# Contents

**List of Tables**  
- Contents 3
  
**List of Figures**  
- Contents 5
  
**Preface**  
- Contents 7
  
**Syntax Diagrams**  
- Contents 9
  
**Adobe Portable Document Format**  
- Contents 11
  
**What’s New in AF/OPERATOR Version 340**  
- Contents 13
  
**Chapter 1. Understanding the Programming Environment**  
- Contents 15
  
**Language Elements**  
- Contents 17
  
**Elements of AF/OPERATOR**  
- Contents 19
  
**Special Considerations for Using the REXX Language**  
- Contents 21
  
**Differences in Passing Parameters**  
- Contents 23
  
**Where to Look for More Information**  
- Contents 25
  
**Chapter 2. Variables in the AF/OPERATOR Environment**  
- Contents 27
  
**About Variables**  
- Contents 29
  
**Characteristics of Variables**  
- Contents 31
  
**Rules for Naming Variables**  
- Contents 33
  
**Defining Variables**  
- Contents 35
  
**How AF/OPERATOR Interprets Variables**  
- Contents 37
  
**AF/OPERATOR-Supplied Variables**  
- Contents 39
  
**Chapter 3. REXX Shared Variables**  
- Contents 41
  
**Creating Rexx Shared Variables**  
- Contents 43
  
**Additional Information about the SYSPLEX RSV Pool**  
- Contents 45
  
**Functions for Shared Variables**  
- Contents 47
  
**SHARVGET**  
- Contents 49
  
**SHARVUPD**  
- Contents 51
  
**SHARVDEL**  
- Contents 53
  
**Stem Variables and Compound Variable Groups**  
- Contents 55
  
**IRXEXCOM Return Codes**  
- Contents 57
  
**Chapter 4. Pattern Matching**  
- Contents 59
  
**Overview of Pattern Matching**  
- Contents 61
  
**The Two Types of Patterns**  
- Contents 63
  
**Pattern Characters**  
- Contents 65
  
**Pattern Variables**  
- Contents 67

---

*Contents* 3
<table>
<thead>
<tr>
<th>Chapter 5.</th>
<th>Candle-Supplied REXX Functions</th>
<th>93</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date/Time Functions</td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>IMSCMD Function</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>NetView Interface Functions</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>OMEGAVIEW Interface Functions</td>
<td>109</td>
<td></td>
</tr>
<tr>
<td>POVI Functions</td>
<td>121</td>
<td></td>
</tr>
<tr>
<td>REDIRECT Function</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>System Functions</td>
<td>141</td>
<td></td>
</tr>
<tr>
<td>Variable Functions</td>
<td>165</td>
<td></td>
</tr>
<tr>
<td>Communication Functions</td>
<td>170</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 6.</th>
<th>AF/OPERATOR Commands</th>
<th>183</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issuing Commands</td>
<td>187</td>
<td></td>
</tr>
<tr>
<td>AF</td>
<td>189</td>
<td></td>
</tr>
<tr>
<td>AF (no operand)</td>
<td>192</td>
<td></td>
</tr>
<tr>
<td>AF AUTOREF and AF NOAUTOREF</td>
<td>193</td>
<td></td>
</tr>
<tr>
<td>AF CANCEL</td>
<td>195</td>
<td></td>
</tr>
<tr>
<td>AF CMD and AF NOCMD</td>
<td>197</td>
<td></td>
</tr>
<tr>
<td>AF COMTRACE</td>
<td>199</td>
<td></td>
</tr>
<tr>
<td>AF DOM and AF NODOM</td>
<td>204</td>
<td></td>
</tr>
<tr>
<td>AF DUB AS_PROCESS</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>AF FREE</td>
<td>206</td>
<td></td>
</tr>
<tr>
<td>AF LOG and AF NOLOG</td>
<td>207</td>
<td></td>
</tr>
<tr>
<td>AF MATINFO</td>
<td>208</td>
<td></td>
</tr>
<tr>
<td>AF QUIET and AF NOQUIET</td>
<td>209</td>
<td></td>
</tr>
<tr>
<td>AF RECOVER and AF NORECOVER</td>
<td>211</td>
<td></td>
</tr>
<tr>
<td>AF RELOAD(HOSTNAME)</td>
<td>212</td>
<td></td>
</tr>
<tr>
<td>AF RXSEARCH</td>
<td>213</td>
<td></td>
</tr>
<tr>
<td>AF SSRESET and AF SSIDRAIN</td>
<td>214</td>
<td></td>
</tr>
<tr>
<td>AF STATS</td>
<td>215</td>
<td></td>
</tr>
<tr>
<td>AF STOP and AF KILL</td>
<td>217</td>
<td></td>
</tr>
<tr>
<td>AF TODSTART</td>
<td>219</td>
<td></td>
</tr>
<tr>
<td>AF WARN and AF PROCESS</td>
<td>220</td>
<td></td>
</tr>
<tr>
<td>AF WTO and AF NOWTO</td>
<td>222</td>
<td></td>
</tr>
<tr>
<td>AFADDR</td>
<td>223</td>
<td></td>
</tr>
<tr>
<td>AFCALL</td>
<td>226</td>
<td></td>
</tr>
<tr>
<td>ALLOC (AL)</td>
<td>228</td>
<td></td>
</tr>
<tr>
<td>BEEP</td>
<td>231</td>
<td></td>
</tr>
<tr>
<td>CLOSE (CL)</td>
<td>234</td>
<td></td>
</tr>
<tr>
<td>COMM</td>
<td>236</td>
<td></td>
</tr>
<tr>
<td>COMM DISPLAY</td>
<td>237</td>
<td></td>
</tr>
<tr>
<td>COMM RELEASE</td>
<td>245</td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>COMM START</td>
<td>248</td>
<td></td>
</tr>
<tr>
<td>COMM STOP</td>
<td>252</td>
<td></td>
</tr>
<tr>
<td>COMM SUSPEND</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>CP</td>
<td>258</td>
<td></td>
</tr>
<tr>
<td>DELETE (DEL)</td>
<td>259</td>
<td></td>
</tr>
<tr>
<td>DOM</td>
<td>260</td>
<td></td>
</tr>
<tr>
<td>EXEC (EX)</td>
<td>262</td>
<td></td>
</tr>
<tr>
<td>EXECDROP</td>
<td>264</td>
<td></td>
</tr>
<tr>
<td>EXECLOAD</td>
<td>266</td>
<td></td>
</tr>
<tr>
<td>EXECMAP</td>
<td>267</td>
<td></td>
</tr>
<tr>
<td>FREE (FR)</td>
<td>268</td>
<td></td>
</tr>
<tr>
<td>HELP (HE)</td>
<td>270</td>
<td></td>
</tr>
<tr>
<td>IMSOPER</td>
<td>272</td>
<td></td>
</tr>
<tr>
<td>JESSOPER (J3OPER)</td>
<td>273</td>
<td></td>
</tr>
<tr>
<td>LINK</td>
<td>274</td>
<td></td>
</tr>
<tr>
<td>LINK DEFINE</td>
<td>275</td>
<td></td>
</tr>
<tr>
<td>LINK DELETE</td>
<td>280</td>
<td></td>
</tr>
<tr>
<td>LINK DISPLAY</td>
<td>283</td>
<td></td>
</tr>
<tr>
<td>LINK START</td>
<td>286</td>
<td></td>
</tr>
<tr>
<td>LINK STOP</td>
<td>289</td>
<td></td>
</tr>
<tr>
<td>LOGOFF</td>
<td>292</td>
<td></td>
</tr>
<tr>
<td>LOGON</td>
<td>293</td>
<td></td>
</tr>
<tr>
<td>MLWTO</td>
<td>298</td>
<td></td>
</tr>
<tr>
<td>NVALERT</td>
<td>301</td>
<td></td>
</tr>
<tr>
<td>NVISHOW</td>
<td>304</td>
<td></td>
</tr>
<tr>
<td>NVISTART</td>
<td>306</td>
<td></td>
</tr>
<tr>
<td>NVISTOP</td>
<td>307</td>
<td></td>
</tr>
<tr>
<td>OPEN (OP)</td>
<td>308</td>
<td></td>
</tr>
<tr>
<td>OPER</td>
<td>310</td>
<td></td>
</tr>
<tr>
<td>PEER</td>
<td>316</td>
<td></td>
</tr>
<tr>
<td>PEER START</td>
<td>317</td>
<td></td>
</tr>
<tr>
<td>PEER STOP</td>
<td>318</td>
<td></td>
</tr>
<tr>
<td>PEERLINK</td>
<td>320</td>
<td></td>
</tr>
<tr>
<td>PEERLINK DEFINE</td>
<td>322</td>
<td></td>
</tr>
<tr>
<td>PEERLINK DELETE</td>
<td>323</td>
<td></td>
</tr>
<tr>
<td>PEERLINK DISPLAY</td>
<td>324</td>
<td></td>
</tr>
<tr>
<td>PEERLINK START</td>
<td>325</td>
<td></td>
</tr>
<tr>
<td>PEERLINK STOP</td>
<td>326</td>
<td></td>
</tr>
<tr>
<td>Programmerless Open VTAM Interface</td>
<td>327</td>
<td></td>
</tr>
<tr>
<td>READ</td>
<td>328</td>
<td></td>
</tr>
<tr>
<td>SHOW (SH)</td>
<td>330</td>
<td></td>
</tr>
<tr>
<td>SHOW ENQUEUE</td>
<td>332</td>
<td></td>
</tr>
<tr>
<td>SHOW FILES</td>
<td>334</td>
<td></td>
</tr>
<tr>
<td>SHOW HOSTNAME</td>
<td>335</td>
<td></td>
</tr>
<tr>
<td>SHOW LOGONS</td>
<td>336</td>
<td></td>
</tr>
</tbody>
</table>

Contents
List of Tables

Table 1. TSO/E REXX Commands Supported/Not Supported .......................... .31
Table 2. AF/OPERATOR-Supplied System Variables ................................. .50
Table 3. AF/OPERATOR-Supplied Match Variables ................................. .52
Table 4. Wait Command Match Variables ............................................. .56
Table 5. JES2 Interface Match Variables ............................................. .57
Table 6. JES3 Interface Match Variables ............................................. .59
Table 7. AF/OPERATOR Line Variables ................................................. .61
List of Figures

FIGURE 1. Services and Components of AF/OPERATOR ........................................... 24
FIGURE 2. REXX Language Environment .............................................................. 29
FIGURE 3. Variables in the REXX and AF/OPERATOR Environments ...................... 33
FIGURE 4. Language Reference Manuals ............................................................... 40
FIGURE 5. SHARVPUT Function ....................................................................... 69
FIGURE 6. SHARVGET Function ....................................................................... 72
FIGURE 7. SHARVUPD Function ....................................................................... 74
FIGURE 8. SHARVDEL Function ....................................................................... 77
FIGURE 9. OMEGAMON Interface Exec. ............................................................... 443
Preface

About this document
The AF/OPERATOR® Command Reference Manual is designed primarily for those who are already familiar with the concepts and features of AF/OPERATOR. It contains the syntax and usage of the product control language commands for use in REXX execs. See the AF/OPERATOR User’s Guide for an introduction to AF/OPERATOR concepts and features.

Documentation set information
The following manuals comprise the AF/OPERATOR documentation set:
- AF/OPERATOR Configuration and Customization Guide
- AF/OPERATOR User’s Guide
- AF/OPERATOR Command Reference Manual
- AF/OPERATOR User Interface Guide
- AF/OPERATOR OMEGAMON II for SMS Data Interface
- AF/OPERATOR Message Manager
- AF/OPERATOR Started Task Manager
- AF/OPERATOR: Using the Subsystem Logging Facility
- Connecting AF/OPERATOR and AF/REMOTE Using TCP/IP
- Candle Product Messages Manual
Where to look for more information

For more information related to this product, please see the
- technical documentation CD-ROM that came with your product
- technical documentation information available on the Candle Web site at www.candle.com
- online help provided with this product

Ordering additional documentation

To order additional product manuals, contact your Candle Customer Support representative.

We would like to hear from you

Candle welcomes your comments and suggestions for changes or additions to the documentation set. A user comment form, located at the back of each manual, provides simple instructions for communicating with the Candle Information Development department.

You can also send email to UserDoc@candle.com. Please include "AF/OPERATOR Command Reference Manual Version 340" in the subject line.
Syntax Diagrams

This manual uses the format below to describe the syntax of the AF/OPERATOR commands.

- Some commands can be abbreviated. If an abbreviation is applicable, it is shown in parentheses next to the command name.
- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.
- The symbol below indicates the statement is continued:

>______________>

- The symbol below indicates that a line is continued from the previous line:

>______________

- The symbol below indicates the end of a statement:

>______________<

- Required items appear on the horizontal line (the main path).

```
STATEMENT __________ Required Item __><
```

- Optional items appear below the main path.

```
STATEMENT _________________________><
    OPTIONAL ITEM
```

- When you can choose from two or more items, they are stacked vertically.
  If you must choose one of the items, the item at the top of the stack appears on the main path.

```
STATEMENT __________ Required Choice1 __><
    Required Choice2
```

If choosing one of the items is optional, the entire stack appears below the main path.

```
STATEMENT __________><
    Optional Choice1
    Optional Choice2
```
Syntax Diagrams

- An arrow returning to the left above the main line indicates an item that can be repeated.

```
STATEMENT  Repeatable item
```

A repeat arrow above a stack indicates that you can make more than one choice from the stacked items, or repeat a single choice.

- If punctuation marks, parentheses, arithmetic operators, or other symbols are shown, they must be entered as part of the syntax.

- An operand name ending in /s specifies a choice of one or more of the operands, the various choices separated by space(s). For example:

  `option/s`

  specifies

  `option1 option2 ...

- An operand name ending in ,s specifies a choice of one or more of the operands, the various choices separated by commas. For example:

  `pattern,s`

  specifies

  `pattern1,pattern2,...`

Default operands are underlined.
Adobe Portable Document Format

Printing this book

Candle supplies documentation in the Adobe Portable Document Format (PDF). The Adobe Acrobat Reader will print PDF documents with the fonts, formatting, and graphics in the original document. To print a Candle document, do the following:

1. Specify the print options for your system. From the Acrobat Reader Menu bar, select File > Page Setup… and make your selections. A setting of 300 dpi is highly recommended as is duplex printing if your printer supports this option.

2. To start printing, select File > Print… on the Acrobat Reader Menu bar.

3. On the Print pop-up, select one of the Print Range options for
   - All
   - Current page
   - Pages from: [ ] to: [ ]

4. (Optional). Select the Shrink to Fit option if you need to fit oversize pages to the paper size currently loaded on your printer.

Printing problems?

The print quality of your output is ultimately determined by your printer. Sometimes printing problems can occur. If you experience printing problems, potential areas to check are:

- settings for your printer and printer driver. (The dpi settings for both your driver and printer should be the same. A setting of 300 dpi is recommended.)
- the printer driver you are using. (You may need a different printer driver or the Universal Printer driver from Adobe. This free printer driver is available at www.adobe.com.)
- the halftone/graphics color adjustment for printing color on black and white printers (check the printer properties under Start > Settings > Printer). For more information, see the online help for the Acrobat Reader.
- the amount of available memory in your printer. (Insufficient memory can cause a document or graphics to fail to print.)

For additional information on printing problems, refer to the documentation for your printer or contact your printer manufacturer.

Contacting Adobe

If additional information is needed about Adobe Acrobat Reader or printing problems, see the Readme.pdf file that ships with Adobe Acrobat Reader or contact Adobe at www.adobe.com.
This section provides a description of the new features that have been incorporated into AF/OPERATOR Version 340.

**TCP/IP connectivity enhancements**

TCP/IP communications have been simplified by permitting you to optionally remove the AF packet header when communicating between AF/OPERATOR and systems or applications that are unable to comply with the Candle AF packet header protocol. You accomplish this by means of a new option on the LINK DEFINE and COMM START commands. In addition, new SEND and RECEIVE datatypes have been added to the COMSDRCV REXX function. You must use the SEND and RECEIVE datatypes when you want to transfer data over a link having the AF packet header turned off.

New keywords have been added to COMM START and COMM STOP commands. The CONNECT_EXEC keyword on the COMM START command identifies a named exec that runs whenever a connection to the server is established and is mandatory when you have specified AFPACKET(OFF). The CID keyword on the COMM STOP command identifies a particular connection to a server to be stopped.

A new COMADMIN REXX function provides GIVE and TAKE options that permit passing ownership of a connection from one match to another.

