character. For example,
  - CICS* matches CICSA, CICSABC, CICS123 and any other job name containing a CICS prefix.
  - CICS*05 matches CICSD05, CICS205, CICS1105, and so on.
  - *05 matches any job name ending with 05.
  - * alone matches all job names.
- Lowercase letters are acceptable, but the AOF converts them to uppercase for event testing.

Initialization, Processing, and Termination Sections of TLM Rules

The general guidelines for coding the initialization, processing, and termination sections and the various AOF tools that you can use, apply to TLM rules.

More information:

AOF Rule Structure (see page 21)
AOF Rule Tools (see page 29)

RETURN Statements in the)PROC Section of a TLM Rule

The OPS/REXX RETURN statement must specify one of the following values:

NORMAL

Returns without granting a time limit extension. Unless another IEFUTL SMF exit in the system grants an extension, the system terminates the job or address space.

EXTEND

Extends the time limit exceeded by the number of seconds in the TLM.EXTEND variable.

Default: RETURN NORMAL

The return values listed here are character *constants* rather than keywords. An unrecognized return value, for example, a misspelled value, defaults to a value of NORMAL.

If multiple TLM rules are active, and more than one return EXTEND is performed, then the last non-zero value specified for variable TLM.EXTEND is used as the time extension seconds value.

To nullify the extension granted by a prior TLM rule, a rule may set TLM.EXTEND to zero and return NORMAL.

Execution Considerations for TLM Rules

The processing section of a rule that responds to a TLM event executes in the address space that exceeded the time limit. Therefore, any type of logic that could possibly suspend the processing of a TLM rule should be performed by scheduling an OPS/REXX program to a CA OPS/MVS OSF TSO, TSL, or TSP server.

The active JSCB in the ending address space is the region control task, or the initiator. This causes the ACCOUNT, EXECPGM, and MODULE operands of OPSINFO to return the values for the RCT or initiator program, rather than the application program that is ending.

The AOF execution limits apply to the processing section of a rule that responds to a time-limit-exceeding event.

More information:

Building and Controlling AOF Rules (see page 49)
Code and Debug AOF Rules (see page 59)

OPS/REXX Host Environments in the)PROC Section of a TLM Rule

The)PROC section of a TLM rule has the following host environments with the following TLM rule characteristics. The AOFDEFAULTADDRESS parameter specifies the default host environment for TLM rules.

ADDRESS AOF

Sent to CA OPS/MVS. Does not wait. Output is not returned.

ADDRESS AP

Sent to MSF and then forwarded to the CA Automation Point system. Does not wait. Output is not returned.

ADDRESS EPI

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS HWS

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS ISPEXEC

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS LXCON

The VM and Linux command requests sent to USS server for execution. Does not wait. Output is not returned. The List request runs inline and returns VM and Linux system data in stem variables.

ADDRESS MESSAGE

Sent as a WTO. The AOFDEST parameter specifies the destination.

ADDRESS MQ

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS NETMAN

Sent to a CA OPS/MVS internal CA Netman request queue to issue MGPT commands. Does not wait. Output is not returned.

ADDRESS NETMASTR

Sent to CA NetMaster NM for SNA on the local system. Does not wait. Output is not returned.

ADDRESS OPER

Sent to target console as specified through the OCCONSOLENAME parameters when no CONNAME operands are present. Output is not returned. Schedule an OPS/REXX program in a server when a command output interrogation is needed.

ADDRESS OPSCTL

Sent to a specified facility. If the facility is ECF or OSF, does not wait. If the facility is MSF, slight wait occurs. The external data queue returns the output.

Note: If the command is MFS LIST, no wait occurs.

ADDRESS OPSDYNAM

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS OSF

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSO servers.

ADDRESS OSFTSL

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSL servers.

ADDRESS OSFTSP

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSP servers.

ADDRESS SOF

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS SQL

Does not wait. Processed synchronously for requests that can be satisfied on the local system. Output is returned in stem variable. Error messages, if any, are returned to an external data queue.

ADDRESS SYSVIEWE

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS TSO

Sent to an OSF TSO server. Does not wait. Output is not returned.

ADDRESS USS

Sent to a USS server. Does not wait. Output is not returned. Schedule an OPS/REXX program in a server if a command output interrogation is needed.

ADDRESS WTO

Does not wait. Output is sent to specified (or default) destination. When attempting a WTOR, host command is sent to a TSO server for execution. The response is then returned to the server. Schedule an OPS/REXX program in a server if the WTOR response interrogation is needed.

More information:

Code and Debug AOF Rules (see page 59)

AOF Variables Available in a TLM Rule

You can use all AOF variable types in TLM rules, as described in the chapter "AOF Rule Tools (see page 29)." You can use the following unique AOF event variables in the)PROC section of a TLM rule, and you can manually interrogate the corresponding OPSLOG display field as an aid in debugging or implementing rule logic.

TLM.COLOR

The color that the message text will have in OPSLOG browse

Data Type: 1-byte binary (unprintable), read/write

Sample Value: '00'X

Note: Use the OPSCOLOR function of OPS/REXX to set the TLM.COLOR variable.

OPSLOG Browse Column: COLOR

TLM.CPUCOUNT

The number of extensions for CPU time-limit-exceeding that has been granted thus far by TLM rules. This variable is incremented each time a non-zero extension is granted by TLM rules.

Data Type: Integer, read-only

Sample Value: 3 (3 extensions granted)

TLM.CPUSECS

The number of CPU seconds that have been granted thus far by TLM rule extensions. Each time a non-zero extension is granted by TLM rules, the number of seconds is added to this variable.

Data Type: Integer, read-only

Sample Value: 600 (10 minutes of extension)

TLM.EXTEND

The number of CPU or elapsed time seconds to add to the CPU or wait time limit to allow the job to continue. If this value remains zero, the job is canceled by the system.

Data Type: Integer, read/write

Sample Value: 120 (2 minute extension)

TLM.JOBNAME

The name of the job that has exceeded the CPU/wait time limit taken. JOBNAME is taken from JMRJOB in the JMR control block

Data Type: Character, read-only

Sample Value: IBMUSER

OPSLOG Browse Column: JOBNAME

TLM.JMRADDR

The address of the JMR control block passed to IEFUTL SMF exit. This address may be used with the OPSTORE function of OPS/REXX to access any field in the JMR to obtain data that is not provided by the TLM event variables. The IBM macro IEFJMR maps the contents of the JMR.

Data Type: 4-byte (unprintable), binary

Sample Value: '00ABC004'X

TLM.LIMIT

The type of time limit that has been exceeded by this job

Data Type: Character, read-only

Possible Values:

- JOB-Job processor time limit exceeded
- STEP-Step processor time limit exceeded
- WAIT-SMF continuous wait time limit exceeded
- **Sample Value:** WAIT (TSO user went home without logging off)

TLM.RDRDATE

The date on which the system input reader recognized the JCL JOB statement for this job. The value is seven digits long, with the high order four digits set to the year, and the low order three digits set to the day of the year.

Data Type: Character, read-only

Sample Value: 2003023

TLM.RDRTIME

The time at which the system input reader recognizes the JCL JOB statement for this job. The time value is in hundredths of seconds since midnight.

Data Type: Character, read-only

Sample Value: 3976655

TLM.SUBSYS

The subsystem name of the job used by SMF for workload accounting. Subsystem names are defined in the SMFPRMxx member of parmlib and extracted from the OUCBSUBN field of the OUCB control block.

Data Type: Character, read-only

Sample Value: TSO

TLM.TEXT

The OPSLOG message text that describes the time-limit-exceeding event for the above JOBNAME

Data Type: Character, read-only

Sample Value: IBMUSER EXCEEDED WAIT TIME LIMIT

OPSLOG Browse Column: Text is always displayed

TLM.USER

An 8-byte variable providing communication between rules executing for the same TLM event. The variable can contain any installation data that these rules need, and it can store a character string displayable through OPSLOG Browse.

Data Type: User-defined, read/write

Notes:

- Before AOF processing, this variable is initialized to binary zeros. It is then passed to each rule that executes for the same TLM event; each rule can look at or change the variable contents before passing the variable to the next rule for the TLM event.
- The primary purpose for the USER variable is to provide a method to pass a small amount of data between the rules. This data may be binary or mixed case. The USER field may also be used for filtering in the OPSLOG. However, USER data used for OPSLOG filtering must be uppercase and displayable.

OPSLOG Browse Column: USER

TLM.USERCOM

The value contained in the JMRUCOM of the JMR control block. This field is sometimes used to point to tables or control blocks used by installation SMF exits.