**Passing larger amounts of data on a TCP/IP match**

You can now pass more data to an individual match. AF/OPERATOR will permit passing a larger amount of parameter data to a REXX procedure. In addition, it will provide the capability to create a conversation between matches so that multiple 32K packets can be transferred. This is accomplished by permitting the DATARPLY datatype to code the replylength keyword on the COMSDRCV REXX function.

**Automated peer-to-peer link management**

By predefining link attributes, it is now possible to automatically establish connections at AF/OPERATOR startup. The following product changes support this function:

- The optional RECOVERY keyword on the LINK DEFINE command indicates that an INACTIVE link having a desired state of ACTIVE is to be automatically started when the specified server becomes available.
The new ACTIVATE keyword on the LINK DEFINE command indicates that a LINK START is to be automatically attempted for a particular linkid, assuming a successful link definition.

The new SCOPE keyword on the LINK START command specifies those links that are to be selected for LINK START processing according to their activation state.

Two new startup parameters, HOSTNAME and LINKDEFS are added. HOSTNAME(xxxxxxxx) overrides the default name of the trusted hostnames member in RKANPAR. The LINKDEFS(xxxxxxxx) parameter overrides the default name of the link definitions member in RKANPAR. During product startup, when TIMEOUT(nnn) is specified on the LINKDEFS keyword, it specifies the amount of time to wait for the LINK START command issued immediately after processing the LINKDEFS member to complete before startup is allowed to continue.

Changes to the OPER command when RESP is specified

The OPER command issues an MVS or subsystem command. On this command, the RESP parameter specifies that a set of line variables will receive the response text resulting from the command. Additional parameters have been added to the OPER command when specified with the RESP parameter:

- **TIMEOUT(pp,ss):** The new ss value specifies the number of seconds (from 1 through 3600) that AF/OPERATOR is to wait for each response message line from a multi-line write to operator before assuming the response is complete.

- **MAXLINES(nnnn):** This new parameter specifies that a response is to be deemed complete when the specified number of lines is received.

- **ENDMSG(prefix):** This new parameter specifies that the response is to be deemed complete when the specified prefix is encountered in one of the response lines.

The ss value is also added to the OPERRESP AF/OPERATOR startup parameter.

In addition, the CMDSDRCV function has been modified such that when it causes an OPER command with RESP specified to be executed on a remote system, it also causes an AOCASE variable to be built when the reply from the remote system is received.

Additional modifications

- New global variables have been added for COM matches.

- A DUB_AS_PROCESS command, having the same function as the AF/OPERATOR startup parameter of the same name, has been provided.

Storage Constraint Relief

The amount of storage constraint relief realized will vary depending on AF/OPERATOR startup parameters and will be equal to (MAXMAT - MAXRUN) * 480 bytes.
**Online documentation**

With Version 340, Candle Corporation has moved AF/OPERATOR manuals from IBM BookMaster to Adobe FrameMaker. This move was made to better enable us to address our customers’ needs by providing tools that enhance productivity.

One of the results of the move is that it is no longer possible to create BookManager versions of the AF/OPERATOR manuals. However, the manuals remain available online in the Adobe PDF version on CD-ROM and are also available on the Candle Corporation website at [www.Candle.com](http://www.Candle.com).

The documentation CD being provided with this release has robust and easy-to-use search capabilities. You can search for information in multiple volumes, multiple versions, and across products. The CD also provides easy setup of search indexes with a single click of the mouse.

If you want to order printed copies of the documentation, please contact your Candle Support Services representative.

**Enhancements to product documentation**

- Additional documentation about obtaining SMF data has been added to the *AF/OPERATOR User’s Guide*.

- Additional documentation about using the Probe Directive, Probe Input, and Misc.Parms fields has been added to the *OMEGACENTER Status Manager User’s Guide*.
Introduction

AF/OPERATOR lets you write REXX execs to perform complex action sequences in support of your automation needs. A REXX exec is a collection of commands which, taken together and appropriately sequenced, can be thought of as a small application program. This chapter explains the language elements that are part of the AF/OPERATOR programming environment and gives an overview of the services provided by each element.

“Where to Look for More Information” on page 39 explains where to find detailed information for writing REXX execs.

This manual is a guide for the AF/OPERATOR language commands and functions. This manual contains only overview information and reference information on the syntax of programming functions and commands. For a description of how AF/OPERATOR works, see the AF/OPERATOR User’s Guide.

Chapter Contents

Language Elements ................................................................. 22
Elements of AF/OPERATOR .................................................. 23
Special Considerations for Using the REXX Language ................. 29
Differences in Passing Parameters ........................................... 38
Where to Look for More Information ...................................... 40
Language Elements

You can create REXX execs using the IBM TSO/E REXX language instructions to control the program flow (the sequential execution of a REXX procedure such as DO WHILE).

These REXX execs can be used to perform tasks like suppressing system messages, issuing commands, and initiating traps.

There are two types of commands in the AF/OPERATOR environment:

- product control commands
- automation commands

Product control commands allow you to define and execute traps, activate and control the Multi-System Management Facility (MSMF), and perform other AF/OPERATOR tasks. See “AF/OPERATOR Commands” on page 183 for more information.

Automation commands allow you to perform automation tasks like suppressing system messages. Automation commands are comprised of host commands and Candle-supplied REXX Functions.

The next section gives pertinent information on each command type illustrated above. First, to give you an understanding about how automation is achieved, the components and services of AF/OPERATOR are detailed. Then, we list special considerations for using REXX as the procedural language. Lastly, this chapter lists the manuals to research for more information.
Elements of AF/OPERATOR

AF/OPERATOR provides services that manage your console, schedule jobs, and issue operator commands to MVS and its subsystems. The architecture of AF/OPERATOR allows you to embed commands in REXX execs. For more information, see “Special Considerations for Using the REXX Language” on page 29.

This section describes components, services, and each phase of operation: setting traps, specifying system events, and manipulating data using variable services. The last subject discussed is addressing other products from AF/OPERATOR.

Components and Services

Figure 1 shows the relationship between the REXX and the product control language and also details the services and the components of the product.

The following terms define essential components of AF/OPERATOR:

- **traps**: Statements which contain user-specified criteria to be used in selecting system events, such as the type of event, the conditions under which it should be detected, and the action that will be taken.
- **matches**: Internal processes initiated when successful event detection occurs. A match constitutes a single automation event and uniquely identifies it for the environment and users.
- **variables**: Any text or numeric string that represents a value to be substituted later during execution.

AF/OPERATOR provides programming services in the form of Candle-supplied REXX functions, language, and control commands.
Product control commands

Commands that control the operation of the AF/OPERATOR product from the console or from within the product control language. They include the EXEC, PEER, and SHOW commands and also commands that begin with AF described at the beginning of “AF/OPERATOR Commands” on page 183.

Automation commands

**Functions**

Programming functions that receive data, process it, and return a value to the procedure. AF/OPERATOR provides a set of Candle-supplied REXX functions developed to supplement the built-in REXX functions provided by IBM REXX. They are listed in “Candle-Supplied REXX Functions for Variable Handling” on page 32.

**Host commands**

A set of commands that carries out system automation actions when a trap is triggered and a match is created. They are listed in “AF/OPERATOR Commands” on page 183.
The following diagram lists examples of the commands and functions provided by AF/OPERATOR. This is not a complete list of those available.

<table>
<thead>
<tr>
<th>Candle-Supplied REXX Functions</th>
<th>Host Commands</th>
<th>Product Control Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIVE</td>
<td>TRAP</td>
<td>AF CANCEL</td>
</tr>
<tr>
<td>DAYOFWK</td>
<td>ALLOC</td>
<td>EXEC</td>
</tr>
<tr>
<td>JOBSTEP</td>
<td>LOGON</td>
<td>PEER START</td>
</tr>
<tr>
<td>MATCH</td>
<td>NVALERT</td>
<td>PEER LINK STOP</td>
</tr>
<tr>
<td>STOP</td>
<td>OPER</td>
<td>SHOW FILES</td>
</tr>
<tr>
<td>SECONDS</td>
<td>WTO</td>
<td>WTO</td>
</tr>
<tr>
<td>HMMNSS</td>
<td>etc.</td>
<td>etc.</td>
</tr>
</tbody>
</table>

**Traps and Matches**

Automation commands work by setting traps to execute simple commands or to execute more complicated REXX execs when specified system events take place. REXX execs can call other REXX execs which employ automation commands and Candle-supplied REXX functions. They can also create and access variables supplied by AF/OPERATOR, another automation application, or another operator.

For example, you can set up a trap that detects when a production CICS™ region abends and then changes the color of a corresponding OMEGAVIEW® status bar.

This next section discusses traps, matches, and important commands for executing trap actions. It concludes with examples of using AF/OPERATOR or the REXX language to address other products.

**Trap Components**

Trap components must include:

1. The type of event being trapped:
   - WTO—write-to-operator messages
   - CMD—console commands
   - TOD—time-of-day specifications
   - xtype—OMEGAMON® exceptions.

2. The conditions or pattern(s) under which an event will be recognized. Pattern recognition is a variable service described in “Classes of Variables” on page 21.

3. Other selection and command criteria such as jobname, class, log, event origin or test.
4. The automation actions to be taken upon successful event detection (matches). There are three types of actions that traps may perform:

**Immediate**
Specific actions that occur as soon as AF/OPERATOR initiates the match. These include suppressing a message or command, and highlighting or changing the route code of a message.

**Simple extended**
Single action that you can specify in addition to immediate actions. These include issuing a single MVS command, sending a message to TSO users, adding/deleting another AF/OPERATOR trap, and so forth.

**Complex extended**
Actions that involve multiple instructions to AF/OPERATOR and/or conditional logic. These include issuing multiple subsystem commands in response to a single command typed at the console and dynamically deleting traps at a certain time of day.

The TRAP ADD command, shown below, illustrates how the components of a trap are specified:

```
TRAP ADD(trapname) traptype('pattern') actions
```

This is an example of a trap that detects when a CICS region has abended and takes the complex action of changing an OMEGAVIEW status bar to red

```
TRAP ADD(CICSDOWN) WTO('IEF450I') jobname(CICSPROD), action('EX MVCHANGE
"CICSPROD CRITICAL"')
```

Certain keywords of the TRAP command specify *immediate actions* that perform important automation tasks, such as managing messages. In addition, you may use the trap command to route selected messages to other MVS systems and display them on consolidated consoles. You may also want to perform *complex extended actions*, like executing a REXX exec and sending commands to external environments.

To perform complex extended actions, use the ACTION keyword in a trap statement followed by any product control command. See “Special Considerations for Using the REXX Language” on page 29 for an explanation of important commands you will need for programming complex actions. For details on their syntax and examples of their use, see the alphabetical listing in “AF/OPERATOR Commands” on page 183.
Executing a REXX exec

One of the product control commands you can specify is EXEC, which starts the execution of a REXX exec. You can use the EXEC command in a TRAP ADD statement with the ACTION parameter or within the body of the REXX exec.

The EXEC command allows you to access global variable pools and pass parameters back and forth. The section “Differences in Passing Parameters” on page 38 tells how to pass parameters between the product control language and REXX.

Addressing Other Products from AF/OPERATOR

AF/OPERATOR provides special commands for communicating directly with other products, such as VM (the CP command), JES3 (the JES3OPER command), NetView® (the NVALERT command), MVS or one of its subsystems (the OPER command), and OMEGAVIEW.

AF/OPERATOR can also communicate with specific VTAM™ sessions, like OMEGAMON and the Programmerless Open VTAM Interface (POVI), by logging on and issuing the AFADDR command. The sessions can be established and addressed by the application name you determine when you issue the LOGON command. The format for the LOGON command is:

```
LOGON OMMVS APPLID(applid) NAME(sessname)
```

Sample REXX Language Address

Here is how the logon and address would appear in a REXX exec:

```/* REXX */
ADDRESS AFHOST
"LOGON OMMVS APPLID(OMVTAM) NAME(OMEGAMON) INTERVAL(00:01:00)",
"USERID(ABC99/AOPASS)"
"AFADDR OMEGAMON"
"JOBN MYJOB"
"STEP"
"'WAIT'"
"AFADDR"
```

This sequence will set AF/OPERATOR as the current REXX environment, log onto the VTAM application representing OMEGAMON for MVS, and send a command to that OMEGAMON session.

When using the AFADDR command to send OMEGAMON commands from a REXX exec, you must include the “AFADDR” line at the end of the sequence. Since the OMEGAMON commands are accumulated in a buffer, this line is necessary to clear the buffer and force the OMEGAMON commands to be processed. Otherwise, the OMEGAMON commands will not be issued.

The difference between the REXX ADDRESS command and the AFADDR command (which is downwardly compatible with AF/OPERATOR ADDRESS) is the destination of the command. REXX ADDRESS sends commands to the specified host environment, while AFADDR applname (or the AF/OPERATOR ADDRESS command) sends commands to either POVI or an OMEGAMON. In the above example, the AFADDR command used without a qualifier sets the current host environment back to the previous host environment.
The following example shows how the same logon and address would appear using a function:

```rexx
/* REXX */
Call OMSGL 'SYSA','OMMVS','APPLID(OMSYSA)'
Call OMEGAMON 'JobN myJob;step;wait;'stack'
```
Special Considerations for Using the REXX Language

You may want to use the REXX language for its advanced program control capabilities and data manipulation features. The following figure represents the components and services of the REXX language environment.

FIGURE 2. REXX Language Environment

<table>
<thead>
<tr>
<th>REXX Language Environment</th>
<th>Local Variable Pool</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Program Coordination Services</strong></td>
<td><strong>Local variables</strong></td>
</tr>
<tr>
<td>Procedural instructions</td>
<td>(Variables used in the REXX execs)</td>
</tr>
<tr>
<td>For example: SIGNAL DO SAY RETURN EXIT</td>
<td>For example: ARG() DATE() FIND() LENGTH() SIGN() SYMBOL()</td>
</tr>
<tr>
<td>Variable assignment instructions</td>
<td>For example: symbol= ARG PULL REXX functions GLBVGET() SYSVGET() SYSVPUT() GLBVPUT()</td>
</tr>
</tbody>
</table>

As shown in the diagram, program coordination services include procedural instructions, variable assignment instructions, and the standard REXX functions supplied in TSO/E REXX. These instructions and functions are documented in IBM’s TSO/E REXX/MVS Reference; differences in the AF/OPERATOR implementation of REXX are discussed in this section.

Compatibility with MVS and TSO/E Environments

AF/OPERATOR supports the MVS and TSO/E REXX environments only to the extent required for functionality such as that provided by the Resource State Manager and OMEGACENTER Status Manager. Thus, most MVS and TSO/E REXX functions either are not supported at all (as, for example, TCPIP Sockets, ISPF tables and variables, etc.), or have minimal support in the AF/OPERATOR environment (as, for example, ALLOC, MAKEBUF, etc.).
Specifying the Procedure as a REXX Exec

A REXX exec must contain a comment in the first line that contains the word REXX as shown in the example. (REXX can be in either upper or lower case.)

/* REXX */
SAY "HELLO WORLD"
EXIT

Addressing Other Environments

Product control commands are issued from REXX execs through a REXX facility known as the host command interface. Whenever the REXX interpreter encounters an instruction that it does not recognize, it passes that instruction to the current host environment for processing. You can change the current environment explicitly by using the REXX ADDRESS instruction.

The name of the AF/OPERATOR environment is AFHOST, and that environment is established as the default each time a REXX exec is invoked. While AFHOST is the only environment supported by AF/OPERATOR, any environment that is provided by the MVS host is also accessible from AF/OPERATOR. Because these environments are subject to change by IBM, no specific list is provided here. The names of the MVS host environments can be found in the IBM OS/390 TSO/E REXX Reference.

Note: Since AF/OPERATOR runs in a non-TSO MVS address space, you cannot access TSO or ISPF services from REXX execs running under AF/OPERATOR. Specifically, the following host environments cannot be addressed: ISPEXEC, ISPEDIT, and TSO. This limitation also applies to other vendors who supply REXX environments in their own address space.

Addressing AF/OPERATOR

Since REXX passes unrecognized commands to the current or default host environment, you could pass commands to AFHOST by simply including them in the command stream. Good programming practices, however, dictate that you include the REXX command

ADDRESS AFHOST

to specify explicitly that you want non-REXX text passed to AF/OPERATOR.

If you change the host environment to MVS or ATTACH, then you must change it back to the AF Host Environment before you can issue subsequent product control commands.