Data Type: 4-byte binary (unprintable), read-only

Sample Value: '0A002CFC'X

Note: Use the OPSTORE function of OPS/REXX to access any storage pointed to by TLM.USERCOM.

TLM.WAITCOUNT

The number of extensions for wait time limit-exceeding that has been granted thus far by TLM rules. This variable is incremented each time a non-zero extension is granted by TLM rules.

Data Type: Integer, read-only

Sample Value: 3 (3 extensions granted)

TLM.WAITSECS

The number of wait time seconds that have been granted thus far by TLM rule extensions. Each time a non-zero extension is granted by TLM rules, the number of seconds is added to this variable.

Data Type: Integer, read-only

Sample Value: 600 (10 minutes of extension)

Debug a TLM Rule

Debug a TLM rule to intercept and review events that exceed a time limit.

To debug a TLM rule

1. Set the CA OPS/MVS BROWSETLM parameter to YES
2. Set the TLM event profile of your OPSLOG display to Y.

   This setting lets you view all TLM events.
3. With these parameters set, display the OPSLOG EVENT column.

   This lets you see recorded TLM events. The column contains details for each time limit-exceeding rule.

   You can use additional OPSLOG display columns to further interrogate the event.

More information:

Code and Debug AOF Rules (see page 59)

Time-of-Day Rules

Time-of-day (TOD) rules let you schedule automation to perform system tasks that need to be done at specific times or time intervals. Additionally, you can use TOD rules to initiate proactive automation, such as probing critical applications to validate response times or interrogating system resource usage, thus determining potential problems before they occur.

Installation Requirements for TOD Rules

If you are using the CATCHUPYES or CATCHUPMAN TOD qualifiers as described next, you must have the SYSCHK1 DD allocated to your OPSMAIN task.

Note: For more information on allocating this linear VSAM file for the SYSCHK1 DD, if it was not performed during your original installation of CA OPS/MVS, see the *Administration Guide*.

)TOD—Event Definition Section of TOD Rules

Every TOD rule requires the presence of at least one event specifier, with a maximum of ten. The first event specifier may be coded on the)TOD event definition line, as shown in the example above, or it may be coded on its own line following the event definition line. Additional specifiers, if any, must be coded one per line, following the first specifier. Start-time, End-time, Interval, and Max Execs can be declared independently.

Options follow the first event specifier and must be entered in the order shown in the previous code sample. Each of the following options controls the operation of the whole rule, not just the first event specifier.

Use this format for coding the time-of-day event definition section:

```
)TOD  EventSpecifier1,[MSGALLOW],[CATCHVAL],[SYNCH]
      EventSpecifier2
…
      EventSpecifier10
```

EventSpecifier

Describes when and how often a TOD rule should execute.

An event specifier has four elements: Start-time, Interval, End-time, and MaxExecs. A comma separates each element.

Start-time and End-time

(Optional) Start-time and End-time are both defined as a *todspec*, which may be written in one of several formats. For example, a *todspec* can specify a day of the week, a specific date and time, or a delay after the rule is enabled. End-time must be specified in the same *todspec* format as Start-time. It is not valid to specify a day of the week as a Start-time and then specify a specific date as an End-time.

Note: Start-time and End-time are optional. You may specify either or both.

Days of the week begin on Sunday at 00:00 and end on Saturday at 23:59. It is invalid to specify a Start-time that is later than an End-time. For example, you cannot code a rule to begin execution on or before Saturday and end on or after Sunday.

Note: Rules with start and end times coded as dates, rather than days of the week, are not subject to this restriction. Code two event specifiers for the rule, one ending on Saturday and another starting on Sunday, to avoid this limitation.

Interval

This element specifies the frequency of rule executions, once Start-time has been satisfied and before End-time is reached. If *interval* is omitted, the rule executes only once at Start-time. *Interval* is the amount of time, that is, the number of specified time units that the AOF waits between rule executions. It is expressed as:

```
n interval
```

Following is an example of the frequency and type of an *interval*:

- *n*-An integer multiplier indicating the number of *interval* time units
- *interval*-One of the following time units: DAY(S), WEEK(S), HOUR(S), MINUTE(S) or MIN(S), SECOND(S) or SEC(S)

For example: 2 HOURS, 30 SECS, 1 MIN

MaxExecs

This element is an integer that specifies the maximum number of times a rule executes for the event specifier. The event specifier no longer triggers the rule once this limit is reached.

MSGALLOW

(Optional) Controls the OPS3900O message, which is an audit trail indicating the next TOD setting for a rule. *Msgallow* may be set to MSG to allow the message, or NOMSG to disallow the message. The default is MSG. Do not specify the *Msgallow* qualifier more than once in a rule if the TOD rule contains more than one *todspec* line.

CATCHVAL

(Optional) Determines whether catch-up processing occurs for the rule. Catch-up processing allows or disallows a TOD rule to execute if CA OPS/MVS was not active during its scheduled executing time. Do not specify the *catchval* qualifier more than once in a rule if the TOD rule contains more than one *todspec* line. Valid values are:

- CATCHUPYES - tells CA OPS/MVS to perform catch-up processing for this rule.
- CATCHUPNO - tells CA OPS/MVS not to perform catch-up processing for this rule. This is the default.
- CATCHUPMAN - tells CA OPS/MVS to ask the operator whether this rule requires catch-up processing.

Notes:

- Timely operator response to a CATCHUPMAN WTOR can influence the executing of subsequent TOD rules (all of which run under the task that is waiting for operator response).
- Catch-up applies to all event specifiers for the rule in which it is specified.
- Catch-up executing is not considered when CA OPS/MVS computes whether a rule has reached the maximum execute-count limit.
- Neither CATCHUPYES nor CATCHUPMAN may be specified on the TOD rule specification when enabling a dynamic TOD rule. This causes a syntax error and the rule is not enabled.

SYNCH

(Optional) Determines whether TOD rule execution is synchronized on a rounded interval boundary. Do not specify the *synch* qualifier more than once in a rule if the TOD rule contains more than one *todspec* line. Valid values are:

- SYNCH - Indicates rule execution is synchronized. This is the default.
- NOSYNCH - Indicates rule execution is not synchronized.

Note: The SYNCH value does not apply to TOD rules that specify a time to execute after it is enabled (that is, rules in which the *+*nn value* format is used). These rules always execute with a NOSYNCH value.

Defining the Todspec

You can specify each *todspec* for TOD rules in the format of a date or time. In addition, you can instruct a TOD rule to execute at a specified interval after it is enabled. The following syntaxes are available for specifying a date or time *todspec* value in TOD rules:

Date

The day, month, year, or day of the week (depending on the format you use to specify the date):

- *dd*-A two-digit integer (01 through 31) corresponding to a day of the month
- *MMM*-One of the following three-character abbreviations for a month: JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC
- *year*-A four-digit year (for example, 2002)
- *mm*-A two-digit integer (01 through 12) corresponding to a month of the year
- *weekday*-The full name of a weekday (for example, SUNDAY, MONDAY)

Note: In TOD rules, the week starts on Sunday at 00:00, which implies that Saturday follows Sunday. Start-time must be before End-time; therefore, it is invalid to start a rule on Saturday and end it on Sunday if you use the *weekday* format. For an example, see Start/end order in Rules Governing the Coding of TOD Event Specifiers in this chapter.

Format (any of the following):

- *dd MMM year*
- *year/mm/dd*
- *weekday*

Time

The time in 24-hour military format, as follows:

- *hh*-A two-digit integer (00 through 23) indicating the hour
- *mm*-A two-digit integer (00 through 59) indicating the minutes after the hour
- *ss*-A two-digit integer (00 through 59) indicating the seconds after the minute. This value is optional.

Format: *hh*:*mm*[:*ss*]

Start at some interval after rule is enabled

This type of *todspec* is commonly used in dynamically created TOD rules (explained in the chapter “Building and Controlling AOF Rules”) to initiate automation that needs to be performed at some time interval after a specific event occurs, as follows:

- *nn*-a number greater than 0
- *value* -DAY(S), WEEK(S), HOUR(S), MINUTE(S) or MIN(S), or SECOND(S) or SEC(S). For example, the following rule executes 1 minute after it is enabled and every 30 seconds thereafter, until it executes a total of three times:

```
)TOD *+1 MINUTE,30 SECONDS,,3
```

Note: This is a special type of TOD rule SPEC that is only eligible to be used one time per TOD rule.

Format: *+*nn value*

Rules Governing the Coding of TOD Event Specifiers

The following rules govern the coding of TOD event specifiers:

- The number of event specifiers that may be coded is 1 to 10.
- The number of event specifiers allowed per line is 1.
- Uppercase or lowercase letters are acceptable.
- Blank lines are permitted between time event specifiers.
- Any number of blanks is permitted between qualifiers.
- An event specifier may begin in any column, except on the)TOD line.
- An event specifier may use the entire line (72 characters).
- Because TOD rules allow multiple spec lines, a section header must follow them before comments can be used. Either a)INIT card must be inserted prior to the comment card or the comment must be moved after the)PROC card.

- If your Start-time and Stop-time are exact dates, for example 03 JUN 2002 and 04 JUNE 2002, the End-time must be in the future. The rule does not enable if End-time has already expireTime defaults:

  If the starting date/time specification contains a time, and the ending date/time specification does not, the time part of the starting date/time specification is used for the ending date/time specification. For example, given the following starting and ending specifications, the rule would execute Monday, Tuesday, Wednesday, Thursday, and Friday at 08:00 on each day:

  ```
  )TOD MONDAY 8:00, DAY, FRIDAY
  ```

  If the *Start-time* contains only a time value, with an interval unit of HOURS, MINS, or SECS, and no *End-time* is specified, then the default *End-time* would be midnight each day. For example, given the following TOD specifications, the rule would execute starting at 15:00 every day, and then execute every 5 minutes until midnight:

  ```
  )TOD 15:00,5 MINS
  ```

  If the *Start-time* contains a day of the week value, with an interval unit of HOURS, MINS, or SECS, and no *End-time* is specified, then the default *End-time* would be Sunday at 00:00:00. For example, given the following TOD specifications, the rule would execute starting on Tuesday 08:00, and then execute every 1 hour until Sunday at 00:00:00 and resume again on Tuesday at 08:00:

  ```
  )TOD TUESDAY 08:00,1 HOUR
  ```

  If the TOD specifier contains just a starting specification that contains a day of the week and time value, then the default *End-time* would be on a week boundary. For example, given the following TOD specifier, the rule would execute every Friday at 06:00:00:

  ```
  )TOD FRIDAY 06:00:00
  ```

- Completeness and order of date and time: date/time specifications may omit the date or the time, and they may be coded in any order; use a blank to separate the date and the time, as shown in the following example:

  ```
  21 JAN 2002 1:20
  ```

  Omitting the executing date causes the rule to execute every day. If the time is omitted, CA OPS/MVS assumes it to be 00:00:00, midnight (there is an exception, which is noted in the next rule).

- Start/end order: the starting date/time specification must indicate a point in time that is before the ending date/time specification. The following is an example that will **not** enable because the AOF week starts on Sunday and ends on Saturday:

  ```
  )TOD FRIDAY,DAY,MONDAY
  ```

  The correct method to create a TOD rule that executes at midnight on Friday, Saturday, Sunday, and Monday is:

  ```
  )TOD    FRIDAY,DAY,SATURDAY
          SUNDAY,DAY,MONDAY
  ```

- Format compatibility (date/time): the ending date/time specification must be the same type as the starting date/time specification; for example, you cannot code:

  ```
  )TOD MONDAY,,2002/10/14
  ```

  because the ending date/time specification is in day-of-year format while the starting date/time specification is in day-of-week format.

- Although you can specify the *msgallow*, *catchval,* and *synch* options only once in a rule, you may specify them on any line of the TOD-event definition section. You do not have to specify these three qualifiers on the same line with each other; you may specify them in any combination you wish. The values of these qualifiers apply to all event specifier values in that rule.

  Note: CA recommends that you code these options on the first event specifier.

Initialization, Processing, and Termination Sections of TOD Rules

The general guidelines for coding the initialization, processing, and termination sections and the various AOF tools that you can use, apply to TOD rules.

More information:

AOF Rule Structure (see page 21)
AOF Rule Tools (see page 29)

RETURN Statements in the)PROC Section of a TOD Rule

The RETURN statement has no special meaning in the processing section of a TOD rule.

Execution Considerations for TOD Rules

The processing section of a TOD rule executes in the CA OPS/MVS main address space. Therefore, various OPS/REXX host environments that can cause waits to occur, such as issuing z/OS commands through the ADDRESS OPER **and** collecting the command output, can possibly suspend the main address space. The simple rule of thumb to follow is to keep the logic simple in a TOD rule, and any complex logic or interactive logic that may cause a wait to occur should be done in an OPS/REXX program that gets triggered to an OSF TSO server on behalf of the TOD rule.

The AOF execution limits apply to the processing section of a rule that responds to a security event.

More information:

Building and Controlling AOF Rules (see page 49)
Code and Debug AOF Rules (see page 59)

OPS/REXX Host Environments in the)PROC Section of a TOD Rule

The)PROC section of a TOD rule has the following host environments with the following TOD rule characteristics. The AOFDEFAULTADDRESS parameter specifies the default host environment for TOD rules.

ADDRESS AOF

Sent to CA OPS/MVS. Does not wait. Output is not returned.

ADDRESS AP

Sent to MSF and then forwarded to the CA Automation Point system. Does not wait. Output is not returned.

ADDRESS EPI

Runs inline. Waits for output in the external data queue.

ADDRESS HWS

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS ISPEXEC

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS LXCON

Runs inline. Waits for output that is returned in stem variables.

ADDRESS MESSAGE

Sent as a WTO. The AOFDEST parameter specifies the destination.

ADDRESS MQ

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS NETMAN

Sent to a CA OPS/MVS internal CA Netman request queue to issue MGPT commands. Does not wait. Output is not returned.

ADDRESS NETMASTR

Sent to CA NetMaster NM for SNA on the local system. Does not wait. Output is not returned.

ADDRESS OPER

Sent to target console as specified through the OCCONSOLENAME parameters when no CONNAME operands are present. Output is not returned. Schedule an OPS/REXX program in a server when a command output interrogation is needed.

ADDRESS OPSCTL

Sent to the specified facility. If the facility is ECF or OSF, it does not wait. If the facility is MSF, a slight wait occurs. Output is returned to the external data queue.

Note: If the command is MSF LIST, no wait occurs.

ADDRESS OPSDYNAM

Runs Inline. Returns output in variables.

ADDRESS OSF

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSO servers.

ADDRESS OSFTSL

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSL servers.

ADDRESS OSFTSP

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSP servers.

ADDRESS SOF

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS SQL

Does not wait. Processed synchronously for requests that can be satisfied on the local system. Output is returned in stem variable. Error messages, if any, are returned to the external data queue.

ADDRESS SYSVIEWE

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS TSO

Sent to an OSF TSO server. Does not wait. Output is not returned.

ADDRESS USS

Runs Inline. Waits for output in stem variables.

ADDRESS WTO

Does not wait. Output is sent to the specified console. If you attempt a WTOR, it runs inline and then the WTOR waits for a response in the external data queue.

More information:

Code and Debug AOF Rules (see page 59)
Execution Considerations for TOD Rules (see page 335)

AOF Variables Available in a TOD Rule

You can use all AOF variable types in TOD rules, as described in the chapter "AOF Rule Tools (see page 29)." You can use the following unique AOF event variables in the)PROC section of a TOD rule, and you can manually interrogate the corresponding OPSLOG display field as an aid in debugging or implementing rule logic.

TOD.CATCHUP

A value indicating whether a TOD rule is executing as usual or in catch-up mode

Data Type: Character, read-only

Possible Values:

- Y-The rule is executing in catch-up mode
- N-The rule is executing as usual

Sample Value: Y

Notes:

- The TOD.CATCHUP variable is similar to the Automate &CATCHUP environmental variable.
- A rule is executing in catch-up mode when it first executes at CA OPS/MVS initialization and either:
  - The value of the *catchup* optional TOD qualifier is CATCHUPYES.
  - The value of the *catchup* optional TOD qualifier is CATCHUPMAN, and the reply of the operator to the WTOR concerning CATCHUPMAN rules is YES.

TOD.FIREMISSED

The TOD.FIREMISSED variable contains 5 pieces, or words, of information. The information in these words depends on whether the rule that is currently executing is executing as a result of catch-up processing:

- If the rule is executing as a result of catch-up processing, TOD.FIREMISSED contains information about the last time the rule should have executed but did not.
- If the rule is not executing as a result of catch-up processing, TOD.FIREMISSED contains information about the current execution of the rule.