One other convention is worth noting. You may want to execute a product control command but change the current environment for one command only; to do this, place the desired AF/OPERATOR command on the same line as an ADDRESS command. That is, the sequence:

ADDRESS MVS

ADDRESS AFHOST “TRAP ADD(MYTRAP) TOD(*) ACTION(‘EXEC execname’)”

will change the host environment to MVS and then execute the AF/OPERATOR TRAP command while retaining MVS as the current host environment.
**Addressing TSO/E REXX Commands to MVS**

As noted previously, you cannot address the TSO environment from the AF/OPERATOR implementation of REXX; however, certain TSO/E REXX commands do function if addressed to MVS. Table 1 on page 31 shows which commands you can and cannot use in this environment:

<table>
<thead>
<tr>
<th>Supported Using ADDRESS MVS</th>
<th>Not Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELSTACK</td>
<td>EXECUTIL</td>
</tr>
<tr>
<td>DROPBUF</td>
<td>HI</td>
</tr>
<tr>
<td>MAKEBUF</td>
<td>HT</td>
</tr>
<tr>
<td>NEWSTACK</td>
<td>RT</td>
</tr>
<tr>
<td>QBUF</td>
<td>TE</td>
</tr>
<tr>
<td>QELEM</td>
<td>TS</td>
</tr>
<tr>
<td>QSTACK</td>
<td></td>
</tr>
<tr>
<td>SUBCOM</td>
<td></td>
</tr>
</tbody>
</table>

For example, to send the NEWSTACK command to the MVS environment, code the following:

```/* REXX */
  . . .
  address MVS 'NEWSTACK'
```

**TSO/E External Functions Not Supported**

The following TSO/E external functions are not supported in the AF/OPERATOR REXX environment:

- LISTDSI()
- MSG()
- OUTTRAP()
- PROMPT()
- SYSDSN()
- SYSVAR()
**Issuing Automation Commands**

When you place automation commands in your execs, you must enclose them in single or double quotes. The quotes tell REXX to ignore the command; REXX does not try to interpret a string in quotes. REXX passes the command directly to the current host command environment, which in this case is the AF Host Environment.

The statement

```
“WTO ‘Job YEAREND is not active”
```

will cause the AF/OPERATOR command, WTO, to be executed, resulting in a console display of the message **Job YEAREND is not active**. If you omit the quotes, REXX will perform a standard library search for the first module called WTO that it can execute. This scenario causes unpredictable results, including unexpected abends.

*Note:* We recommend using double quotes around all commands directed to AF/OPERATOR to avoid confusion with nested single quotes, such as those that would result when specifying a literal string as in the example above.

**Exchanging Variable Values Between REXX and AF/OPERATOR**

This section describes the special functions supplied for variable manipulation in REXX. It also discusses AF/OPERATOR variables that require special handling in the REXX environment.

You can also create REXX shared variables (RSVs), which can be used by all execs and may be preserved after the AF/OPERATOR started task terminates. For more information, see “REXX Shared Variables” on page 65.

**Candle-Supplied REXX Functions for Variable Handling**

In REXX, you must use Candle-supplied REXX functions to transfer data between local REXX variables and AF/OPERATOR, and also to transfer data between local REXX variables and REXX shared variables. For definitions of system variables and global variables, see “Scope of Variables” on page 43.

The following figure illustrates the functions provided to handle the transfer. Examples of these functions are provided in “Candle-Supplied REXX Functions” on page 93.
Reading from a File

The READ command transfers a record from a file into an AF local variable only. REXX execs, however, can only access system or global variables. READ must be given a global scope first in order to be accessed by a REXX exec. To transfer the value and make it global, insert the following lines into your exec:

```plaintext
VARNAME = " "
R = GLBVPUT("VARNAME")
"READ SYSIN VARNAME"
```
Deleting System and Global Variables
AF/OPERATOR provides specific Candle-supplied REXX functions for deleting system and global variables. They are SYSVDEL and GLBVDEL.

REXX EXECIO Support
This feature of AF/OPERATOR provides support for the REXX command, EXECIO. You can use EXECIO in REXX execs you write that run under AF/OPERATOR.

Purpose
New execs you write can use the EXECIO command for input/output tasks instead of using the AF/OPERATOR automation language commands: OPEN, CLOSE, READ, and WRITE. Execs that use the EXECIO command for I/O tasks require less coding effort than execs that use AF/OPERATOR automation language commands and are more portable.

Scope of files opened
Files opened by EXECIO have a scope of LOCAL (only the calling exec can access the file) on an AF/OPERATOR system.

Implementation
In order to support the EXECIO command in execs running under AF/OPERATOR, this feature replaces the IBM-supplied I/O routine, IRXINOUT, with a Candle-supplied I/O routine, KOGRXIO. Refer to the IBM TSO Extensions Version 2 REXX/MVS Reference for more information on the IBM replaceable input/output routine used by the EXECIO command.

REXX Examples
This section compares examples of REXX execs that perform I/O tasks. Examples using embedded AF/OPERATOR automation controls (OPEN, CLOSE, READ, and WRITE) are followed by examples using the EXECIO command to perform the same task.

Reading data into variables
The following example uses AF/OPERATOR automation commands to read data from a file into REXX stem variables for later processing.
Here is the same example using EXECIO.

```rexx
/* REXX */
ADDRESS AFHOST
"ALLOC DDNAME(IFILE) DSN('SAMPLE.INPUT.FILE')"
"OPEN IFILE INPUT"
data = ""
data.0 = 0
status = glbvput('data')
"READ IFILE DATA"
status = glbvput('data')
do while(data ^= '')
i = i + 1
data.i = data
"READ IFILE DATA"
status = glbvget('data')
end
data.0 = i
"CLOSE FILE"
"FREE DDNAME(IFILE)"
return
```

```rexx
/* REXX */
ADDRESS AFHOST
"ALLOC DDNAME(IFILE) DSN('SAMPLE.INPUT.FILE')"
ADDRESS MVS "EXECIO * DISKR IFILE (STEM DATA. FINIS)"
"FREE DDNAME(IFILE)"
return
```
Reading and writing data between files

The following example uses AF/OPERATOR automation commands to read data from one file and write it to another.

Here is the same example using EXECIO.
The following example uses the EXECIO command to read and write a fixed number of records at a time from one file to another. This may be necessary if the file is too large for available storage.

```rexx
/* REXX */
ADDRESS AFHOST
records_per_block = 100
"ALLOC DDNAME(IFILE) DSN('SAMPLE.INPUT.FILE')"
"ALLOC DDNAME(OFILE) DSN('SAMPLE.OUTPUT.FILE')"
eof = 0
do while eof = 0
   ADDRESS MVS "EXECIO" records_per_block "DISKR IFILE (STEM DATA.)"
   if rc > 0 then eof = 1
   ADDRESS MVS "EXECIO" records_per_block "DISKW OFILE (STEM DATA.)"
end
ADDRESS MVS "EXECIO 0 DISKR (FINIS)"
ADDRESS MVS "EXECIO 0 DISKW (FINIS)"
"FREE DDNAME(IFILE)"
"FREE DDNAME(OFILE)"
return
```

**Where to find more information**

Refer to the IBM TSO Extensions Version 2 REXX/MVS Reference for the syntax, usage, and return codes of the EXECIO command.

**Passing Condition Codes Between REXX and AF/OPERATOR**

AF/OPERATOR supplies a set of variables (global match variables) initialized automatically when a match occurs. All AF/OPERATOR-supplied global match variables listed in “AF/OPERATOR Global Match Variables” on page 52 can be accessed from within REXX execs, except for AOLASTCC, AOMAXCC, and AOENDCC, which require special consideration.

AF/OPERATOR commands generally set condition codes, reflected in the global variables AOLASTCC, AOMAXCC, and AOENDCC. These variables cannot be obtained using GLBVGET. In addition, AF/OPERATOR commands issued from within REXX execs will not generate these values.

For the AOLASTCC variable in the REXX language, this information is available in the REXX standard variable, RC.

The AF/OPERATOR variable, AOMAXCC, returns the highest AOLASTCC value set by statements that have been executed in the current match. There is no equivalent for AOMAXCC in REXX; however, you can simulate it by using the RC variable and the MAX() REXX function.

AOENDCC, a variable that allows you to set a threshold value that specifies the highest AOLASTCC value before termination, has its equivalent in REXX’s SIGNAL ON and CALL ON instructions. AOENDCC does not affect the execution of REXX execs.
Differences in Passing Parameters

The automation command EXEC allows you to pass a parameter list; however, you need to take into account the different ways in which REXX execs may interpret parameter strings. The REXXARGS startup option selected—either REXXARGS(TSO) or REXXARGS(AF)—determines the way in which AF/OPERATOR interprets them. The only time you can set the REXXARG option is during initialization. If you wish to change it, you need to reinitialize AF/OPERATOR. You may also need to recode your execs because the new setting will affect their interpretation.

The format of the EXEC command is:

```
EX execname 'parm1 parm2 parm3 . . .'
```

Under REXXARGS(TSO)

If you have specified the REXXARGS(TSO) startup option, parameters passed to REXX execs are treated differently. A REXX exec receives parameters delimited by blanks as a single parameter. For example, the AF/OPERATOR command

```
"EX execname ""HELLO THERE OPERATOR"
```

will interpret parameters as follows:

**REXX exec**

A parameter whose contents are ‘HELLO THERE OPERATOR’ (The parameters inside the quotes are passed as one parameter whether or not there are spaces between words in the statement.)

If you want to separate the above parameters in the REXX exec, you must pass the parameters individually.

```
EX execname ‘HELLO’, ‘THERE’, ‘OPERATOR’
```

Under REXXARGS(AF)

If you have specified the REXXARGS(AF) startup option (the default), REXX execs receive parameters separately. The sample REXX code below converts parameters from the way they are received under REXXARGS(AF) to the way REXX execs normally receive them under TSO/E (as one parameter).

```
"EX execname ‘HELLO THERE OPERATOR’"
```

```r
PRM= "" /* null parm string */
DO I = 1 TO ARG() /* loop for # parameters */
   PRM = PRM ARG(I) /* concatenate to parmstring */
END
```

At the end the value of PRM is ‘HELLO THERE OPERATOR’ where ARG(1) is ‘HELLO’, ARG(2) is ‘THERE’ and ARG(3) is ‘OPERATOR’. 
Using Quotes in Parameter Strings

The number of single quotes changes at execution time. Note the following rules to maintain proper syntax in your REXX execs:

- The outer quotes—that is, quotes that enclose the parameter string—are never passed and are stripped from the parameters at execution time.
- Two consecutive single quotes become one after execution. For example:
  
  ‘CAN’T INITIALIZE CICS’

  becomes

  CAN’T INITIALIZE CICS

- To represent a single quote, use two consecutive single quotes as shown in the example above.
Where to Look for More Information

This section briefly describes the manuals that you can consult for information on the different language elements.

FIGURE 4. Language Reference Manuals

AF/OPERATOR Language Information

The AF/OPERATOR User's Guide explains general concepts about the product functions, such as traps, use of pattern characters, and variable handling. It also describes the specialized facilities of AF/OPERATOR, such as the Programmerless Open VTAM Interface and the OG*TSO testing facility.

REXX Language Information

All statements comprising the REXX language are described in the IBM publication TSO/E REXX/MVS Reference. Consult this manual for general information about REXX and for detailed syntax on specific REXX instructions and REXX functions.
Introduction

Variables are important elements of the product control language. A variable is any text or numeric string that represents a value to be substituted later during execution. Variables are useful because they allow you to save information you may want to refer to later.

Chapter Contents

About Variables ......................................................... 42
Characteristics of Variables ........................................... 43
Rules for Naming Variables ........................................... 46
Defining Variables ..................................................... 47
How AF/OPERATOR Interprets Variables ......................... 48
AF/OPERATOR-Supplied Variables ................................. 50
About Variables