Data Type: Character, read-only

Possible Values: This list explains the contents of the 5 words:

- Word 1-If the rule is executing in catch-up mode, this is the date when the rule was last scheduled to execute but did not, in the form *yyyy/mm/dd*. If the rule is not executing in catch-up mode, this is the current date, in the format *yyyy/mm/dd*.
- Word 2-If the rule is executing in catch-up mode, this is the time when the rule was last scheduled to execute but did not, in the form *hh:mm:ss*. If the rule is not executing in catch-up mode, this is the current time, in the format *hh:mm:ss*.
- Word 3-If the rule is executing in catch-up mode, this is the date when the rule was last scheduled to execute but did not, in Julian format: *yyyyddd*. If the rule is not executing in catch-up mode, this is the current date, in Julian format: *yyyyddd*.
- Word 4-If the rule is executing in catch-up mode, this is the time, calculated as seconds since January 1, 1980 (the date when IBM introduced the PC), when the rule was last scheduled to execute but did not. If the rule is not executing in catch-up mode, this is the current time, calculated as seconds since January 1, 1980.
- Word 5-If the rule is executing in catch-up mode, this is the day of the week (SUN, MON, TUE, WED, THU, FRI, SAT) when the rule was last scheduled to execute but did not. If the rule is not executing in catch-up mode, this is the current day of the week.

Sample Value: 2002/08/25 14:29:00 2002237 493828140 FRI

Note: The TOD.FIREMISSED variable was developed to support the Automate &CDATE, &CDAY, &CTIME, &CJULDATE, and &CCLOCK environmental variables.

TOD.NEXTFIRE

A value indicating the next time a rule will execute

Data Type: Character, read-only

Possible Values:

- The date and time the rule will execute, in *yyyy/mm/dd hh:mm:ss* format
- NONE if the rule will not execute again

Sample Value: 2002/08/25 14:30:00

Note: The next execute date in automateable message OPS3900O is in the *yyyy/mm/dd* format. For example:

```
OPS3900O RULE 0.TODTEST FOR TOD 2002/08/25 14:30 SET
```

TOD.USER

An 8-byte variable providing communication between rules that execute for the same TOD event. The variable can contain any installation data that these rules need, and it can store a character string displayable through OPSLOG Browse.

Data Type: User-defined, read/write

Notes:

- Before AOF processing, this variable is initialized to binary zeros.
- The CA OPS/MVS Test Facility shows TOD events both before and after AOF processing.
- OPSLOG Browse does not show TOD events unless one of these conditions is true:
  - The CA OPS/MVS BROWSETOD parameter is set to YES, allowing the event to appear in OPSLOG Browse *before* AOF processing.
  - The CA OPS/MVS RULETRACE parameter is set to ON, allowing the event to appear in OPSLOG Browse *after* AOF processing.

For descriptions of the BROWSETOD and RULETRACE parameters, see the *Parameter Reference*.

OPSLOG Browse Column: USER

Debug a TOD Rule

For a discussion on debugging techniques that you can use in all AOF rules, see the chapter "Code and Debug AOF Rules (see page 59)."

Examples: TOD Rules Event Specifiers

This section contains examples of various TOD rule event specifiers. In the following examples, a week runs from Sunday to Saturday.

- Example 1: This rule executes every day of the week, every five minutes, on a five-minute boundary from midnight:

  ```
  )TOD ,5 MIN
  ```

- Example 2: This rule executes every day, every five minutes, on a five-minute boundary from the time the rule is enabled:

  ```
  )TOD ,5 MIN,,,,,NOSYNCH
  ```

- Example 3: This rule executes every 15 minutes on Friday and Saturday:

  ```
  )TOD FRIDAY,15 MIN
  ```

- Example 4: This rule executes every hour, Monday through Friday:

  ```
  )TOD MONDAY,1 HOUR,SATURDAY
  ```

- Example 5: This rule executes every two hours on Saturday and Sunday:

  ```
  )TOD SUNDAY,2 HOURS,MONDAY
       SATURDAY,2 HOURS
  ```

- Example 6: This rule executes every half hour from 12:00 to 15:00, every day:

  ```
  )TOD 12:00,30 MIN,15:00
  ```

- Example 7: This rule executes every day at 8:00 for the next seven days:

  ```
  )TOD 08:00,,,7
  ```

- Example 8: This rule executes every day at 01:00 and it will catch up if CA OPS/MVS is down:

  ```
  )TOD 01:00,,,,,CATCHUPYES
  ```

- Example 9: This rule executes every hour, on the hour, from 8:00 December 25 through 7:00 December 26:

  ```
  )TOD 25 DEC 2002 08:00,1 HOUR,26 DEC 2002 08:00
  ```

Examples: Complete TOD Rules

- Example 1:

```
)TOD 02:00,,,,,CATCHUPYES
)PROC
/**********************************************************/
/* This TOD rule will fire 2 AM every day to simply      */
/* set up the initiators to handle the backup period.    */
/**********************************************************/

/**********************************************************/
/* Send information message to sysplex master console    */
/* to indicate that we are configuring initiators using  */
/* the OPS/REXX ADDRESS WTO host environment. Use the    */
/* OPS/REXX OPSINFO function to obtain the name of the   */
/* sysplex master console.                               */
/**********************************************************/
CONSOLE=OPSINFO('MSTCONSNM')    /* GET CURRENT PLEX MSTR */
address WTO
"MSGID(OPSAUTO5) TEXT('INITS CONFIGURED FOR BACKUPS')",
   "CNNAME("CONSOLE")"

/**********************************************************/
/* Issue commands to set initiators using OPS/REXX       */
/* ADDRESS OPER host command environment.                */

/**********************************************************/
address OPER
"COMMAND($TI1-10,ABC) NOOUTPUT"
"COMMAND($TI11-20,EFG) NOOUTPUT"
```

- Example 2: The following two rules demonstrate how to use dynamic TOD rules to initiate a programmatic action at a time after an event occurs. The first rule creates the TOD rule and the second rule deletes the dynamic TOD rule.

  For details about the logic, see the comments in the rules.

```
)MSG DFHPA1108
)PROC
/**************************************************************/
/* This rule will fire on a very early CICS initialization    */
/* message and simply create a dynamic TOD rule to fire 30    */
/* minutes from now. The JOBNAME of this CICS region will     */
/* be used as the name of the TOD rule allowing these monitor */
/* rules to perform this check for all CICS regions. the logic*/
/* of the TOD rule is to simply issue a notification message  */
/* to the local master console that indicates that a CICS did */
/* not initialize. This TOD rule will be deleted on the       */
/* initialization message, meaning that the region is OK and  */
/* thus no message will be issued.                            */
/* DFHPA1108  A04IC4SL DFHSIT6$ HAS BEEN LOADED.              */
/**************************************************************/
REGION = MSG.JOBNAME        /* GET THE REGION NAME OF MESSAGE */
/* Queue the text of the dynamic TOD rule to the EDQ          */

queue ")TOD *+30 MINS "
queue ")PROC"
queue "CONSOLE=OPSINFO('LOCMSTCONSNM')"
queue "address WTO"
queue "'MSGID(ALERT:) TEXT(''CICS "REGION" HAS NOT INITIALIZED'')',"
queue "'CNNAME('CONSOLE') DESC(2)'"
/* ENABLE THE DYNAMIC TOD RULE WITH NAME OF THE REGION        */
address AOF "ENABLE *DYNAMIC."REGION


)MSG DFHSI1517
)PROC
/**************************************************************/
/* This rule will delete dynamic TOD rule that was ENABLEd for*/
/* this region during startup, thus indicating all is well.   */
/*   DFHSI1517 A04IC4S1 Control is being given to CICS.       */
/**************************************************************/
REGION = MSG.JOBNAME        /* GET THE REGION NAME OF MESSAGE */
MSG.TEXT=TRANSLATE(MSG.TEXT)  /* UPPER CASE THE MESSAGE       */
/* VERIFY THAT THIS REALLY IS THE CONTROL GIVEN TO MESSAGE    */
if POS('CONTROL IS BEING GIVEN TO CICS',MSG.TEXT) > 0 then
address AOF "DISABLE *DYNAMIC."REGION
```

UNIX System Services Rules

UNIX System Services (USS) message rules allow you to write automation procedures for messages that originate from CCS for z/OS.

Message sources in the CCS for z/OS include:

- The USS SYSLOG daemon
- Locally generated CCS for z/OS messages
- CA NSM messages that are forwarded to the CA Event Manager component by other platforms and systems (such as Windows and UNIX servers)

A rule responds to a USS message event when *one* of the following occurs:

- A local USS application writes a message to the USS syslog.
- A message is created by or forwarded to CCS for z/OS on the local system.

A special USS event type is also provided in USS AOF rules to monitor the creation and termination of every USS process. These events are created using the USS dynamic system exits which are activated by setting the CA OPS/MVS parameter INITUSSPROC to a value of YES at product initialization.