This chapter provides examples of variables and then explains their attributes, including type, scope, and class. The discussion covers the naming, definition, modification, deletion, and interpretation of variables. The chapter concludes with tables of AF/OPERATOR-supplied, match, JES2 Interface Match, and line variables. AF/OPERATOR variables are frequently used in REXX execs. AF/OPERATOR also provides pools of variables that can be accessed by a REXX exec. The following provides several examples of variables used in a REXX exec:

```/* REXX */
ARG JOB
ADDRESS AFHOST
DO WHILE stopflg ^= 'YES'
   "WTO 'Task "JOB" has ended'"
   "WTOR '(RESTART) reply A to abort and U to restart' reply(answer)"
   "DOM &WTONUM"
GLBVGET(answer)
temp = substr(answer,1,1)
if temp = ' ' THEN DO
   NULLANS = NULLANS - 1
   "WTO 'Multiple null responses - match ended'"
   stopflg = 'YES'
   END
ELSE if temp = 'A' THEN DO
   stopflg = 'YES'
   END
ELSE if temp = 'U' THEN DO
   ADDRESS AFHOST,
   "OPER 'TEXT'"
   stopflg = 'YES'
   END
ELSE DO
   "WTO ''Invalid response:" answer "Please reenter!!''''"
   END
END
RETURN(0);```
Characteristics of Variables

AF/OPERATOR maintains variable pools (or groups) that contain data that REXX execs may share. When using variables in your procedures, you should be aware of their attributes and also of the different ways of creating them. Variables have the following attributes:

- type and length
- scope
- class

Type and Length of Variables

There are three types of variables:

**Character (CHAR)**
Contains character strings as values. You can declare the length of a character variable; the system can store up to 255 characters per variable. The amount of storage reserved for the variable depends on the number of characters in the string at the time.

**Integer (INT)**
Contains integers as values. Integer variables are 4 bytes long.

**Hexadecimal (HEX)**
Contains hexadecimal numbers as values. Hexadecimal variables are 4 bytes long.

*Note:* When reading in files, the default is to truncate character variables. For this reason, any blank record in your input file may be interpreted as a null character, resulting in a premature end-of-file (EOF). If you cannot avoid having blank lines in the input file, consider using character variables prefixed with Z99. AF/OPERATOR variables prefixed by Z99 are not truncated.

Scope of Variables

**Local Variables**
A local variable can be referenced only in the REXX exec where it is defined, and its contents are accessible only from a single REXX exec. The contents are stored in variable pools maintained by REXX.

**Global Variables**
AF/OPERATOR creates global variables during the processing of a match and deletes them when the match is completed. AF/OPERATOR also creates many variables automatically at match time, such as variables containing the jobname and jobtype of the issuer of a WTO or command.

Global variables hold data accessible to all REXX execs called within a given match process. Every match is assigned a single global pool. This allows each match to isolate its global data from other matches that may be executing with it, as well as allowing the sharing of variables within a match process.

Use the following commands to access global variables from REXX execs:

- GLBVGET
Characteristics of Variables

- GLBVPUT
- GLBVDEL
- GLBVUPD

System Variables
When you start AF/OPERATOR, certain system variables are established. System variables exist for as long as AF/OPERATOR is active and can be used in any REXX exec by issuing the SYSVGET function. They may also be retained between runs of AF/OPERATOR by using the checkpointing facility. See the AF/OPERATOR Configuration and Customization Guide for additional information on checkpointing.

You can change the values of system variables by writing a REXX exec that uses SYSVPUT and SYSVUPD.

You can define additional system variables, but you cannot delete any system variable that begins with the letters AO (whether system- or user-defined) because these are reserved for AF/OPERATOR-defined variables. System variables contain data accessible from all REXX execs executing in AF/OPERATOR. This allows sharing between matches.

Note: In REXX, you can access data in global and system variables by transferring the data between REXX local variables and AF/OPERATOR variables via Candle-supplied REXX functions. Refer to “Special Considerations for Using the REXX Language” on page 29 for information.

REXX Shared Variables
REXX shared variables (RSV) can be created and stored in a pool so that they are accessible until AF/OPERATOR ends.

You can create and manage a pool of REXX shared variables by using the following functions:
- SHARVPUT
- SHARVGET
- SHARVUPD
- SHARVDEL

REXX shared variables must be created from a REXX exec. Shared variables in the SYSTEM RSV pool can be used by all execs executing within a single AF/OPERATOR address space. Shared variables in the SYSPLEX RSV pool can be used by all REXX execs executing within any AF/OPERATOR address space sharing the same XES cache structure. RSVs are available automatically at AF/OPERATOR initialization.

A REXX shared variable is preserved after the exec that created it terminates. You can even request that the shared variable’s value be preserved after the AF/OPERATOR started task terminates. See “REXX Shared Variables” on page 65 for more information on REXX shared variables and functions.
Classes of Variables

All classes of variables supplied by AF/OPERATOR are *symbolic*; that is, the variable contents are substituted as each statement is interpreted for execution.

Besides defining a variable in a REXX exec, AF/OPERATOR allows substitution using the following symbolic variables:

**Line**
- Line variables capture the text responses to operator commands or multi-line screen displays. See the “Line Variables” on page 60 for more information.

**AF/OPERATOR-supplied**
- AF/OPERATOR supplies system and global variables whose values are defined automatically. System variables are set at AF/OPERATOR initialization. Certain global variables (also called *match* variables) are set when a match occurs, and their values are available to any REXX exec executed under that match. These variables are listed in “AF/OPERATOR-Supplied Variables” on page 50.

  **Note:** See also “Special Considerations for Using the REXX Language” on page 29 for exceptions to the match variable values in the REXX environment.

**Pattern**
- Pattern variables are generic text strings you create by combining explicit text with *wildcard* characters, such as question marks (?) and asterisks (*). WTO, CMD, and xtype traps allow text to be captured by pattern variables. Pattern matching is very useful for trapping or displaying messages, commands, or exceptions. Pattern matching is discussed more extensively in “Pattern Matching” on page 83.

**AOTXT**
- AOTXT variables capture a MLWTO (multi-lined WTO) when the MLWTO(FIRST) startup parameter is specified in your RKANPAR member.

**Internal**
- AF/OPERATOR generates and uses undocumented internal variables for housekeeping and message display. Some of these may appear when you list variables with the SHOW VARS command.
Rules for Naming Variables

A variable name can consist of up to 8 alphanumeric or symbols with an alphabetic first character and must *not* be one of the following reserved words:

- AND
- ELSE
- END
- EQ
- GE
- GOTO
- GT
- IF
- LE
- LINKID

**Note:** Do not begin user-defined variables with the letters AO or OG. Variable names that begin with AO and OG are reserved for AF/OPERATOR-defined variables. If AF/OPERATOR encounters such a variable, it does not issue an error message.
Defining Variables

Variables can be defined by the system (implicitly) or in patterns by using REXX.

User- Versus AF/OPERATOR-Defined Variables

User-defined variables are:

- implicitly defined in a pattern
- defined in REXX using SYSVPUT and GLBVPUT

System-defined variables can be:

- predefined by AF/OPERATOR
- defined by your programmer

You can use SYSVGET and SYSVPUT to access these variables.

AOSMFID, which defines the host system name, is an example of a system-defined variable. See “AF/OPERATOR-Supplied Variables” on page 50 for a complete list of AF/OPERATOR-supplied system variables.

Any command executed as a result of a trap match has access to the global variables listed in “AF/OPERATOR Global Match Variables” on page 52.

Pattern Variables Defined in Patterns

A pattern variable is created by pattern-matching, a process used by AF/OPERATOR to compare a user-specified pattern with a string of target characters to determine whether they match. You create the variable but the system assigns its value as event text; that is, text that matches the pattern in a message or command. You can extract information from event text and place it in pattern variables by using AF/OPERATOR pattern characters with variable names, as described in “Pattern Variables” on page 88. Pattern variables are always global and character type.

Note: Avoid using names of AF/OPERATOR system, match, or AF/OPERATOR-reserved variables.
How AF/OPERATOR Interprets Variables

Substituting the value of a variable for its variable or symbolic name is called symbolic substitution. The following section discusses symbolic substitution, obtaining the value of a variable, and determining the number of ampersands.

In AF/OPERATOR, the variable name is preceded by one or two ampersands (&).

Symbolic variables, with ampersand prefixes, are required:
- in text enclosed in quotes (as in WTO 'THE FAILING TASK IS: &TASK')
- in the ACTION field of a TRAP ADD or TRAP CHANGE command, since that field is enclosed in quotes
- in fields enclosed in parentheses
- for conversion from character to numeric data

For example, symbolic substitution occurs in a statement enclosed in quotes, such as:

WTO 'STORAGE IN UNIT &UNITID IS NEARLY EXCEEDED'

The compiler substitutes the value of UNITID for the symbolic variable name &UNITID and issues the message:

STORAGE IN UNIT 2 IS NEARLY EXCEEDED

AF/OPERATOR may truncate leading and trailing blanks during variable substitution, unless the variable is specified notrunc, which preserves the blanks. This should be taken into consideration when calculating contents of character strings in operations such as substring.

Obtaining the Value of a Variable

When you want to retrieve the value of a variable, place an ampersand (&) before the variable name as in WTO 'THE FAILING TASK IS: &TASK'. The & is required, unless the variable is placed in an IF statement. The contents of the variable are substituted as each statement is interpreted for execution.

Obtain the value of a variable by preceding the variable name with one or two ampersands. Optionally, to mark the end of the variable name and insert text (an alphanumeric character) that follows, insert a period between the last letter of the variable name and the next alphanumeric character. When the value of the variable is substituted, the trailing text shows on the statement. The period is deleted when symbolic substitution takes place.

For example, if the value of the variable &SYSTEM is A, and the following command is executed:

OPER 'S CICS&SYSTEM.PRD'

the OPER command issued as a result of the statement above is

S CICSAPRD

Determining the Number of Ampersands

Symbolic substitution occurs when the trap fires. The number of ampersands placed in front of the variable determines when the value is substituted. When two ampersands are used,
AF/OPERATOR reduces the number to one when the REXX exec executes and reduces again (and thus deletes the remaining ampersand) when the trap fires. When one ampersand is used, AF/OPERATOR substitutes the value when the REXX exec executes and again when the trap fires. For example, the action parameter of a trap

```
“ACT('TPUT ”&AOGDATE” USER(&TSOID’)”
```

after execution, becomes

```
TPUT ‘&AOGDATE’ USER(SYS001)  <== ampersands reduced + &TSOID resolved
```

Then, when the trap fires, the statement becomes

```
02/10/17                           <== &AOGDATE resolved
```

**Console Execution**

The following example illustrates using variables within TRAP ADD commands from the console. When a TRAP ADD command executes, it defines a trap.

When you execute a TRAP ADD command from the operator console, place a single ampersand before the variable. Because this command is issued from the console, no compiling occurs to reduce the number of ampersands. Symbolic substitution occurs as soon as AF/OPERATOR fires the trap. The following console command adds a trap named TR123 that suppresses all WTO messages that start with IST. It then sends each message, with a date stamp, to TSO userid ABCD23.