- A USS process event message with the fixed message ID of USSPROCBEG is generated for the creation of every new USS process.
- A USS process event message with the fixed message ID of USSPROCEND is generated for the termination of every USS process.
- An extended set of USS AOF rule variables is defined for USS process events in order to provide detailed process information not normally available in conventional USS message rules.

Installation Requirements for USS Rules

The CCS for z/OS must be active on the system.

The optional USS component of CA OPS/MVS must be installed.

Set the parameters INITUSS and USSRULES to YES.

Installation Requirements for USS Process Event Rules

The USS process event rule feature does not require any CCS components or the CA OPS/MVS USS server components. The INITUSSPROC parameter controls the installation of the CA OPS/MVS dynamic system exit routine for USS process events while the USSPROCRULES and BROWSEUSSPROC parameters control the processing of these events.

Note: For more information, see the *Administrator Guide* and the *Parameter Reference*.

)USS—Event Specifier of USS Rules

Use this format for coding the USS event definition section:

```
)USS msgidspec
```

msgidspec

Specifies the message ID specifier. Follow these guidelines when specifying the character string:

- Specify one to ten characters of the message ID.
- The string cannot contain embedded blank spaces. You can use the wildcard (*) character. For example,
  - IMW* matches IMW234, IMW56, IMW6705 and any other USS event identifier containing an IMW prefix.
  - IMW*05 matches IMWCD05, IMW205, IMW67505, and so on.
  - *05 matches any message ending with 05.
  - * alone matches all messages.

  Note: For information on using this specifier, see the chapter "Code and Debug AOF Rules (see page 59)."
- Lowercase letters are acceptable, but the AOF converts them to uppercase for event testing.

)USS USSPROCBEG—Event Specifier of USS Process Event Rules

USS process events generate the following USS messages for USS AOF rule processing:

```
USSPROCBEG DENM44SS 03FD 67108949 SPAWN INIT /bin/onetstat
USSPROCEND DENM44SS 03FD 67108949 SPAWN TERM /bin/onetstat
```

The data elements in these messages are:

- A fixed message ID for begin or end of a process
- The job name of the process.
- The ASID of the process in printable hex.
- The USS process number of the process.
- The USS service that caused the process to be created.
- The word INIT for process initialization events or the word TERM for termination events. If the process terminates abnormally the TERM word may be replaced by MEMTERM or ABTERM.
- The USS program name associated with the process. This name can be up to 128 characters long. The message text can be truncated in OPSLOG if the program name is too long. The full message text is available in the AOF rule variable USS.FULLTEXT.

Because USS process events have only two static message IDs, use the following format for coding the USS process event definition section:

```
)USS USSPROCBEG or USSPROCEND or USSPROC*
)PROC
  IF uss.type <> 'PROCESS' Then Return 0
```

You should code one USS rule for each message ID. In the USSPROCBEG rule, use the program path name and additional event data to determine if the process is managed by SSM. If so, the current state of the USS resource would be changed to the UP state and the process ID, job name, ASID, and exit timestamp would be added to the data columns of the USS resource for stopping the resource when requested. In the USSPROCEND rule, you can use the same process information to match the USS resource in SSM and set the current state to the DOWN state.

Initialization, Processing, and Termination Sections of USS Rules

USS rules follow the general guidelines for coding the initialization, processing, and termination sections and the various AOF tools that you can use.

More information:

AOF Rule Structure (see page 21)
AOF Rule Tools (see page 29)

RETURN Statements in the)PROC Section of a USS Rule

The OPS/REXX RETURN statement has no special meaning in the processing section of a USS rule. The return value has no effect on AOF processing.

Execution Considerations for USS Rules

The processing section of a rule that responds to a USS message event executes in the address space that is running CCS for z/OS. This address space varies depending on how the CA Event Manager process (CAIOPR) is started in a USS address space of z/OS. Therefore, any type of logic that could possibly suspend the processing of a USS rule should be performed by scheduling an OPS/REXX program to a CA OPS/MVS OSF TSO, TSL, or TSP server.

The AOF execution limits apply to the processing section of a rule that responds to a security event.

More information:

Code and Debug AOF Rules (see page 59)

OPS/REXX Host Environments in the)PROC Section of a USS Rule

The)PROC section of a USS rule has the following host environments with the following USS rule characteristics:

ADDRESS AOF

Sent to CA OPS/MVS. Does not wait. Output is not returned.

ADDRESS AP

Sent to MSF and then forwarded to the CA Automation Point system. Does not wait. Output is not returned.

ADDRESS EPI

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS HWS

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS ISPEXEC

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS LXCON

The VM and Linux command requests sent to USS server for execution. Does not wait. Output is not returned. The List request runs inline and returns VM and Linux system data in stem variables.

ADDRESS MESSAGE

Sent as a route code|WTO.

ADDRESS MQ

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS NETMAN

Sent to the CA OPS/MVS internal CA Netman request queue to issue MGPT commands. Does not wait. Output is not returned.

ADDRESS NETMASTR

Sent to CA NetMaster NM for SNA on the local system. Does not wait. Output is not returned.

ADDRESS OPER

Sent to target console as specified through the OCCONSOLENAME parameters when no CONNAME operands are present. Output is not returned. Schedule an OPS/REXX program in a server when a command output interrogation is needed.

ADDRESS OPSCTL

Sent to a specified facility. If the facility is ECF or OSF, does not wait. If the facility is MSF, slight wait occurs. The external data queue returns the output.

Note: If the command is MSF LIST, no wait occurs.

ADDRESS OPSDYNAM

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS OSF

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSO servers.

ADDRESS OSFTSL

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSL servers.

ADDRESS OSFTSP

Schedule TSO commands, CLISTs, or REXX EXECs to CA OPS/MVS OSF TSP servers.

ADDRESS SOF

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS SQL

Does not wait. Processed synchronously for requests that can be satisfied on the local system. Output is returned in stem variable. Error messages, if any, are returned to an external data queue.

ADDRESS SYSVIEWE

Not supported. Schedule an OPS/REXX program in a server to perform this functionality.

ADDRESS TSO

Sent to an OSF TSO server. Does not wait. Output is not returned.

ADDRESS USS

Sent to a USS server. Does not wait. Output is not returned. Schedule an OPS/REXX program in a server if a command output interrogation is needed.

ADDRESS WTO

Does not wait. Output is sent to specified (or default) destination. When attempting a WTOR, host command is sent to a TSO server for execution. The response is then returned to the server. Schedule an OPS/REXX program in a server if the WTOR response interrogation is needed.

More information:

Code and Debug AOF Rules (see page 59)

AOF Variables Available in a USS Rule

You can use all AOF variable types in USS rules. You can use the following unique AOF event variables in the)PROC section of a USS rule, and you can manually interrogate the corresponding OPSLOG display field as an aid in debugging or implementing rule logic.

USS.ATTRIBUTE

The CA Event Manager console display flag byte for the video display attributes of the message

- '00'X-Default (low intensity)
- '01'X-Make the message blink
- '02'X-Display the message in reverse video

Data Type: 1-byte binary (unprintable), read-only

Sample Value: '01'X

Note: ATTRIBUTE is a bit flag, and bits can be combined. '03'X means blinking reverse video.

USS.AUTOTOKN

The first 8 characters of the platform name on which the message originated. This value is extracted from the USS.TAG variable for inclusion in OPSLOG.

Data Type: Character, read/write

Sample Value: OS390

For USS process events, this variable contains an abbreviated name for the z/OS dynamic exit that created the event.

Possible values are:

- POSTINIT
- IMAGINIT
- PRETERM

OPSLOG Browse Column: AUTOTOKN

USS.CATEGORY

A character string assigned by the issuer of the message for message clarification

Data Type: Character, read-only

Sample Value: System error

USS.COLOR

The color that the message text will use in OPSLOG Browse. This message is initially set to the message color in the CA Event Manager console display.

Data Type: 1-byte binary (unprintable), read/write

Sample Value: '00' X

Note: Use the OPSCOLOR function of OPS/REXX to set the USS.COLOR variable.

OPSLOG Browse Column: COLOR

USS.DESC

The z/OS descriptor codes for the message. If the CA Event Manager highlights this message, descriptor code 2 is set.

Data Type: 2-byte binary (unprintable), read/write

Sample Value: '4000' X

Note: Use the OPSBITS function of OPS/REXX to set the USS.DESC variable. For a description of message descriptor codes, see the IBM documentation.

OPSLOG Browse Column: ROUTE or ROUTEX

USS.DEVICE

A character string that identifies a device associated with an SNMP trap message that was captured by the CATRAPD daemon and converted to a message in the CA Event Manager.