```
TRAP ADD(TR123) WTO(‘IST’”) SUP ENA +
ACT(‘TPUT ”&AOTEXT &AOGDATE” TSOID(ABCD23’)
```

---

(*Variables in the AF/OPERATOR Environment*)
AF/OPERATOR-Supplied Variables

AF/OPERATOR automatically supplies a set of system variables, (global) match variables, JES2 Interface Match variables, and line variables. This section provides tables of these variables and also describes the global variable AOOMWAIT, which allows you to change the OMEGAMON interval.

AF/OPERATOR System Variables

The following table contains some of the system variables supplied by AF/OPERATOR. It is not a complete list. It contains only those system variables that are most commonly used.

Table 2. AF/OPERATOR-Supplied System Variables

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Can user assign?</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOARM</td>
<td>No</td>
<td>Char</td>
<td>ARM element structure name (if any) from the ARM parameter.</td>
</tr>
</tbody>
</table>
| AOASCII         | Yes              | Char  | REXX table that can be used with the REXX TRANSLATE function to convert EBCDIC characters to ASCII. For example:  
|                 |                  |       | stringb=TRANSLATE(stringa, AOASCII, AOEBCDIC)                              |
| AOCACHE         | No               | Char  | XES structure name (if any) from the CACHE parameter.                       |
| AOCMSAD         | No               | Int   | Address of KOGCMS module.                                                  |
| AOCMSLN         | No               | Int   | Length of KOGCMS module in bytes.                                          |
| AOCONSID        | No               | Int   | Console address space ID.                                                  |
| AOCONUSE        | Yes              | Char  | Highest console number used.                                               |
| AOCPULIM        | Yes              | Int   | Default maximum CPU seconds per match (Equals CPULIM in the parameter file). |
| AOCPUMOD        | No               | Hex   | CPU model number.                                                          |
| AOCPUSER        | No               | Hex   | CPU serial number.                                                         |
| AOCSAFRE        | No               | Int   | Bytes of CSA free space available for AF/OPERATOR.                         |
| AOCASASIZ       | No               | Int   | Bytes of CSA space allocated for AF/OPERATOR.                              |
| AOEBCDIC        | Yes              | Char  | REXX table that can be used with the REXX TRANSLATE function to convert ASCII characters to EBCDIC. For example:  
|                 |                  |       | stringb=TRANSLATE(stringa, AOEBCDIC, AOASCII)                              |
| AOLINKID        | No               | Char  | Same as AOLSYSNM. Contains value set for LINKID startup parm.               |
| AOLSYSNM        | No               | Char  | Contents of the LINKID parameter.                                          |
### Table 2. AF/OPERATOR-Supplied System Variables

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Can user assign?</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOMAXMAT</td>
<td>No</td>
<td>Int</td>
<td>Maximum number of concurrent matches. (Equals MAXMAT in the parameter file.)</td>
</tr>
<tr>
<td>AOMAXTRP</td>
<td>No</td>
<td>Int</td>
<td>Maximum number of traps allowed. (Equals MAXTRP in the parameter file.)</td>
</tr>
<tr>
<td>AOMSGGLIM</td>
<td>Yes</td>
<td>Int</td>
<td>Maximum number of messages per match. (Equals MSGGLIM in the parameter file.)</td>
</tr>
<tr>
<td>AONVICMD</td>
<td>No</td>
<td>Int</td>
<td>Number of NetView Interface commands received since the last NVISTART.</td>
</tr>
<tr>
<td>AONVICTM</td>
<td>No</td>
<td>Int</td>
<td>Total number of NetView Interface commands received since AF/OPERATOR was started.</td>
</tr>
<tr>
<td>AORLS</td>
<td>No</td>
<td>Char</td>
<td>AF/OPERATOR release number.</td>
</tr>
<tr>
<td>AOSMFID</td>
<td>No</td>
<td>Char</td>
<td>SMF system ID.</td>
</tr>
<tr>
<td>AOSCTAD</td>
<td>No</td>
<td>Hex</td>
<td>Address of AF/OPERATOR SSCT entry.</td>
</tr>
<tr>
<td>AOSUBSYS</td>
<td>No</td>
<td>Char</td>
<td>Name for the SUBSYS startup parm.</td>
</tr>
<tr>
<td>AOSYSCNM</td>
<td>No</td>
<td>Char</td>
<td>Name of the console specified in the CID() startup parameter. If the CID() startup parameter is not specified, then this is the name of the console from which the START command for AF/OPERATOR was issued.</td>
</tr>
<tr>
<td>AOSYSCON</td>
<td>No</td>
<td>Int</td>
<td>ID of the console specified in the CID() startup parameter. If the CID() startup parameter is not specified, then this is the ID of the console from which the START command for AF/OPERATOR was issued.</td>
</tr>
<tr>
<td>AOSYSID</td>
<td>No</td>
<td>Char</td>
<td>System name found in the MVS Communications Vector Table (CVT) or value of SYSID startup parm.</td>
</tr>
<tr>
<td>AOSYSNUM</td>
<td>No</td>
<td>Int</td>
<td>Number of matches since last restart of AF/OPERATOR.</td>
</tr>
<tr>
<td>AOTASK</td>
<td>No</td>
<td>Char</td>
<td>AF/OPERATOR task name.</td>
</tr>
<tr>
<td>AOTASKAS</td>
<td>No</td>
<td>Int</td>
<td>AF/OPERATOR address space ID.</td>
</tr>
<tr>
<td>AOVSADD</td>
<td>No</td>
<td>Int</td>
<td>Count of new variables written to the VSAM checkpoint dataset.</td>
</tr>
<tr>
<td>AOVSDEL</td>
<td>No</td>
<td>Int</td>
<td>Count of variables deleted from the VSAM checkpoint dataset.</td>
</tr>
<tr>
<td>AOVSMOD</td>
<td>No</td>
<td>Int</td>
<td>Count of variables in the VSAM checkpoint dataset that were modified.</td>
</tr>
<tr>
<td>AOVSN</td>
<td>No</td>
<td>Char</td>
<td>AF/OPERATOR version number.</td>
</tr>
</tbody>
</table>
Notes:
- If the MSGLIM value is reset using SYSVPUT, the new value will immediately take effect systemwide, even for the execution of the function that reset it.
- If the CPULIM value is reset using SYSVPUT, the new value will immediately take effect systemwide for other matches that start executing, but not for the execution of the function that reset it or for any other active matches.

AF/OPERATOR Global Match Variables

Any command executed as the result of a trap match has access to the global variables listed in the following table.

Important
Users should not create variables with the same name as any of the following match variables. Access to the value of the match variable will be lost or a non-zero value will be generated.

Table 3. AF/OPERATOR-Supplied Match Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Can user assign?</th>
<th>Type</th>
<th>Description</th>
<th>Available in this type of match</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOAREAID</td>
<td>No</td>
<td>Char</td>
<td>Console Area ID (A...Z).</td>
<td>-</td>
</tr>
<tr>
<td>AOASID</td>
<td>No</td>
<td>Int</td>
<td>Address space ID.</td>
<td>CMD WTO</td>
</tr>
<tr>
<td>AOAUTOTK</td>
<td>Yes</td>
<td>Char</td>
<td>An 8-byte string whose value is taken from the AUTO parameter in the MPFLSTxx member of PARMLIB. For an example please see note following table.</td>
<td>-</td>
</tr>
<tr>
<td>AOCASE</td>
<td>No</td>
<td>Int</td>
<td>Case number from WAIT command.</td>
<td>CMD TOD WTO COM</td>
</tr>
<tr>
<td>AOCLNTEX</td>
<td>No</td>
<td>Char</td>
<td>Extended client identifier information; available in the CONNECT_EXEC only.</td>
<td>-</td>
</tr>
<tr>
<td>AOCLNTID</td>
<td>No</td>
<td>Char</td>
<td>Client identifier information.</td>
<td>-</td>
</tr>
<tr>
<td>AOCMDNO</td>
<td>No</td>
<td>Int</td>
<td>Sequentially assigned unique command number.</td>
<td>CMD WTO</td>
</tr>
<tr>
<td>AOCONNID</td>
<td>No</td>
<td>Char</td>
<td>An 8-byte name identifying the connection through which the match was created.</td>
<td>-</td>
</tr>
<tr>
<td>AOCONNM</td>
<td>No</td>
<td>Char</td>
<td>Name of the console from which a given command is issued or a given WTO is sent.</td>
<td>CMD WTO</td>
</tr>
<tr>
<td>AOCONS</td>
<td>No</td>
<td>Int</td>
<td>ID of console to which trapped message is routed.</td>
<td>CMD WTO</td>
</tr>
<tr>
<td>AOCOUNT</td>
<td>No</td>
<td>Int</td>
<td>Count of trap matches.</td>
<td>CMD TOD WTO X</td>
</tr>
<tr>
<td>Variable</td>
<td>Type</td>
<td>Class</td>
<td>Description</td>
<td>Location</td>
</tr>
<tr>
<td>---------------------</td>
<td>------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>AOCTFLG</td>
<td>Int</td>
<td>Control</td>
<td>Maximum CPU seconds for the current match. This may differ from the default CPU limit, set by CPULIM in the parameter file, when the CPULIM REXX function is used.</td>
<td>CMD</td>
</tr>
<tr>
<td>AOCPULIM</td>
<td>Yes</td>
<td>Int</td>
<td>Maximum CPU seconds for the current match. This may differ from the default CPU limit, set by CPULIM in the parameter file, when the CPULIM REXX function is used.</td>
<td>TOD WTO X</td>
</tr>
<tr>
<td>AODESC</td>
<td>No</td>
<td>Hex</td>
<td>Descriptor code(s) of trapped message. For an example see the following table.</td>
<td>WTO</td>
</tr>
<tr>
<td>AODOMNO</td>
<td>No Int</td>
<td>Unique message number of the WTO being DOM’d. Equivalent to the global variable AOWTONO in a WTO match.</td>
<td>WTO</td>
<td></td>
</tr>
<tr>
<td>AODOMWTN</td>
<td>No Char</td>
<td>Name of the original WTO trap (with the DOMACT specification) that produced the temporary DOM trap for this event.</td>
<td>WTO</td>
<td></td>
</tr>
<tr>
<td>AODSPFLG</td>
<td>No Int</td>
<td>Disposition flag.</td>
<td>WTO</td>
<td></td>
</tr>
<tr>
<td>AOEBCFLG</td>
<td>No Char</td>
<td>ECB flag.</td>
<td>WTO</td>
<td></td>
</tr>
<tr>
<td>AOENDCC</td>
<td>Yes Int</td>
<td>User established maximum condition code. If exceeded, the match is terminated. This variable is present only if the trap action invokes a command file. <strong>Note:</strong> While AOENDCC can be used in a REXX exec and returns a correct value, it does not affect the REXX exec itself.</td>
<td>CMD TOD WTO X COM</td>
<td></td>
</tr>
<tr>
<td>AOERRLN</td>
<td>No Char</td>
<td>Error line text or command line text.</td>
<td>CMD TOD WTO X COM</td>
<td></td>
</tr>
<tr>
<td>AOGDATE</td>
<td>No Int</td>
<td>Current Gregorian date (mm/dd/yy).</td>
<td>CMD TOD WTO X COM</td>
<td></td>
</tr>
<tr>
<td>AOGDATE4</td>
<td>No Int</td>
<td>Current Gregorian date (mm/dd/yyyy).</td>
<td>CMD TOD WTO X COM</td>
<td></td>
</tr>
<tr>
<td>AOHCPYID</td>
<td>No Char</td>
<td>Hard copy ID.</td>
<td>WTO</td>
<td></td>
</tr>
<tr>
<td>AOJCLAS</td>
<td>No Char</td>
<td>Jobclass of trapped message or command. (Same as JES2 variable if JES2 is running.)</td>
<td>WTO</td>
<td></td>
</tr>
<tr>
<td>AOJID</td>
<td>No Char</td>
<td>Job number, STC number, TSU number, or console number.</td>
<td>WTO</td>
<td></td>
</tr>
<tr>
<td>AOJNAME</td>
<td>No Char</td>
<td>Jobname, taskname, or TSO user ID.</td>
<td>WTO</td>
<td></td>
</tr>
<tr>
<td>AOJTYPE</td>
<td>No Char</td>
<td>Job type (IMS, JOB, STC, TSU).</td>
<td>WTO</td>
<td></td>
</tr>
<tr>
<td>AOLASTCC</td>
<td>No Int</td>
<td>Condition code from last operation. This variable is present only if the trap action invokes a command file. <strong>Note:</strong> this variable may return an invalid value when used in a REXX exec.</td>
<td>CMD TOD WTO X</td>
<td></td>
</tr>
<tr>
<td>AOLID</td>
<td>No Char</td>
<td>LINKID of transmitting address space or, if not remote, the current address space.</td>
<td>WTO</td>
<td></td>
</tr>
<tr>
<td>AOLINTYP</td>
<td>No Int</td>
<td>Line type flag</td>
<td>WTO</td>
<td></td>
</tr>
</tbody>
</table>
### AF/OPERATOR-Supplied Variables

**Table 3. AF/OPERATOR-Supplied Match Variables**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
<th>CMD</th>
<th>TOD</th>
<th>WTO</th>
<th>X</th>
<th>COM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOMATENT</td>
<td>No Bin</td>
<td>Match control block contents.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOMATNUM</td>
<td>No Int</td>
<td>Match number from match entry.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOMAXCC</td>
<td>No Int</td>
<td>Maximum condition code within current match. It should be noted that this variable may return an invalid value when used in a REXX exec.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOMCSFLG</td>
<td>No Int</td>
<td>MCS flags.</td>
<td></td>
<td></td>
<td>WTO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOMLWFLG</td>
<td>No Int</td>
<td>MLWTO flag.</td>
<td></td>
<td></td>
<td>WTO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOMSCFLG</td>
<td>No Int</td>
<td>Miscellaneous flag.</td>
<td></td>
<td></td>
<td>WTO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOMSGPFX</td>
<td>No Char</td>
<td>Message prefix (for example, TIME and JOB/STC/TSU NUMBER)</td>
<td></td>
<td>WTO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOMSGTYP</td>
<td>No Int</td>
<td>Message type flags</td>
<td></td>
<td>WTO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AONVCLST</td>
<td>No Char</td>
<td>Name of NetView exec that issued the AOCMD REXX function. This variable only pertains to NetView matches.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOPTYPE</td>
<td>No Char</td>
<td>Protocol type of connection; available in the CONNECT_EXEC only.</td>
<td></td>
<td></td>
<td>WTO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOREPID</td>
<td>No Int</td>
<td>Reply number - only for WTOs.</td>
<td></td>
<td>WTO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOROUT</td>
<td>No Hex</td>
<td>Message route code. For an example please see note following table.</td>
<td></td>
<td>WTO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOSID</td>
<td>No Char</td>
<td>The system name where the original WTO or CMD match occurred. For local systems, value of AOSYSID variable. For remote systems, system name of the transmitting address space. (IMS region if jobtype is IMS).</td>
<td></td>
<td>WTO</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOSTATUS</td>
<td>No Int</td>
<td>Status flag.</td>
<td></td>
<td>WTO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOTEXT</td>
<td>Yes Char</td>
<td>Text of trapped event.</td>
<td></td>
<td>WTO</td>
<td>X</td>
<td>COM</td>
<td></td>
</tr>
<tr>
<td>AOTIME</td>
<td>No Int</td>
<td>Time match occurred (hh:mm:ss). (Not the current system time.)</td>
<td></td>
<td>TOD</td>
<td>WTO</td>
<td>X</td>
<td>COM</td>
</tr>
<tr>
<td>AOTRAP</td>
<td>No Char</td>
<td>Trap name for current match.</td>
<td></td>
<td>TOD</td>
<td>WTO</td>
<td>X</td>
<td>COM</td>
</tr>
<tr>
<td>AOTTYPE</td>
<td>No Char</td>
<td>The type of trap that caused the current match. (CMD, TOD, WTO, XOC, XO1, XOM, XO2, COM)</td>
<td></td>
<td>TOD</td>
<td>WTO</td>
<td>X</td>
<td>COM</td>
</tr>
<tr>
<td>AUERID</td>
<td>No Char</td>
<td>The RACF userid associated with the command or WTO.</td>
<td></td>
<td>WTO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOUSR</td>
<td>Yes Char</td>
<td>By default, the first 4 characters of the TSO, STC, or JOB name. The field can be overwritten when you generate a command or WTO using AOSIM.</td>
<td></td>
<td>WTO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOVERS</td>
<td>No Int</td>
<td>Version flag.</td>
<td></td>
<td>WTO</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

54  **AF/OPERATOR Command Reference Manual Version 340**
### Table 3. AF/OPERATOR-Supplied Match Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Input</th>
<th>Binary Equivalent</th>
<th>Binary Digit Positions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOVUTIMO</td>
<td>Yes</td>
<td>Int</td>
<td></td>
<td>Variable that controls whether a VUOPEN will time out and allows you to set the timeout limit.</td>
</tr>
<tr>
<td>AOWTONO</td>
<td>No</td>
<td>Int</td>
<td></td>
<td>Unique message number of the WTO being DOM’d.</td>
</tr>
<tr>
<td>AOXANSID</td>
<td>No</td>
<td>Char</td>
<td></td>
<td>APPL name (applname) specified during a successful LOGON command. This is the name by which AF/OPERATOR identifies the VTAM session.</td>
</tr>
<tr>
<td>CP</td>
<td>No</td>
<td>Char</td>
<td></td>
<td>The AF/OPERATOR command being executed.</td>
</tr>
</tbody>
</table>

**Notes:**

1. In this table, CMD indicates a trap triggered by an operator command; TOD indicates a trap that takes action on a specific date at a specific time of day or a match that takes action at evenly spaced intervals; WTO indicates a trap that is triggered by write-to-operator messages; X indicates a trap triggered by OMEGAMON exception messages; and COM indicates a match triggered by information related to sending commands to, and receiving responses from, linked systems.

2. If no JES is running, the variable named A0JCLAS contains less detailed jobclass information than if JES is running.

3. You can use the AOAUTOTK variable to group WTOs in a logical fashion. For example, if the value of the AUTO parameter is AUTO(YES/TAPEMNT), then the value of AOAUTOTK is TAPEMNT, and you can group all tapemount messages together. However, AOAUTOTK is a match variable, not a trap option; therefore, its value can be used for grouping but not for selecting WTOs.

4. The following table provides an example of a route code for AOROUT and a descriptor code for AODESC. The 4-digit hex representation corresponds to 16 binary digits. Each binary digit signifies whether the specific route or descriptor code is selected (1) or not selected (0). The AOROUT global match variable is 16 bytes.

<table>
<thead>
<tr>
<th>Input</th>
<th>Binary Equivalent</th>
<th>Binary Digit Positions</th>
<th>Descriptor/Route Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROUT(C705)</td>
<td>1100011100000101</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DESC(8001)</td>
<td>1000000000000001</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Note:** The route code may contain up to 128 binary digit positions.
Wait Command Match Variables

The following global match variables are created when you execute a WAIT WTO() or WAIT CMD() command. The variables defined include the matched text and the WAIT trigger conditions. If the WTO messages trapped are an MLWTO, the count of the lines in the MLWTO are placed in the variable AOWTX# and the message lines are placed in a series of variables with a prefix of AOWTX concatenated with the message lines sequence number (for example, AOWTX1, AOWTX2, AOWTX3,...AOWTX999). These new variables are described in the following table.

Note: These variables are not user-assignable.

A WTO trap will only trap the first line of an MLWTO if you have specified MLWTO(FIRST) at startup. This is also true for a WTO trap generated by WAIT.

Table 4. Wait Command Match Variables

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Source field in $MATENT</th>
<th>Type</th>
<th>Length</th>
<th>Description</th>
<th>Available in this type of trap</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOWTTEXT</td>
<td>AOCMDTXT</td>
<td>CHAR</td>
<td>=&lt;128</td>
<td>Matched text</td>
<td>C - - -</td>
</tr>
<tr>
<td>AOWTTEXT</td>
<td>WTMSGTXT</td>
<td>CHAR</td>
<td>=&lt;128</td>
<td>Matched text</td>
<td>- - W -</td>
</tr>
<tr>
<td>AOWTX#</td>
<td>MLWTO CNT</td>
<td>DEC</td>
<td>=&lt;999</td>
<td>Count of MLWTO lines</td>
<td>- - W -</td>
</tr>
<tr>
<td>AOWTXN</td>
<td>TEXT LINE</td>
<td>CHAR</td>
<td>=&lt;70</td>
<td>MLWTO lines</td>
<td>- - W -</td>
</tr>
<tr>
<td>AOWTASID</td>
<td>AOMATSID</td>
<td>INT</td>
<td>2</td>
<td>ASID of issuer (AF/OPERATOR)</td>
<td>C - W -</td>
</tr>
<tr>
<td>AOWTCONM</td>
<td>AOCMDCNM</td>
<td>CHAR</td>
<td>8</td>
<td>Console name</td>
<td>C - W -</td>
</tr>
<tr>
<td>AOWTCONM</td>
<td>WTCONNM</td>
<td>CHAR</td>
<td>8</td>
<td>Console name</td>
<td>C - W -</td>
</tr>
<tr>
<td>AOWTDESC</td>
<td>WTDSECCD</td>
<td>HEX</td>
<td>4</td>
<td>Descriptor codes</td>
<td>- - W -</td>
</tr>
<tr>
<td>AOWTJCLS</td>
<td>AOMATJBC</td>
<td>CHAR</td>
<td>1</td>
<td>Job class of issuer (AF/OPERATOR)</td>
<td>C - W -</td>
</tr>
<tr>
<td>AOWTJNAM</td>
<td>AOMATJBN</td>
<td>CHAR</td>
<td>8</td>
<td>Job name, TSO ID, task name (AF/OPERATOR)</td>
<td>C - W -</td>
</tr>
<tr>
<td>AOWTJID</td>
<td>AOMATJID</td>
<td>CHAR</td>
<td>8</td>
<td>JES job ID (JOB/STC/TSU XXXX)</td>
<td>C - W -</td>
</tr>
<tr>
<td>AOWTROUT</td>
<td>AOMATXROU</td>
<td>BIN</td>
<td>16</td>
<td>(HEX) EBCDIC rout codes</td>
<td>- - W -</td>
</tr>
<tr>
<td>AOWTSYS</td>
<td>AOMATSYS</td>
<td>CHAR</td>
<td>8</td>
<td>System ID</td>
<td>C - W -</td>
</tr>
</tbody>
</table>
JES2 Interface Match Variables

AF/OPERATOR JES2 interface match variables contain a wealth of automation data, including:

- a more accurate representation of jobclass
- job message class
- job originating node
- job input device
- TSO ID that will be notified when the job completes
- job JES2 job identifier
- job account number
- job bin number

The JES2 variables available for each match are described in the table below.

Table 5. JES2 Interface Match Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>JES2 JCT field</th>
<th>Length in characters</th>
<th>Information contained in the variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOACCTN</td>
<td>JCTACCTN</td>
<td>4</td>
<td>Account number</td>
</tr>
<tr>
<td>AOINDEV</td>
<td>JCTINDEV</td>
<td>8</td>
<td>Input device of the job</td>
</tr>
<tr>
<td>AOJCLASS</td>
<td>JCTJCLAS</td>
<td>1</td>
<td>Jobclass data</td>
</tr>
<tr>
<td>AOJOBID</td>
<td>JCTJOBID</td>
<td>8</td>
<td>First 3 characters=jobtype, then JES2 job ID</td>
</tr>
<tr>
<td>AOMCLASS</td>
<td>JCTMCLAS</td>
<td>1</td>
<td>Message class</td>
</tr>
<tr>
<td>AONONDE</td>
<td>JCTNONDE</td>
<td>8</td>
<td>Originating node of the job</td>
</tr>
<tr>
<td>AOPNAME</td>
<td>JCTPNAME</td>
<td>20</td>
<td>Programmer’s name in JCL</td>
</tr>
<tr>
<td>AOROOMN</td>
<td>JCTROOMN</td>
<td>4</td>
<td>Room (output bin) number</td>
</tr>
<tr>
<td>AOTSUID</td>
<td>JCTTSUID</td>
<td>7</td>
<td>TSO user ID (NOTIFY= in JCL)</td>
</tr>
</tbody>
</table>
**AOACCTN**  
The AOACCTN variable contains the contents of the JCTACCTN field in the JES2 JCT. It is 4 characters in length.

**AOINDEV**  
The AOINDEV variable contains the contents of the JCTINDEV field in the JES2 JCT. It is 8 characters in length. You can adjust security, priority, and job scheduling/performance based on the AOINDEV match variable.

**AOJCLASS**  
The AOJCLASS variable contains the contents of the JCTJCLAS field in the JES2 JCT. It is 1 character in length.

**AOJOBID**  
The AOJOBID variable has the contents of the JCTJOBID field in the JES2 JCT. It is 8 characters in length, in the form `tttnnnnn` where `ttt` is the type of job (STC, JOB, or TSU) and `nnnnn` is the 5-digit JES2 job identifier.

**AOMCLASS**  
The AOMCLASS variable contains the contents of the JCTMCLAS field in the JES2 JCT. It is 1 character in length. You can use the information for a priority based on message class algorithm when printing jobs. You can also use the variable to restrict certain message classes (for example, forms marked “confidential”).

*Note:* A similar variable, AOJCLS, is an AF/OPERATOR match variable.

**AONONDE**  
The AONONDE variable contains the contents of the JCTNONDE field in the JES2 JCT. It is 8 characters in length. The AONONDE variable provides an important piece of automation data. For example, the system on which the job originated can be important for security (don’t allow programmer “X” to run on this machine), job scheduling (don’t allow jobclass t from node A until after 9:00), and priority (jobs from node CACOWLAB are more important than jobs from CACOWLAF).

**AOPNAME**  
The AOPNAME variable contains the contents of the JCTPNAME field in the JES2 JCT. It is 20 characters in length and derived from the user-specified programmer name in the job’s JCL. You can use it to provide conditional logic based on the programmer’s name, or to get the programmer’s name for reports.

**AOROOMN**  
The AOROOMN variable contains the contents of the JCTROOMN field in the JES2 JCT. It is 4 characters in length.

**AOTSID**  
The AOTSID variable contains the contents of the JCTTSUID field in the JES2 JCT. It is 7 characters in length. AOTSID contains the value of the NOTIFY= parameter in users JCL. The variable can be useful in sending job related messages back to the TSO user who will be notified when the job completes.
JES3 Interface Match Variables

AF/OPERATOR JES3 interface match variables contain a wealth of automation data including:

- a more accurate representation of jobclass
- job message class
- job originating node
- JES3 job identifier
- job account number
- JES3 GLOBAL system name
- JES3 jobclass group name
- programmer name from JCL

The JES3 variables available for each match are described below.

Table 6. JES3 Interface Match Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>JES3 source field</th>
<th>Length in characters</th>
<th>Information contained in the variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOACCNT</td>
<td>ACTACCNT</td>
<td>7</td>
<td>Account number</td>
</tr>
<tr>
<td>AOJCLASS</td>
<td>JMRCLASS</td>
<td>1</td>
<td>Jobclass data</td>
</tr>
<tr>
<td>AOJOBID</td>
<td>MEMJBID</td>
<td>8</td>
<td>First 3 characters=jobtype, then JES3 job ID</td>
</tr>
<tr>
<td>AOJ3GLBL</td>
<td>MPNAME</td>
<td>8</td>
<td>JES3 GLOBAL system name</td>
</tr>
<tr>
<td>AOJ3GRP</td>
<td>MEMINIT</td>
<td>8</td>
<td>JES3 jobclass group name</td>
</tr>
<tr>
<td>AOMCLASS</td>
<td>JCTJMGPO</td>
<td>1</td>
<td>Message class</td>
</tr>
<tr>
<td>AONONDE</td>
<td>MEMNODE</td>
<td>8</td>
<td>Originating node of the job</td>
</tr>
<tr>
<td>AOPNAME</td>
<td>ACTPRGNM</td>
<td>20</td>
<td>Programmer’s name in JCL</td>
</tr>
</tbody>
</table>
The line variables contain text of responses to operator commands or of trapped messages and exceptions.

At various times, AF/OPERATOR receives information that needs to be divided into usable units. For example, it may trap text from a WTO that contains keywords or numbers that AF/OPERATOR must read and respond to. The WTO may consist of more than one line. When AF/OPERATOR sends commands to OMEGAMON, the information it receives might fill up a whole screen.

AF/OPERATOR uses two facilities, LINE and AOTXT, for isolating pieces of text from a line or screen and assigning them to a variable.

(AF/OPERATOR also provides a mechanism for dividing a line of text into units that are read into variables. These are pattern variables. See “Pattern Variables” on page 88.)
The following table lists the line variables.

Table 7. AF/OPERATOR Line Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Can user assign?</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOCPL001 etc.</td>
<td>No</td>
<td>Char</td>
<td>Text of a line returned by a CP command.</td>
</tr>
<tr>
<td>AOCPL#</td>
<td>No</td>
<td>Int</td>
<td>Number of lines returned by a CP command.</td>
</tr>
<tr>
<td>AOLINPFX</td>
<td>Yes</td>
<td>Char</td>
<td>User-defined prefix to replace LINE.</td>
</tr>
<tr>
<td>AOMPROD</td>
<td>No</td>
<td>Char</td>
<td>Product name.</td>
</tr>
<tr>
<td>AOMPROF</td>
<td>No</td>
<td>Char</td>
<td>Profile member suffix.</td>
</tr>
<tr>
<td>AOMSYSID</td>
<td>No</td>
<td>Char</td>
<td>Profile member suffix.</td>
</tr>
<tr>
<td>AOMVERS</td>
<td>No</td>
<td>Char</td>
<td>Product version number.</td>
</tr>
<tr>
<td>AOTXT1 etc.</td>
<td>No</td>
<td>Char</td>
<td>Text of a line in a multi-line trapped message.</td>
</tr>
<tr>
<td>AOTXT#</td>
<td>No</td>
<td>Int</td>
<td>Number of lines in a multi-line trapped message.</td>
</tr>
<tr>
<td>LINE1 etc.</td>
<td>No</td>
<td>Char</td>
<td>Text of a line in a multi-line screen display.</td>
</tr>
<tr>
<td>LINE#</td>
<td>No</td>
<td>Int</td>
<td>Number of lines in a multi-line screen display. Note that if LINE# or its equivalent contains the value 999, it is likely that you issued an MVS command that returned more than 1000 lines of data. You may need to issue more specific commands to limit the amount of output.</td>
</tr>
</tbody>
</table>

The LINE and AOTXT facilities are especially important. The LINE facility is used for all multi-line information for AF/OPERATOR, except for text that triggers a trap. The AOTXT facility is used for trapped text. Multi-line information for the LINE facility comes to AF/OPERATOR from several sources:

- traps
- the text of a multi-line WTO
- responses to OMEGAMON commands, such as TSOJ
- responses to commands that AF/OPERATOR sends to MVS or some other system or subsystem
- responses to a VTAM application accessed through the Programmerless Open VTAM Interface

Consider, for example, responses to OMEGAMON commands.

When you operate OMEGAMON manually, the result of most commands is a multi-line display of text (including letters, numbers, and symbols). When AF/OPERATOR issues a command to OMEGAMON requesting system information, the result is a set of variables. Each variable contains a line of text that would have been displayed in manual operation.
For example, manually issuing the following commands:

```
#TSOJ
TSOJ
CPU
```
results in a display like the following:

```
#TSOJ           12
 TSOJ USER28A USER27 USER10 USER28B USER05A USER16 USER12
 CPU  5 SEC 30 SEC 25 SEC 15 SEC 1 SEC 20 SEC 15 SEC
```

If AF/OPERATOR executes a REXX exec containing the same commands:

```
ADDRESS OMMVS
' #TSOJ'
' TSOJ'
' CPU'
```

The result is a set of variables like the following:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINE 1</td>
<td>Line 1 of the data display</td>
</tr>
<tr>
<td>LINE 2</td>
<td>Line 2 of the data display</td>
</tr>
<tr>
<td>LINE 3</td>
<td>Line 3 of the data display</td>
</tr>
</tbody>
</table>

AF/OPERATOR always omits the top line (the INFO-line) of an OMEGAMON screen.

The value of LINE1 is:

```
#TSOJ           12
```

The value of LINE2 is:

```
TSOJ USER28A USER27 USER10 USER28B USER05A USER16 USER12
```

The value of LINE3 is:

```
CPU  5 SEC 30 SEC 25 SEC 15 SEC 1 SEC 20 SEC 15 SEC
```
Structure of LINE Variables

Each variable returned consists of two parts:

- A 1- to 5-character alphabetic prefix, such as OM. The default prefix is LINE.
- A 1- to 3-digit numeric suffix, such as 12.

You can specify the prefix by declaring AOLINPF as a global variable initialized to the prefix of your choice.

For example, if a command contains:

```markdown
AOLINPF = OM
GLBVPUT ('AOLINPF')
```

then a subsequent OMEGAMON command will return the following set of variables:

- OM1
- OM2
- OM3

**Line-Count Variable:** When AF/OPERATOR returns a set of LINE variables representing the lines in a screen display, it also returns a variable containing the number of lines in the display.

The name of that variable consists of two parts:

- The same alphabetic prefix used for the LINE variables, if you have defined a prefix, such as OM. However, if AF/OPERATOR uses the default prefix LINE, it uses the prefix LINE for the line-count variable.
- The suffix # (a pound sign).

For example, if AOLINPF is OM, the line-count variable is OM#. If an OMEGAMON command returns 15 line variables, the value of OM# is 15.

Since 999 lines of data is the capacity of the line-variable facility, if LINE# contains the value 999 it is likely that some data was missed. If this occurs, the variables LINE1 through LINE999 contain the first 999 lines of returned data, and more specific commands to limit the amount of output should be issued.

AF/OPERATOR determines the end of a display as follows. If it encounters more than 255 blanks, starting in column 1 of line \( n \), it regards line \((n-1)\) as the last line of the display, and it sets the line-count variable to \((n-1)\).

**AOTXT Variables:** Any text that triggers an AF/OPERATOR trap is stored by the AOTXT facility. This may be the text of an operator command, a WTO/WTOR, or an OMEGAMON exception.

The AF/OPERATOR-supplied match variable AOTEXT stores one line of text.

The text of MLWTOs (multi-line WTOs) is stored in the AF/OPERATOR-supplied line variables AOTXT1 through AOTXT999. Each AOTXTn variable stores one line of text, similar to the LINE variables discussed previously. Each AOTXT variable consists of:

- The prefix AOTXT.
A 1- to 3-digit numeric suffix.

The line-count variable is AOTXT#. This is the number of AOTXT variables returned by AF/OPERATOR.

Since 999 lines of data is the capacity of the line-variable facility, if AOTXT# contains the value 999 it is likely that some data was missed. If this occurs, the variables AOTXT1 through AOTXT999 contain the first 999 lines of returned data and more specific commands to limit the amount of output should be issued.

You cannot specify a prefix to replace AOTXT.

The AOOMWAIT Variable

Several OMEGAMON commands begin a process that accumulates data over a small period of time. For example, the OMEGAMON for MVS command MCPU05 accumulates data on CPU status during an interval set in the OMEGAMON profile, and then displays this data.

When the command is first issued, OMEGAMON responds:

```
MCPU05 >> CPU Measurement in Initialization <<
```

To cause a delay in the preset OMEGAMON interval, you can declare the global variable AOOMWAIT and use it to set the length of time to wait for OMEGAMON data before returning the display variables.

This sequence from a REXX exec sets the interval to 2 seconds and issues the MCPU command to OMEGAMON:

```
/* REXX */
aoomwait=2
x=GLBVPUT("AOOMWAIT")
ADDRESS AFHOST
   "AFADDR OMMVS"
   "' MCPU05'"
   "AFADDR"
   "SHOW VARS(LINE*)"
```
Introduction

This chapter details how to create and save REXX shared variables, which can be used by all execs. The shared variables exist within a shared variable pool. There are two types of shared variable pools. The SYSTEM pool is always available. The SYSPLEX pool is an optional pool and is available to systems across a Sysplex that have specified the same XES structure name on the CACHE startup parameter.

Chapter Contents

Creating Rexx Shared Variables .................................................. 66
Additional Information about the SYSPLEX RSV Pool ....................... 67
Requirements ................................................................. 68
Functions for Shared Variables ................................................... 69
SHARVGET ........................................................................... 72
SHARVUPD ........................................................................... 74
SHARVDEL ........................................................................... 77
Stem Variables and Compound Variable Groups ................................ 79
IRXEXCOM Return Codes ........................................................ 82
Creating Rexx Shared Variables

AF/OPERATOR includes 4 functions that allow you to create the **REXX Shared Variables** (RSVs) that are contained in a shared variable pool. RSVs are unique because a shared variable’s value can be used by all execs and is preserved after the exec that created it terminates. You can even specify that the shared variable’s value be preserved after the AF/OPERATOR started task terminates.

The following table shows when to use each of the four functions.

<table>
<thead>
<tr>
<th>IF you want to...</th>
<th>THEN use the...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy the current value of a local REXX variable to a shared variable. (If the shared variable does not yet exist, AF/OPERATOR creates it.)</td>
<td>SHARVPUT function</td>
</tr>
<tr>
<td>Retrieve the current value of a shared variable and copy it to a local REXX variable.</td>
<td>SHARVGET function</td>
</tr>
<tr>
<td>Copy the current value of a REXX variable to a shared variable if the shared variable value matches a value specified in the function call.</td>
<td>SHARVUPD function</td>
</tr>
<tr>
<td>Delete the shared variable from the shared variable pool.</td>
<td>SHARVDEL function</td>
</tr>
</tbody>
</table>

**Note:**
- Except where explicitly noted, a shared variable has the same features and limitations as any other REXX variable.
  
  For example, RSVs are not limited to 8-character variable names as are AF/OPERATOR command language variables. Also, RSVs do not have a 240 character maximum length. Variables in the SYSTEM pool are only restricted by the amount of virtual storage available. Variables in the SYSPLEX pool are limited by the MAXSIZE specification on the CACHE startup parameter. For more information see “Startup Parameters” in the AF/OPERATOR Configuration and Customization Guide.

- A shared variable can only be created from a REXX exec.

- Shared variables in the SYSTEM pool are available to all REXX execs executing within a single AF/OPERATOR address space. Shared variables in the SYSPLEX pool are available to all REXX execs executing within any AF/OPERATOR address space sharing the same XES cache structure.
Additional Information about the SYSPLEX RSV Pool

- The SYSPLEX REXX shared variable (RSV) pool is enabled by adding the CACHE startup parameter to the startup member in RKANPAR.
- Variables in the SYSPLEX RSV pool are stored in an XES cache structure defined within the active CFRM policy of the Sysplex coupling facility.
- Multiple AF/OPERATOR systems across a Sysplex can share data within the SYSPLEX RSV pool if each of them specifies the same XES structure name on the CACHE startup parameter.
- The structure is persistent, so that data is retained even when there are no active AF/OPERATOR systems connected to it.
Requirements

The RSV feature is invoked automatically during AF/OPERATOR initialization, so you don't need to turn anything on to create and use shared variables.

If you want to checkpoint your shared variable values so that they will be preserved across stops and starts of the AF/OPERATOR started task, you may need to make two small changes to your AF/OPERATOR startup parameters. (See the AF/OPERATOR Configuration and Customization Guide for general information about checkpointing variables.)

Follow the procedure below to set up checkpointing for your shared variable values. Note that checkpointing is only available for variables in the SYSTEM pool.

1. Access the AMS Customizer as described in AF/OPERATOR Configuration and Customization Guide and go to the Primary Customization Menu.

2. Select Define Startup Parameters.

3. On the first panel of parameters, set the CKPT parameter to Y. This will activate the checkpointing facility when AF/OPERATOR initializes.

   **Note:** Variables from the SYSPLEX pool cannot be written to the AF/OPERATOR checkpoint dataset.

4. Press Enter to display the second panel. Find the column of RELOAD subparameters.

5. Indicate that you want to reload RSV values from the checkpoint dataset at AF/OPERATOR initialization by:
   - specifying ALL (reload all data from the checkpoint dataset) in the TYPE field.
   - or including RSVS in the TYPE field. (You can specify more than one data type in this field.)

6. Press Enter to regenerate the AF/OPERATOR startup parameter member. Review the generated code when it appears, and press F3 to exit the member.

   **Note:** If you expect to checkpoint more than a few shared variables, you should recalculate the space required for the VSAM checkpoint dataset. See the AF/OPERATOR Configuration and Customization Guide for details.
Functions for Shared Variables

This section lists functions that allow you to:

- copy a local REXX variable value to a shared variable in a system pool
- retrieve the value of a shared variable and copy it to a local REXX variable
- verify a shared variable equals a specified value and copy it to a local REXX variable
- delete a shared variable from a shared variable pool.

SHARVPUT

The SHARVPUT function copies the current value of a local REXX variable to a shared variable in the system pool. If the corresponding shared variable does not already exist, AF/OPERATOR creates one with the same name as the local REXX variable.

In the following figure, the value before the function is performed is:

value_1='Bernice'

The function that copies the value to the shared variable is:

/* REXX */
ret=SHARVPUT('SYSTEM',value_1)

FIGURE 5. SHARVPUT Function
Functions for Shared Variables

The SHARVPUT function returns one of the following codes upon completion.

- **pool_name** May be either ‘SYSTEM’ or ‘SYSPLEX’.
- **variable_name** The 1- to 250-byte name of a local REXX variable.
- **CKPT** Writes a checkpoint record for variable_name with the updated value.
- **NOCKPT** Default. No checkpoint record is written with this value.

**Important**
AF/OPERATOR does not write a checkpoint record for the new value of a shared variable unless you specify the CKPT operand. At restart, AF/OPERATOR reloads a shared variable value as recorded in the checkpoint dataset - even if that value was recorded prior to several thousand changes.

Variables from the SYSPLEX pool cannot be written to the AF/OPERATOR checkpoint. In this case, if the CKPT option is specified, it is ignored. It is not necessary to checkpoint the data contained in the SYSPLEX RSV pool since the structure is persistent and the checkpoint is associated with a single AF/OPERATOR system.

**Return Codes**
The SHARVPUT function returns one of the following codes upon completion.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Completed successfully.</td>
</tr>
<tr>
<td>8</td>
<td>The variable_name specified is not a local REXX variable.</td>
</tr>
<tr>
<td>16</td>
<td>A parameter is missing or misspelled.</td>
</tr>
<tr>
<td>20</td>
<td>The variable’s value was truncated.</td>
</tr>
<tr>
<td>24</td>
<td>Structure full. The variable cannot be set.</td>
</tr>
<tr>
<td>32</td>
<td>The function was executed in a non-REXX environment.</td>
</tr>
<tr>
<td>64</td>
<td>No RSV pool exists.</td>
</tr>
<tr>
<td>68</td>
<td>Internal error.</td>
</tr>
<tr>
<td>72</td>
<td>Internal error; ENQ error.</td>
</tr>
<tr>
<td>76</td>
<td>Internal; XES data sharing error. Accompanied by message !AOP3799.</td>
</tr>
</tbody>
</table>

**Note:** If SHARVPUT returns a code other than those listed above, see “IRXEXCOM Return Codes” on page 82.
Examples

The following shows the use of SHARVPUT in a REXX program.

/* REXX */
COMP_CODE = aofunction(args)
RET_CODE = SHARVPUT('SYSTEM','COMP_CODE','CKPT')
IF RET_CODE <> 0 THEN DO
   CALL WTO 'UNABLE TO SAVE COMP_CODE'
END

Comments

1. On execution, AF/OPERATOR copies the value of local REXX variable COMP_CODE to a shared variable of the same name.
   If the shared variable does not yet exist, AF/OPERATOR creates it.

2. A checkpoint record is written with the updated value of COMP_CODE.
SHARVGET

The SHARVGET function retrieves the current value of a shared variable and copies it to a local REXX variable.

In the following figure, the function that retrieves and copies the value of the variable is:

```/* REXX */
ret=SHARVGET('SYSTEM','value_2')
say 'THE VALUE OF VALUE_2 IS' VALUE_2
```

**FIGURE 6. SHARVGET Function**

Syntax

```>>>---SHARVGET(pool_name,variable_name)-------->>><
```

- **pool_name** May be either ‘SYSTEM’ or ‘SYSPLEX’.
- **variable_name** The 1- to 250-byte name of a shared variable.
Return Codes

The SHARVGET function returns one of the following codes upon completion.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Completed successfully.</td>
</tr>
<tr>
<td>8</td>
<td>The specified variable_name does not exist in the shared variable pool.</td>
</tr>
<tr>
<td>16</td>
<td>A parameter is missing or misspelled.</td>
</tr>
<tr>
<td>20</td>
<td>The variable’s value was truncated.</td>
</tr>
<tr>
<td>24</td>
<td>Structure full. The variable cannot be set.</td>
</tr>
<tr>
<td>32</td>
<td>The function was executed in a non-REXX environment.</td>
</tr>
<tr>
<td>64</td>
<td>No RSV pool exists.</td>
</tr>
<tr>
<td>68</td>
<td>Internal error.</td>
</tr>
<tr>
<td>72</td>
<td>Internal error; ENQ error.</td>
</tr>
<tr>
<td>76</td>
<td>Internal; XES data sharing error. Accompanied by message !AOP3799.</td>
</tr>
</tbody>
</table>

Note: If SHARVGET returns a code other than those listed above, see “IRXEXCOM Return Codes” on page 82.

Examples

The following shows the use of SHARVGET in a REXX program.

```rexx
/* REXX */
RET_CODE = SHARVGET('SYSTEM','START_OPTION')
IF RET_CODE = 0 THEN DO
   IF START_OPTION = 'C'
      END.
ELSE DO ...;

On execution, AF/OPERATOR copies the value of the shared variable START_OPTION to a local variable of the same name. If the local REXX variable does not yet exist, AF/OPERATOR creates it.
SHARVUPD

The SHARVUPD function allows you to verify that a shared variable equals a specified value before copying the current value of the corresponding local REXX variable.

FIGURE 7. SHARVUPD Function

Syntax

```
>>---SHARVUPD(pool_name,variable_name,variable_value-----------------------)---><
|                |
|--- ,CKPT ------|
|                |
+--- ,NOCKPT---+
```

- **pool_name** May be either ‘SYSTEM’ or ‘SYSPLEX’.
- **variable_name** The 1- to 250-byte name of a local REXX variable.
- **variable_value** The 1K- to 32K-byte value to compare to the current value of the corresponding shared variable.
- **CKPT** Writes a checkpoint record for `variable_name` with the updated value.
- **NOCKPT** Default. No checkpoint record is written with this value.
Important
AF/OPERATOR does not write a checkpoint record for the new value of a shared variable unless you specify the CKPT operand. At restart, AF/OPERATOR reloads a shared variable value as recorded in the checkpoint dataset - even if that value was recorded prior to several thousand changes.

Variables from the SYSPLEX pool cannot be written to the AF/OPERATOR checkpoint. In this case, if the CKPT option is specified, it is ignored. It is not necessary to checkpoint the data contained in the SYSPLEX RSV pool since the structure is persistent and the checkpoint is associated with a single AF/OPERATOR system.

Return Codes
The SHARVUPD function returns one of the following codes upon completion.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Completed successfully.</td>
</tr>
<tr>
<td>4</td>
<td>The specified variable_value was not equal to the value of shared variable variable_name.</td>
</tr>
<tr>
<td>8</td>
<td>The specified variable_name is not a shared variable.</td>
</tr>
<tr>
<td>16</td>
<td>A parameter is missing or misspelled.</td>
</tr>
<tr>
<td>20</td>
<td>The variable’s value was truncated.</td>
</tr>
<tr>
<td>24</td>
<td>Structure full. The variable cannot be set.</td>
</tr>
<tr>
<td>32</td>
<td>The function was executed in a non-REXX environment.</td>
</tr>
<tr>
<td>64</td>
<td>No RSV pool exists.</td>
</tr>
<tr>
<td>68</td>
<td>Internal error.</td>
</tr>
<tr>
<td>72</td>
<td>Internal error; ENQ error.</td>
</tr>
<tr>
<td>76</td>
<td>Internal; XES data sharring error. Accompanied by message !AOP3799.</td>
</tr>
</tbody>
</table>

Note: If SHARVUPD returns a code other than those listed above, see “IRXEXCOM Return Codes” on page 82.

Example
The following shows the use of SHARVUPD in a REXX program.

```rexx
/* REXX */
COMP_CODE = aofunction(args)
RET_CODE = SHARVUPD('SYSTEM','COMP_CODE','4','CKPT')
```
Comments
1. On execution, AF/OPERATOR compares the value of shared variable COMP_CODE to the value 4.
2. If the two values are equal, AF/OPERATOR updates the value of the shared value with the current value of local REXX variable COMP_CODE.
3. A checkpoint record is written with the updated value.
SHARVDEL

When you use the SHARVDEL function, AF/OPERATOR deletes the shared variable from the shared variable pool. If a checkpoint record exists for the shared variable, AF/OPERATOR deletes that as well.

In the following figure, the function that deletes the variable is:

```
ret=SHARVDEL('SYSTEM','value_2')
```

FIGURE 8. SHARVDEL Function

Syntax

```
SHARVDEL(pool_name,variable_name)
```

---

pool_name May be either ‘SYSTEM’ or ‘SYSPLEX’.
variable_name The 1- to 250-byte name of a shared variable.
Return Codes
The SHARVDEL function returns one of the following codes upon completion.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Completed successfully.</td>
</tr>
<tr>
<td>8</td>
<td>The specified variable_name is not a shared variable.</td>
</tr>
<tr>
<td>16</td>
<td>A parameter is missing or misspelled.</td>
</tr>
<tr>
<td>24</td>
<td>Structure full. The variable cannot be set.</td>
</tr>
<tr>
<td>32</td>
<td>The function was executed in a non-REXX environment.</td>
</tr>
<tr>
<td>64</td>
<td>No RSV pool exists.</td>
</tr>
<tr>
<td>68</td>
<td>Internal error.</td>
</tr>
<tr>
<td>72</td>
<td>Internal error; ENQ error.</td>
</tr>
<tr>
<td>76</td>
<td>Internal; XES data sharing error. Accompanied by message !AOP3799.</td>
</tr>
</tbody>
</table>

Note: If SHARVDEL returns a code other than those listed above, see “IRXEXCOM Return Codes” on page 82.

Examples
The following shows the use of SHARVDEL in a REXX program.

```rexx
/* REXX */
COMP_CODE = aofunction(args)
RET_CODE = SHARVGET('SYSTEM','COMP_CODE')
IF RET_CODE = 0 THEN DO
  IF COMP_CODE = 0 THEN DO
    RET_CODE = SHARVDEL('SYSTEM','COMP_CODE')
  .
  END
END
```

Comments
1. On execution, AF/OPERATOR copies the value of shared variable COMP_CODE to a local REXX variable of the same name (using the line RET_CODE = SHARVGET('SYSTEM','COMP_CODE')).
   If the variable does not yet exist, AF/OPERATOR creates it (using the line RET_CODE = SHARVGET('SYSTEM','COMP_CODE')).
2. Next, the shared variable name COMP_CODE is deleted from the system pool (using the line RET_CODE = SHARVDEL('SYSTEM','COMP_CODE')).
3. If a checkpoint record exists for this variable, it is also deleted (also using the line RET_CODE = SHARVDEL('SYSTEM','COMP_CODE')).
**Stem Variables and Compound Variable Groups**

You can use both stem variables and compound variables as shared variables. However, these two types of variables function differently.

**Using Stem Variables**

Stem variables, those symbols up to and including the first period (such as Zenia. or Z.), function the same way in AF/OPERATOR as they do in REXX. That is, when you specify a stem variable, it will replace all other stem variables with that common name.

**Using Compound Variables Groups**

However, compound variable groups (CVGs), which are groups containing variables beginning with the same root, (such as Zenia.Atwood or Zenia.T.Q), function differently. When you specify a new stem variable, such as Zenia. and you are working only with compound variables, those variables containing Zenia. are not affected. (If you were working with stem variables, all of the stem variables containing Zenia. would change to the new value you set for Zenia.)

So if you want to work with groups of similarly-named variables, you can use compound variable groups so you won’t have to specify down to the stem level.

For example, you might have variables named STCS.system.name where system can be SYSA, SYSB, or SYSC while name can be any of numerous values. You can increase the efficiency of your REXX exec by specifying SHARVGET(‘SYSTEM’, ‘STCS.SYSA.’) and then put the variables back into the shared pool using SHARVPUT(‘SYSTEM’, ‘STCS.SYSA.’).
Examples
The following example shows how stem variables and compound variables are processed when the SHARVPUT function is used.

Using a compound variable

A.0 = 0
A.1 = 1
A.2 = 2

A.B.1 = A
A.B.2 = B

After you execute the function

rc = SHARVPUT('SYSTEM'; 'A.B.')</n
the shared variable pool contains the following variables and values:

A.B.1 = A
A.B.2 = B

The SHARVPUT function takes all CVGs with A.B. from the local variable pool and copies them to the shared variable pool.

Using a stem variable

Following from the previous example, if you now execute the function

rc = SHARVPUT('SYSTEM'; 'A.')

the shared variable pool contains the following variables and values:

A.1 = 1
A.2 = 2
A.B.1 = A
A.B.2 = B

The SHARVPUT function takes all variables from the local variable pool with a value assigned starting with the stem A. and copies them to the shared variable pool. In this case, it means that A.1, A.2, A.B.1, and A.B.2 are copied.

Stem variables used as shared variables function the same way as stem variables do in REXX, except when you are using compound variable groups (CVGs). In CVGs, when you specify a stem variable, it does not replace all other variables with a common name.
Continuing with the variables from the previous example, if you now execute

\[
A.=0 \\
A.1=3 \\
A.2=4
\]

Setting the stem variable \(A\) resets all variables in the local variable pool with the stem \(A\), before \(A.1\) and \(A.2\) are given new values.

If you now execute the function using the same stem specification

\[
rc=\text{SHARVPUT(‘SYSTEM’,‘A.’)}
\]

the shared variable contains the following variables and values:

\[
A.1=3 \\
A.2=4 \\
A.B.1=A \\
A.B.2=B
\]

The SHARVPUT function takes all variables from the local variable pool with a value assigned starting with the stem \(A\). and copies them to the shared variable pool. In this case, it is only \(A.1\) and \(A.2\) that are copied. The values of \(A.B.1\) and \(A.B.2\) are unaffected by the SHARVPUT on the stem and keep their existing values.

If you use the SHARVGET function with a stem, the values for all of the variables with that stem will be returned.
IRXEXCOM Return Codes

Although unlikely, the RSV functions may also return codes from IBM’s Variable Access Routine, IRXEXCOM. If you encounter one of the following return codes and are sure that your syntax is correct, contact Candle Support Services.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>1xx</td>
<td>Where xx is a non-zero return code after a FETCH operation.</td>
</tr>
<tr>
<td>2xx</td>
<td>Where xx is a non-zero return code after a STORE operation.</td>
</tr>
<tr>
<td>3xx</td>
<td>Where xx is a non-zero return code after a NEXT operation.</td>
</tr>
<tr>
<td>4xx</td>
<td>Where xx is a non-zero return code after a DROP operation.</td>
</tr>
</tbody>
</table>
Introduction

Pattern variables are generic text strings created by combining explicit text with wildcard characters, such as question marks (?) and asterisks (*). Pattern matching saves time and reduces the number of traps required for automation because it lets you enter special characters that represent all possible variations on a string so that you don’t have to type the actual text characters.

Chapter Contents

Overview of Pattern Matching .......................................................... 84
The Two Types of Patterns .............................................................. 85
Pattern Characters ................................................................. 86
Pattern Variables ................................................................. 88
Finding Assistance in Developing Patterns ...................................... 90
Name Patterns ................................................................. 91
Overview of Pattern Matching

Pattern matching is very useful for trapping or displaying all of a certain type of message, command, or exception. For example, if you wanted to suppress all messages that contain the character string IEC10 and end with one digit, you could use pattern matching to accomplish this. The pattern string

'IEC10#'

would match any character string that contains exactly 6 characters, begins with IEC10, and ends with a single digit (IEC100 or IEC109 but not IEC10A).

When comparing patterns, AF/OPERATOR is case sensitive. For example, if your target string contains all uppercase letters, the text string you create to match it must contain all uppercase letters. Case sensitivity is especially important for OMEGAMON and to a lesser extent for WTOs. Commands, however, are typically not case sensitive because AF/OPERATOR converts them to uppercase letters.

This chapter discusses the types of patterns, pattern characters and variables, sources of assistance for pattern-making, and concludes with name patterns. After reading this chapter you will use pattern matching to:

- take full advantage of generalizing trap patterns
- optimize the number of active traps
The Two Types of Patterns

AF/OPERATOR matches two types of patterns:

**Full pattern**
A character string pattern designed to match ordinary character strings, such as a WTO message or a command, and to create pattern variables. Full patterns are used in the following operands:
- CMD('full-pattern') in the TRAP ADD CMD, TRAP CHANGE CMD, and WAIT CMD commands.
- WTO('full-pattern') in the TRAP ADD WTO, TRAP CHANGE WTO, and WAIT WTO commands.
- xtype('full-pattern') in the TRAP ADD xtype and TRAP CHANGE xtype commands.

A full pattern consists of a combination of the following elements enclosed in single quotes:
- alphanumeric text strings
- pattern characters
- names of pattern variables
You can omit the quotes if the pattern string is a single element.

**Name pattern**
A short character string pattern designed to match names, such as trap or variable names. Various SHOW and TRAP commands, for example, use name patterns.

See “Name Patterns” on page 91. In the rest of this chapter, pattern refers to a full pattern.
AF/OPERATOR matches parts of a target string to certain characters in a pattern. These pattern characters are:

- **Question mark (?)** Matches any one alphanumeric character.
- **Pound sign (#)** Matches any one numeric digit.
- **Asterisk (*)** Matches any string.
  
  **Note:** This pattern character is extremely resource-intensive if used as the first character in a pattern. Use caution!

- **Single blank** Matches one or more contiguous blanks.
- **Multiple blanks** Match the same number of contiguous blanks in the target string as there are in the pattern.

In addition, the following pattern characters have special uses:

- **Escape (¬)** Indicates that the next character should be read as a literal character, not as a pattern.
- **Period (.)** Terminates one or more pattern characters.

You can use the question mark, pound sign, and asterisk pattern characters to create pattern variables. See “Pattern Variables” on page 88 for a description of how to do this.

**Note:** Unless you want to create a pattern variable, pattern matching characters in your pattern should always be immediately followed by a period, comma, or space. For example, ??ABC is a correct pattern to match the string xxABC, where xx can be any two characters.

### Asterisks in a Pattern

*Use caution when using the asterisk (*) as a wildcard character.* Because it can match any pattern, it can be extremely resource-intensive if used as the first character in a pattern. AF/OPERATOR recognizes the asterisk (*) as a wildcard character in full patterns under the following conditions: For example, in the trap defined as follows:

```
TRAP ADD(TRP1) WTO(’*.ABENDED’) ....
```

AF/OPERATOR traps only WTO messages whose text ends with the string ABENDED. In this case, the period indicates that the asterisk is a wildcard character for a text string (it does not create a pattern variable).