Data Type: Character, read-only

Sample Value: MVS25B (DASD volume)

USS.FACILITY

A character string assigned by the issuer of the message primarily for identifying the application or source of the message

Data Type: Character, read-only

Sample Value: SAPR3

For USS process events, this variable contains one of the following:

- The USS program name associated with the process. This is equivalent to the last node of the full path name of the file.
- A z/OS program name.

USS.FLAGS

The message type, taken from the CA OPS/MVS message flags described below

Data Type: 2-byte binary (unprintable), read-only

Sample Value: '8000' X

Message Flags: The CA OPS/MVS product sets the following flags with the following bits:

- '8000' X-Single line message flag
- '4040' X-WTOR message flag
- '8040' X-Immediate action message flag set (if message is highlighted in the CA Event Manager)

OPSLOG Browse Column: OPSFLAGS

USS.FULLTEXT

The complete text (up to 3000 characters) of the message taken from the CA Event Manager message block that was passed to the CA OPS/MVS USS message exit. For USS process events, the value is the complete text of the USSPROCBEG or USSPROCEND message.

Data Type: Character, read-only

Sample Value: USR98765 The names of everybody in the company: Joe, Jill, Sam, Sandra, John, Jane, and so on

USS.ID

The message identifier, usually the first token or the first blank delimited word of the message text

Data Type: Character, read-only

Sample Value: IEF125I

Note: The USS.ID variable value determines which message rules execute for the current message event. This variable never contains any special screen characters, leading, or trailing blanks.

OPSLOG Browse Column: MSGID

USS.JOBID

The identifier that JES2 or JES3 assigned to the CA Event Manager address space. If the address space was created using the MSTR subsystem, the value is the first five characters of the job name.

Data Type: Character, read-only

Sample Value: S12345

For USS process events, this variable contains the job ID of the address space running the process.

OPSLOG Browse Column: JOBID

USS.JOBNAME

The job name of the USS address space that is running the CA Event Manager. The value of this variable varies depending on how the Event Manger task is started.

Data Type: Character, read-only

Sample Value: BPXOINIT

For USS process events, this variable contains the job name of the address space running the process.

OPSLOG Browse Column: JOBNAME

USS.MSGFLAGS

The CA Event Manager message flag for special processing display options and handling:

- '01'X-Put the message in the held message display area
- '02'X-Highlight the message
- '04'X-The message was forwarded by the security access facility
- '08'X-Print the WTOR ID number to standard output (stdout)
- '10'X-Bypass CA Event Manager rules processing
- '20'X-Do not display the message on the CA Event Manager console
- '40'X-The source of event is the Windows event log

Data Type: 1-byte binary (unprintable), read-only

Sample Value: '03'X

USS.MSGUSER

The security user ID responsible for issuing the message. On the mainframe, this value is the user ID from the particular security package in use. On Windows, the current logon ID is the usual value. A \ may separate the user ID from other security-related information.

Data Type: Character, read-only

Sample Value: IBMUSER

*For USS proce*ss events, this variable contains the process user ID and alias in the format userid/alias.

USS.NODE

The TCP/IP host name that issued the message. For messages issued from the CCS for z/OS, this is the z/OS TCP/IP host name. For messages issued from a Windows platform, this value is the Domain\Node name.

Data Type: Character, read-only

Sample Value: SY23TCPN

USS.PROCESS

A character string composed of the process ID number and the program name of the message issuer. It is usually in the form *processid,program*. The UNIX command KILL requires the process ID to terminate a USS program.

Data Type: Character, read-only

Sample Value: 000356, OPSAEX

For USS process events, this variable contains the same value as the USS.PRCREATE variable (see the following section).

USS.REPLYID

The reply number associated with a CA NSM message generated by the CAWTOR command or API

Data Type: Character, read-only

Sample Value: 11

Notes:

- This variable is valid only for WTORs.
- The reply ID appears at the front of the USS.FULLTEXT variable; it is enclosed in parentheses.

OPSLOG Browse Column: First part of the text field when the message is a WTOR

USS.REPORTID

The first 8 characters of the facility or application name contained in the message. This value is extracted from the USS.FACILITY variable for inclusion in OPSLOG.

Data Type: Character, read-only

Sample Value: TAPEMGMT

For USS process events, this variable contains one of the following:

- The first eight characters of the process program name following the last '/'
- A z/OS program name

OPSLOG Browse Column: DSPNAME

USS.SEVERITY

A single character indicating the importance of the message. The issuer of the message assigns severity.

Possible Values:

- I-An informational message
- W-A warning message
- E-A serious error message
- S-Successful completion of a function
- F-Failure to complete a function

Data Type: Character, read-only

Sample Value: I

USS.SYNA

The system name of the system issuing the message

Data Type: Character, read-only

Sample Value: MVS34

Note: The system name is derived from the SYSNAME parameter specified in the appropriate IEASYS*xx* member of the Logical Parmlib Concatenation.

OPSLOG Browse Column: SYSNAME or SYNA

USS.SYSID

The system ID of the system where the message was issued (usually the SMF ID). For JES3 messages, the SYSID value derives from the MPNAME field of the Active Main Processor Control Table. For JES2 messages, the SYSID value derives from the SMF ID string.

Data Type: Character, read-only

Sample Value: S000

Note: The OPSLOG Browse column displays two characters of this variable, **not** the complete field. The CA OPS/MVS BROWSEIDFORMAT parameter determines which characters are displayed. For a description of the BROWSEIDFORMAT parameter, see the *Parameter Reference*.

OPSLOG Browse Column: SYSID

USS.TAG

A character string that identifies the type of platform that originated the message

- OS390-CCS for z/OS
- WNT-Windows workstation

Data Type: Character, read-only

Sample Value: OS390

For USS process events, this variable contains an abbreviated name for the z/OS dynamic exit that created the event.

Possible Values:

- POSTINIT
- IMAGINIT
- PRETERM

USS.TERMNAME

The first 8 characters of the network host name on which the message originated. This value is extracted from the USS.NODE variable for inclusion in OPSLOG.

Data Type: Character, read-only

Sample Value: NODESY01

For USS process events, this variable contains a system terminal name, if applicable.

OPSLOG Browse Column: TERMNAME

USS.TEXT

The first 128 characters of the USS message, taken from the CA Event Manager message block passed to the CA OPS/MVS USS message exit. For USS process events, the value is the potentially truncated text of the USSPROCBEG or USSPROCEND message.

Data Type: Character, read/write

Sample Value: WIN 12345 CA NSM now active on node IPNODE1

OPSLOG Browse Column: Text is always displayed.

USS.TOKEN

The CA Event Manager token number is a numeric value that can be the reply ID for a WTOR, an internal record number in the message database, or an offset into the message database. Except for a WTOR, this field is usually 0.

Data Type: Integer, read-only

Sample Value: 3

OPSLOG Browse Column: TOKEN

USS.TYPE

A character string indicating the type of the message issued

Possible Values:

- WTO-A standard message
- WTOR-A message expecting a reply
- COMMAND-An echo message of a command entered
- PROCESS -- For USS process events, distinguishes USS process events from other USS messages having the same message IDs
- REPLY-A reply to a WTOR message
- NKNOWN-Type cannot be classified

Data Type: Character, read-only

Sample Value: WTO

USS.USER

An 8-byte variable providing communication between rules that execute for the same USS message event. The variable can contain any installation data that these rules need, and it can store a character string displayable through OPSLOG Browse.

Data Type: User-defined, read/write

Notes:

- Before AOF processing, this variable is initialized to binary zeros. It is then passed to each rule that executes for the same USS message event; each rule can look at or change the variable contents before passing the variable to the next rule for the USS message event.
- The primary purpose of the USER variable is to provide a method to pass a small amount of data between the rules. This data may be binary or mixed case. The USER field may also be used for filtering in the OPSLOG. However, USER data used for OPSLOG filtering must be uppercase and displayable.

OPSLOG Browse Column: USER

USS.USERDATA

Defines an arbitrary character string assigned by the issuer of the message to pass data related to the message for automation or diagnostic purposes.

Data Type: Character, read-only

Sample Value: TERMERR=100

USS.USERID

The user ID of the security product on your system. This value is always the CA ACF2 user ID from the ACFASVT or the RACF user ID from the current ACEE of the USS address space that is running the CA Event Manager.

Data Type: Character, read-only

Sample Value: SYSPG01

OPSLOG Browse Column: USERID

USS.WORKLOAD

The name of a workload to which the message is related. This message field is to be used by the Workload Management application of CA NSM.

Data Type: Character, read-only

Sample Value: UNIWKLD

For USS process events, this variable contains the same value as the USS.PRTERM variable (see the following section).

USS.WORKSTATION

The name of the workstation on which a workload is performed. This could be a computer or a physical location. This variable is to be used by the CA NSM Workload Management application.

Data Type: Character, read-only

Sample Value: TAPE BACKUP

USS.WTOID

The CA Event Manager message number assigned by the issuer of the message. This is usually a number that appears somewhere in the message ID field, such as 999 in message ID CASH_999_W. This field is usually 0.

Data Type: Integer, read-only

Sample Value: 1032

OPSLOG Browse Column: WTOID

More information:

AOF Rule Tools (see page 29)

Additional AOF Variables Available in a USS Process Event Rule

In addition to the normal USS message AOF variables, the following variables are defined for USS process events only. The variables that begin with 'cr' are for the creating process information when a process creation event has occurred. These variables are null for a process termination event. The variables that begin with 'pr' are for the process that has been created or terminated.

The following USS rule process event REXX stem variables are available:

USS.CRALIAS

The alias name of the z/OS security user name associated with the creating process. This value is usually the same as USS.CRUSERID.

Data Type: Character, read-only

Sample Value: USSWIZ

USS.CRASID

The z/OS address space number in which the creating process is running.

Data Type: 2-byte binary (unprintable), read-only

Sample Value: '00AB'X

USS.CRJOBNAME

The z/OS job name of the address space in which the creating process is running.

Data Type: Character, read-only

Sample Value: TESTJOB1

USS.CRPGMNAME

The name of the z/OS program or HFS or zFS in compatibility mode path file name associated with the creating process.

Data Type: Character, read-only

Sample Value: /bin/sh

USS.CRPID

The USS process id number of the creating process.

Data Type: Integer, read-only

Sample Value: 123789

USS.CRUSERID

The z/OS security user name associated with the creating process

Data Type: Character, read-only

Sample Value: USSGURU

USS.PRALIAS

The alias name of the z/OS security user name associated with the process. This value is usually the same as USS.PRUSERID

Data Type: Character, read-only

Sample Value: USSWIZ

USS.PRASID

The z/OS address space number in which the process is running.

Data Type: 2-byte binary (unprintable), read-only

Sample Value: '00BA'X

OPSLOG Browse Column: ASID

USS.PRCREATE

The type of USS service that was used to create the process.

Data Type: Character, read-only

Possible Values:

- FORK-the USS fork service
- SPAWN-the USS spawn service
- ATEXEC-the USS attach_exec service
- ATEXMVS-the USS attach_execmvs service
- 1STCALL-the process was created when a program called any USS service function. When the task that caused the process to be created ends, the process ends.

Sample Value: FORK

USS.PRJOBNAME

The z/OS job name of the address space the process in which the process is running.

Data Type: Character, read-only

Sample Value: TESTJOB2

OPSLOG Browse Column: JOBNAME

USS.PRPGMNAME

The name of the z/OS program or HFS or zFS in compatibility mode path file name associated with the process.

Data Type: Character, read-only

Sample Value: /u/bin/myprog

USS.PRPID

The USS process id number of the created or terminated USS process.

Data Type: Integer, read-only

Sample Value: 147249

USS.PRTERM

The type of termination that ended the process.

Data Type: Character, read-only

Possible Values:

- TERM-Normal termination
- MEMTERM-the process is being terminated by address space termination.
- ABTERM-the process terminated abnormally

For process initialization events the value is always INIT.

Sample Value: TERM

USS.PRTIMESTMP

The date and time the process was created. Because process ID numbers can be reused by USS, the combination of process number and create time serve as a unique value to identify the process.

Data Type: 8-byte binary value in z/OS time-of-day clock format

Sample Value: 'B256FC206E6980AB'X

USS.PRUSERID

The z/OS security user name associated with the process

Data Type: Character, read-only

Sample Value: USSGURU

OPSLOG Browse Column: USERID

Debug a USS Rule

To debug a TLM rule

1. Set the CA OPS/MVS BROWSEUSS parameter to YES.
2. Set the USS event profile of your OPSLOG display to Y.

   This lets you view all USS events.
3. With these parameters set, display the OPSLOG EVENT column.

   This lets you see recorded USS events. This record contains details of each USS event for which you can use additional OPSLOG display columns to further interrogate each event.

More information:

Code and Debug AOF Rules (see page 59)

Example of a USS Rule

The following is an example of a USS message rule that informs System State Manager that the new IBM Web Server product is running in OMVS:

```
)USS IMW35361
)PROC
/* Inform SSM component that IBM web server running in OMVS has */
/* initialized (CURRENT_STATE = 'UP'). we'll also extract the   */
/*  USS process id and place it in the STCTBL table. This    */
/* ID will be used in the shutdown procedures of the server.    */
if USS.NODE = USS.SYNA then   /* Be sure msg from our system */
  do
    parse upper var USS.PROCESS PROCID '/' .  /* GET USS PROCESS ID */
                                 /* NEEDED TO STOP THIS PROCESS */
    /* Use OPS/REXX SQL host environment to update SSM table    */

    address SQL "UPDATE STCTBL SET CURRENT_STATE = 'UP',",
    "PROCESSID = "PROCID" WHERE NAME = 'WEBSERV'"
/* send a message to a remote UNICENTER server  */
    MSGTXT = 'OPSAUTO1 IBM WEB server initialized at 'TIME()
    address USS
    "WTO TEXT('"MSGTXT"') NODE(MAINUNI) "
  End
return
```

Appendix A: Summary of AOF Coding Guidelines

This section contains the following topics:

OPS/REXX Host Environment Rule Characteristics

This section discusses the rule characteristics of the various OPS/REXX host environments. The information is presented alphabetically by host environment.

More information:

Coding Each AOF Rule Type (see page 69)

ADDRESS AOF

ADDRESS AOF uses the following rule types:

API, ARM, CMD, DOM, EOJ, EOM, EOS, GLV, MSG, OMG, SCR, SEC, TLM, TOD, USS

Rule Characteristics:

- Sent to CA OPS/MVS.
- Does not wait.
- No output returned.

REQ

Rule Characteristics:

- Runs inline.
- Waits for output in external data queue.

ADDRESS AP

ADDRESS AP uses the following rule types:

API, ARM, CMD, DOM, EOJ, EOM, EOS, GLV, MSG, OMG, REQ, SCR, SEC, TLM, TOD, USS

Rule Characteristics:

- Sent to CMSF to be forwarded to the CA Automation Point system.
- Does not wait.
- No output returned.

ADDRESS EPI

ADDRESS EPI uses the following rule types:

API, ARM, CMD, DOM, EOJ, EOM, EOS, GLV, MSG, OMG, SEC, TLM, USS

Rule Characteristics:

- Not supported.
- Schedule an OPS/REXX program in a server to perform this functionality. For details, see the chapter “Code and Debug AOF Rules.”

REQ, SCR, TOD

Rule Characteristics:

- Runs inline.
- Waits for output in external data queue.

ADDRESS HWS

ADDRESS HWS uses the following rule types:

API, ARM, CMD, DOM, EOJ, EOM, EOS, GLV, MSG, OMG, REQ, SCR, SEC, TLM, TOD, USS

Rule Characteristics:

- Not supported.
- Schedule an OPS/REXX program in a server to perform this functionality. For more information, see Code and Debug AOF Rules (see page 59).

REQ

Rule Characteristics:

- Runs inline.
- Waits for output in an external data queue.

ADDRESS MIM

ADDRESS MIM uses the following rule types:

API, ARM, CMD, DOM, EOJ, EOM, EOS, GLV, MSG, OMG, REQ, SCR, SEC, TLM, TOD, USS

Characteristics:

- Request sent to CA MIM
- Output returned in stem variables
- Error messages, if any, returned to external data queue

ADDRESS ISPEXEC

ADDRESS ISPEXEC uses the following rule types:

API, ARM, CMD, DOM, EOJ, EOM, EOS, GLV, MSG, OMG, SCR, SEC, TLM, TOD, USS

Rule Characteristics:

- Not supported.
- Schedule an OPS/REXX program in a server to perform this functionality. For details, see the chapter “Code and Debug AOF Rules.”

REQ

Rule Characteristics:

- Runs inline.
- Waits for output in external data queue.

ADDRESS LXCON

ADDRESS LXCON uses the following rule types:

API, ARM, CMD, DOM, EOJ, EOM, EOS, GLV, MSG, OMG, SEC, TLM, USS

Rule Characteristics:

- Sent to USS a server.
- Does not wait.
- No output is returned.
- Schedule an OPS/REXX program in a server if command output interrogation is needed.
- LIST function of LXCON always runs inline and returns stem variable output.

REQ, SCR, TOD

Rule Characteristics:

- Runs inline.
- Waits for output in stem variables.

ADDRESS MESSAGE

ADDRESS MESSAGE uses the following rule types:

API, ARM, DOM, EOJ, EOM, EOS, GLV, MSG, OMG, SCR, TLM, TOD, USS

Rule Characteristics:

Sent as a route code|WTO.

CMD

Rule Characteristics:

Sent to issuing console.

REQ, SEC

Rule Characteristics:

Sent to TSO user if invoked from a foreground TSO session. Otherwise, it is sent to OPSLOG.

ADDRESS NETMAN

ADDRESS NETMAN uses the following rule types:

API, ARM, CMD, DOM, EOJ, EOM, EOS, GLV, MSG, OMG, SCR, SEC, TOD, TLM, USS

Rule Characteristics:

- Sent to CA OPS/MVS internal CA Netman request queue to issue MGPT commands. Does not wait.
- No output returned.

REQ

Rule Characteristics:

- Sent to CA OPS/MVS internal CA Netman request queue to issue MGPT commands. Waits for command completion.
- Output returned to external data queue.

ADDRESS NETMASTR

ADDRESS NETMASTR uses the following rule types:

API, ARM, CMD, DOM, EOJ, EOM, EOS, GLV, MSG, OMG, REQ, SCR, SEC, TLM, TOD, USS

Rule Characteristics:

- Does not wait.
- Command sent to CA NSM NetMaster Option and possibly forwarded to another CA NSM NetMaster Option system.
- Return code indicates whether the command was successfully passed to CA NSM NetMaster Option on the local system.
- No output returned.

ADDRESS OPER

ADDRESS OPER uses the following rule types:

API, ARM, CMD, DOM, EOJ, EOM, EOS, GLV, MSG, OMG, SCR, SEC, TLM, USS

Rule Characteristics:

- Sent to target console as specified through the OCCONSOLENAME parameters if no CONNAME operands are present.
- No output returned.
- Schedule an OPS/REXX program in a server if command output interrogation is needed.

REQ, TOD

Rule Characteristics:

- If NOOUTPUT keyword is omitted, valid command responses are returned to the external data queue. If NOOUTPUT keyword is present (it should be if you want no output), command is sent to the console as specified through the OCCONSOLENAME parameters.
- Waiting for output is highly discouraged except in cases where the responses are relatively short and timely.

More information:

Code and Debug AOF Rules (see page 59)

ADDRESS OPSCTL

ADDRESS OPSCTL uses the following rule types:

API, ARM, CMD, DOM, EOJ, EOM, EOS, GLV, MSG, OMG, REQ, SCR, SEC, TLM, TOD, USS

Rule Characteristics:

- Sent to specified facility. If facility is ECF or OSF, does not wait. If facility is MSF, slight wait occurs.
- Output returned to external data queue.

Note: If the command is MSF LIST, no wait occurs.

ADDRESS OPSDYNAM

ADDRESS OPSDYNAM uses the following rule types:

API, ARM, CMD, DOM, EOJ, EOM, EOS, GLV, MSG, OMG, SEC, TLM, USS

Rule Characteristics:

- Not supported.
- Schedule an OPS/REXX program in a server to perform this functionality.

REQ, SCR, TOD

Rule Characteristics:

- Runs inline.
- Returns output in variables.

More information:

Code and Debug AOF Rules (see page 59)

ADDRESS OSF, ADDRESS OSFTSL, and ADDRESS OSFTSP

ADDRESS OSF, ADDRESS OSFTSL, and ADDRESS OSFTSP use the following rule types:

API, ARM, CMD, DOM, EOJ, EOM, EOS, GLV, MSG, OMG, REQ, SCR, SEC, TLM, TOD, USS

Rule Characteristics:

- Sent to the relevant OSF TSO server class.
- Does not wait.
- No output returned.

ADDRESS SOF

ADDRESS SOF uses the following rule types:

API, ARM, CMD, DOM, EOJ, EOM, EOS, GLV, MSG, OMG, REQ, SCR, SEC, TLM, TOD, USS

Rule Characteristics:

- Not supported.
- Schedule an OPS/REXX program in a server to perform this functionality.

More information:

Code and Debug AOF Rules (see page 59)

ADDRESS SQL

ADDRESS SQL uses the following rule types:

API, ARM, CMD, DOM, EOJ, EOM, EOS, GLV, MSG, OMG, REQ, SCR, SEC, TLM, TOD, USS

Rule Characteristics:

- Does not wait. Processed synchronously for requests that can be satisfied on the local system.
- Output returned in stem variable.
- Error messages, if any, returned to external data queue.

ADDRESS SYSVIEWE

ADDRESS SYSVIEWE uses the following rule types:

API, ARM, CMD, DOM, EOJ, EOM, EOS, GLV, MSG, OMG, REQ, SCR, SEC, TLM, TOD, USS

Rule Characteristics:

- Not supported.
- Schedule an OPS/REXX program in a server to perform this functionality.

More information:

Code and Debug AOF Rules (see page 59)

ADDRESS TSO

ADDRESS TSO uses the following rule types:

API, ARM, CMD, DOM, EOJ, EOM, EOS, GLV, MSG, OMG, SCR, SEC, TOD, TLM, USS

Rule Characteristics:

- Sent to OSF TSO server.
- Does not wait.
- No output returned.

REQ

Rule Characteristics:

- Runs inline.
- Waits for output in external data queue.

ADDRESS USS

ADDRESS USS uses the following rule types:

API, ARM, CMD, DOM, EOJ, EOM, EOS, GLV, MSG, OMG, SEC, TLM, USS

Rule Characteristics:

- Sent to USS server.
- Does not wait.
- No output returned.
- Schedule an OPS/REXX program in a server if command output interrogation is needed.

REQ, SCR, TOD

Rule Characteristics:

- Runs inline.
- Waits for output in stem variables.

ADDRESS WTO

ADDRESS WTO uses the following rule types:

API, ARM, CMD, DOM, EOJ, EOM, EOS, GLV, MSG, OMG, SEC, TLM, USS

Rule Characteristics:

- Does not wait.
- Sends output to specified (or default) destination.
- If you attempt a WTOR, host command is sent to TSO server for execution and response is returned to server. Schedule an OPS/REXX program in a server if WTOR response interrogation is needed.

REQ, SCR, TOD

Rule Characteristics:

- Does not wait.
- Output sent to specified console.
- If you attempt a WTOR, runs inline and waits for response in external data queue.

SAY Output

The following describes the SAY output rule types and where the SAY output is sent for each rule:

API, ARM, DOM, EOJ, EOM, EOS, MSG, OMG, SCR, TLM, TOD, USS

Sent as a WTO with OPS1000I as a default message ID, which may result in it being broadcasted to physical consoles.

CMD

Sent as a WTO with OPS1000I as a default message ID and is directed to the console that triggered the event.

GLV

Destination is that of the SAY destination for the event that initiated the GLV rule.

REQ

Sent as a PUTLINE with OPS1000I as a default message ID to a TSO user. It is sent as a WTO with OPS3092H as a default message ID (prefixed in front of the OPS1000I) to the hardcopy log if the request originated in any class of OSF TSO server.

SEC

Sent as a PUTLINE with OPS1000I as a default message ID to a TSO user. Otherwise, it is sent as a WTO with OPS1000I as a default message ID, which may result in it being broadcasted to physical consoles.

Note: With the exception of REQ and SEC rules, the SAY instruction should only be used for debugging purposes. Use the ADDRESS WTO OPS/REXX host environment when WTO generation is needed in your automation.

TRACE Output

The following describes the TRACE output rule types and where the TRACE output is sent for each rule:

API, ARM, CMD, DOM, EOJ, EOM, EOS, GLV, MSG, OMG, REQ, SCR, SEC, TLM, TOD, USS

With a default message severity of T for OPS0997, all TRACE activity is sent to the OPSLOG.

Execution Considerations for Each Rule Type

The following describes each rule type and where the processing section of rules execute:

API

In the address space of the application that generated the event

ARM

In the XCFAS address space

CMD

In the address space from which the command was issued (usually CONSOLE)

DOM

In the address space from which the delete operator message was issued

EOJ

In the address space where the job is ending

EOM

In the master address space

EOS

In the address space where the step is ending

GLV

In the same address space in which the REXX program or AOF rule that changed the global variable ran

MSG

In the address space from which the message was issued

OMG

In the OMEGAMON address space

REQ

In the address space from which the OPSREQ TSO command was issued

SCR, TOD

In the CA OPS/MVS address space

SEC

In the address space where the request that caused the security event was issued

TLM

In the address space that has reached its time or wait limit

USS

In the CA NSM address space

Index