If the character following the asterisk is a period, blank, or comma, AF/OPERATOR uses the asterisk as a wildcard character. It does not define a pattern variable. (If you want to define a pattern variable, use a question mark (?), pound sign (#) or asterisk (*) immediately followed by a name. See “Pattern Variables” on page 88 for more information on pattern variables.) If an asterisk is the last character in a pattern, it matches any remaining text of any kind in the target string, including periods, commas, and blanks. For example, the pattern

```
ABC*
```
matches the target string

\texttt{ABCW.X,Y Z @#$}

since the asterisk matches

\texttt{W.X,Y Z @#$}

The pattern

\texttt{ABC*,*}

also matches the same target string, but this time the first asterisk only matches

\texttt{W.X}

Next, the comma in the pattern matches the comma in the target string, and finally the second asterisk matches the rest of the target string.

\texttt{Y Z @#$}

### Blanks in a Pattern

AF/OPERATOR treats blanks (represented by a `b`) in a full pattern in the following way:

- A single blank in a pattern matches any number of contiguous blanks in a text string.
- Two or more contiguous blanks in a pattern match the same number of contiguous blanks in a text string.

For example, the full pattern

\texttt{‘IEC210Ib214-’}

matches all the following strings:

\texttt{IEC210Ib214-}
\texttt{IEC210Ibb214-}
\texttt{IEC210Ibbbb214-}

However, the full pattern

\texttt{‘IEC210Ibb214-’}

matches only the string

\texttt{IEC210Ibb214-}

This treatment of blanks in full patterns helps avoid problems created by message manuals that do not clearly indicate the number of blanks separating operands.
Pattern Variables

During the process of pattern matching, AF/OPERATOR can create variables for you that show the results of pattern matching. These are called pattern variables and are global in scope and character in type. Specifically, pattern variables indicate which parts of the target string are actually matched by the pattern characters.

The asterisk (*), question mark (?), or pound sign (#) may be used to define a pattern variable.

Defining Pattern Variables

To define a pattern variable, follow these steps:

1. Write one of the following pattern character strings:
   - one or a string of \( n \) question marks (?) to match \( n \) alphanumeric characters in the text string
   - one or a string of \( n \) pound signs (#) to match \( n \) numeric digits in the text string
   - a single asterisk (*) to match a substring of characters whose length and type you cannot predict. Remember to use this character with caution.

2. Immediately following the pattern character string, write a variable name that consists of 1 to 24 alphanumeric or national characters (@,#,$) and starts with an alphabetic or national character.

3. End the variable name with a period, comma, or blank (unless it is the last item in the pattern, or is 8 characters long).

The character following the variable name determines the content and extent of the variable assignment. For example, if a comma (,) follows the variable name, then the variable contains the text string up to the next comma in the string.

For example, in the full pattern

'IEC210I 214-*RC,*TEXT'

the asterisks (*) adjacent to RC and TEXT define these strings as pattern variable names. The pattern matches any target string that matches these criteria:

- begins with IEC210I
- continues with one or more blanks
- continues with 214-
- contains a comma in the rest of the string

AF/OPERATOR assigns any text between the hyphen and the comma to RC, and assigns any text following the comma to TEXT.
The following are examples of other pattern variables:

?JOB.  ##MSGNUM,

When a full pattern containing a pattern variable matches the text of a message or command, AF/OPERATOR does the following:

- Triggers the trap.
- Creates a global character variable with a maximum length of 255 characters.
- Assigns to the variable the text matched by the pattern character. This implies that a match exists. AF/OPERATOR does not match a null string.

For more information about other AF/OPERATOR variables, see “Variables in the AF/OPERATOR Environment” on page 41.

### Using the Asterisk to Create Pattern Variables

For a pattern character to define a character string as a variable name, the pattern character must be adjacent to the character string. Otherwise, the character string is simply part of the pattern text that AF/OPERATOR matches. This is important because if the asterisk is read as part of the pattern text, the asterisk becomes a wildcard character. For example, in the trap defined as follows:

```
TRAP ADD(TRP1) WTO(*ABENDED) ....
```

AF/OPERATOR will trap every WTO message in the system, and store them one after another in the variable ABENDED. Trap-matching is more efficient if you code the following trap pattern:

```
TRAP ADD(TRP1) WTO(*.ABENDED) ....
```

AF/OPERATOR traps only WTO messages whose text ends with the string ABENDED. In this case, the period indicates that the asterisk is a wildcard character for a text string (it does not create a pattern variable).

**Important:** If you use the asterisk (*) to create a pattern variable, it can require extensive processing.
Finding Assistance in Developing Patterns

The following describes sources of assistance for creating patterns.

Developing Patterns from Software Documentation

You can use the messages and codes manual for your operating system or software product as a guide for message patterns. Message manuals usually show the variable fields within a message and the delimiters used between variable fields. For example, the message IEC210I is documented in the IBM manual, *OS/VS2 Message Library:* VS2 System Messages. The entry is as follows:

```
IEC210I 214-rc,mod,jjj,sss,ddn[-#],ddd,ser,dsn
```

This pattern for the above message

```
'IEC210I 214-*RC,*MOD,*JJJ,*SSS,*DDN,*DDD,*SER,*DSN'
```

traps the message and stores the variable information in the following AF/OPERATOR pattern variables:

```
RC, MOD, JJJ, SSS, DDN, DDD, SER, and DSN
```
Name Patterns

A name pattern is a short text pattern used for matching names. A name is a string of 1 to 24 alphanumeric or national characters, where the first character is alphabetic or national. For example, a name pattern consisting of the string S followed by an asterisk matches every target name starting with S. Often, however, a name pattern is the first part of a name—from 1 to 24 characters—followed by an asterisk, for example, TRAP111* or TRAP11*.

Using Wildcard Characters in Name Patterns

The asterisk (*), question mark (?), or pound sign (#) can be used as a wildcard character in up to 24-character name fields.

*Note:* To distinguish between # the national character and # the numeric wildcard, use the escape character (^) to indicate that the next character should be read as a literal character, not as a pattern character.

Wildcard characters can be used in the following commands to match trap, variable, or file names:

- DEL SYSVAR()
- TRAP ENABLE
- TRAP DELETE
- TRAP DISABLE
- SHOW FILES
- SHOW TRAPS
- SHOW VARS

Wildcard characters can be used only as the suffix in the SSCT, sysid, or newsysid fields within the TRAP and SHOW commands. For other cases, you can use wildcard characters anywhere within a name pattern.

*Note:* When you use an asterisk alone, the asterisk wildcard character will match everything. Use caution when using the asterisk alone; if possible, use the asterisk along with a portion of the name for a more efficient match.

Examples

Suppose each of your traps has a name of the form TRAPnnnn (where n is a numeric digit); for example, TRAP1113. Then

```
TRAP ENABLE(TRAP111*)
```

enables TRAP1110 through TRAP1119.

```
TRAP ENABLE(TRAP11*)
```

enables TRAP1100 through TRAP1199.

```
SHOW FILES(LOGON*)
```

might display LOGONO2, LOGONOC, LOGONOI, and LOGONOM.
In the following example, the pound sign (#) functions as a wildcard character that matches one numeric digit and the period indicates that pattern matching ends when the character that follows the period (in this case, X) is encountered.

```
TRAP DELETE(AB#.X)
```

deletes traps such as AB1X, AB2X, and AB5X.

To delete only the trap named AB#X, use the following command:

```
TRAP DELETE(AB^#X)
```

### A Caution in Using Name Patterns

Be careful using name patterns that are short and end in an asterisk or that consist only of an asterisk. They may match more patterns than you intend.

For example:

```
TRAP SHOW(*)
```

would show every trap in storage.

And in this example:

```
TRAP DELETE(*) FORCE
```

would delete every trap in storage and therefore remove your ability to control the product. Product control commands will no longer work if this trap statement is used. However, you can use the MVS modify command to issue commands to restore control.
Introduction

In addition to REXX functions provided by the IBM REXX language, Candle supplies a set of REXX functions for your use. Remember, though, these Candle-supplied REXX functions are usable only by REXX. The types of Candle-supplied REXX functions are listed in the Chapter Contents below.

Unless otherwise indicated, all function input and output is in character format. This chapter refers to global and system variables and pattern matching. For detailed information about these concepts, refer to “Variables in the AF/OPERATOR Environment” on page 41 and “Pattern Matching” on page 83.

If a Candle-supplied REXX function fails for any reason (such as invalid parameters) REXX issues message IRX0040I and raises the syntax condition.

Chapter Contents

Date/Time Functions ................................................................. 94
IMSCMD Function ................................................................. 95
NetView Interface Functions ..................................................... 106
OMEGAVIEW Interface Functions ........................................... 109
POVI Functions ................................................................. 121
REDIRECT Function ............................................................ 136
System Functions ................................................................. 141
Variable Functions ............................................................. 165
Communication Functions ..................................................... 170
Date/Time Functions

Dates specified with 2-digit years (for example, MM/DD/YY) will still be supported into the 21st century. Therefore, any specifications containing 2-digit year dates should still function properly until the year 2075 due to the following assumptions:

- assume 20th century (1975-1999)
- assume 21st century (2000-2074)

Dates specified with 4-digit years are considered valid if the year is within the range of 1901 to 8999.

DAYOFWK(date)

This function accepts a valid Julian date (yyddd or yyyyddd), a Gregorian/USA date (mm/dd/yy or mm/dd/yyyy), or a sort date (yymmd or yyyyymmd) and returns one of the following values: MON, TUE, WED, THU, FRI, SAT, or SUN indicating the day of the week on which the date falls.

HHMMSS(seconds)

This function accepts an integer value representing some number of seconds since midnight (<86400) and returns a 24-hour clock representation of the time in the format hh:mm:ss.

LEAPYEAR(‘date string’)

Character input: A date string in any of the following formats: yyddd, mmddyy, yymmd, yyyyddd, mmdyyyy, yyyyymmd, mm/dd/yy, or mm/dd/yyyy.

Boolean output: TRUE (1) if date contains a leap year; FALSE (0) if date does not contain a leap year. hh:mm:ss.

MMDDYY(Julian-date)

This function accepts a valid Julian date (yyddd or yyyyddd) and returns the corresponding Gregorian/USA date (mm/dd/yy or mm/dd/yyyy, respectively).

SECONDS(time)

This function accepts a time-of-day expressed in 24-hour format hh:mm:ss and returns the number of seconds past midnight.

YYDDD(Gregorian-date)

This function accepts a valid Gregorian/USA date (mm/dd/yy or mm/dd/yyyy) and returns the corresponding Julian date (yyddd or yyyyddd, respectively).
IMSCMD Function

Introduction

This unit describes the Candle-supplied REXX function, IMSCMD. The IMSCMD function is similar to the AF/OPERATOR IMSOPER command. Both allow you to send a command to IMS from a REXX exec. The IMSCMD function also allows you to receive the command response and to issue non-interactive IMS transactions.

Purpose

The IMSCMD function initiates a command interface with a specified IMS control region. IMSCMD allows you to issue an IMS command from a REXX exec and receive the complete command response or to issue a non-interactive IMS transaction. For commands, the function automatically returns the command response in REXX variables (one per line). These are accessible to your exec without further processing.

Using IMSCMD to issue commands

The IMSCMD function’s command interface to IMS is accomplished by defining AF/OPERATOR to IMS as a BMP (Batch Message Processing) region. Because AF/OPERATOR is defined as a BMP, the following cautions are necessary when using the IMSCMD function.

Caution

IMS views an AF/OPERATOR with an active BMP as an IMS-dependent region, and therefore it makes AF/OPERATOR non-cancellable. To shut down AF/OPERATOR, use the AF STOP command. Candle does not recommend using the AF KILL command under these circumstances since cancelling an IMS-dependent region may cause the IMS control region to terminate.

Constraints

Certain IMS commands cannot be sent using the IMSCMD function. The IMSCMD function does not support sending commands to an IMS DBCTL environment, because DBCTL does not support a BMP interface. These commands can be sent to IMS using the AF/OPERATOR IMSOPER command.

Required Customization and Configuration

Before you can use the IMSCMD function, various configuration and customization steps must be performed. For information, see the AF/OPERATOR Configuration and Customization Guide chapter entitled “Configuring the AF/OPERATOR BMP for IMSCMD and the IMS Transaction Execution Interface.”
Using IMSCMD to issue transactions

Before you can use the IMSCMD function to issue transactions, various configuration and customization steps must be performed. For information, see the *AF/OPERATOR Configuration and Customization Guide* chapter entitled “Configuring AF/OPERATOR to Permit IMSCMD to Issue Transactions.”

You can use the IMSCMD REXX function to issue non-interactive transactions.

1. The transaction issued must be non-interactive.
2. Response from the transaction is limited to single-segment messages routed specifically to the transaction code assigned to the BMP.
3. In your transaction, be sure to specify the correct return transaction code; otherwise you may get status **QH** from DL/I.
4. You must perform the appropriate configuration activities as described in the *AF/OPERATOR Configuration and Customization Guide*.

IMSCMD()

**Description**

IMSCMD initiates, sends commands, and receives responses from, or terminates a command interface to a specified IMS control region.

**Syntax**

```plaintext
>> IMSCMD(imsid,START, ,psb,agn,reslib) ) -><

SEND ,command ,timeout ,AUTO ,NOAUTO

STOP ,FORCE ,NOFORCE
```
IMSCMD Function

**imsid**
The system identifier of the IMS control region. IMSID must be four (4) characters or less.

**START**
Initiates a command interface with IMS.

**psb**
The name of the IMS PSB (Program Specification Block) associates with the AF/OPERATOR BMP. This is the name by which the IMS control region identifies AF/OPERATOR. The default PSB name is KOGIMPSB.

**agn**
The Application Group Name. If the Application Group Name is omitted, the PSBNAME specified is used as the Application Group Name. If the PSB parameter is omitted, the default PSBNAME specified during customization is used. If AGNs are not defined in IMS, this parameter is ignored.

This parameter is positional. Any parameters prior to the last parameter specified for START must have a comma inserted as a placeholder.

**reslib**
IMS load library. This parameter specifies a preallocated DDNAME for an IMS RESLIB to be used for this invocation of IMSCMD, and permits more than one release of IMS to be accessed by IMSCMD during any AF/OPERATOR session.

reslib is an optional parameter that is used only with the START function. The specified library will not be opened nor enqueued until the START function is issued. It will not be closed nor dequeued until the STOP function is issued. The default for this parameter is a null value. Any modules required to create an IMS BMP must reside in the standard MVS search order.

This parameter is positional. Any parameters prior to the last parameter specified for START must have a comma inserted as a placeholder.

**SEND**
Sends an IMS command, and receives and stores the command response in REXX variables.

**command**
An IMS command. The maximum command length is 240 character

**Notes:**
- IMS requires that commands be preceded by a command recognition character (such as /).
- The IMSCMD function does not provide a command recognition character for you.
- Commas are not allowed as delimiters in the IMS command, because they may cause a REXX syntax error. Use blanks as delimiters instead.

**timeout**
Optional. Specify, in seconds, the maximum time to wait for the IMS command response. The default is 2 seconds.

**AUTO**
Automatically issue an IMSCMD() START for the default PSB name, KOGIMPSB, if the AF/OPERATOR BMP is not already started. This is the default.

**NOAUTO**
Do not automatically issue an IMSCMD() START if the AF/OPERATOR BMP is not already started.

**STOP**
Terminates a command interface with IMS.

**Comments**
- IMSCMD can be invoked from a REXX exec using standard REXX syntax.
- If the command interface with IMS becomes unavailable, no automatic retry is attempted. See “IMSCMD() STOP” on page 103.
IMSCMD() START

Description
Initiates a command interface to the specified IMS control region.

Note: If OG/MVS's IMS DFSCMTI command processor is installed in the IMS region being accessed by an IMSCMD statement, the START/STOP sub-functions are not required and will be ignored if provided. The START/STOP sub-functions are only used when an IMS BMP is created in the OG/MVS region. When the DFSCMTI command processor is active in the target IMS region, no BMP is created or used in the OG/MVS region.

Syntax

```plaintext
IMSCMD(imsid,START start),psb,agn,reslib)
```

**imsid**
The system identifier of the IMS control region. IMSID must be four (4) characters or less.

**START**
Initializes the AF/OPERATOR BMP for the IMS control region specified by imsid.

**psb**
The name of the IMS PSB (Program Specification Block) associates with the AF/OPERATOR BMP. This is the name by which the IMS control region identifies AF/OPERATOR. The default PSB name is KOGIMPSB.

**agn**
The name of the IMS AGN (Application Group Name) associates with the AF/OPERATOR BMP. This is the RACF AGN associated with this BMP. The default AGN is KOGIMPSB.

**reslib**
The DDNAME of a pre-allocated IMS RESLIB to be used for this invocation of IMSCMD. This allows more than one release of IMS to be accessed by IMSCMD during any AF/OPERATOR session. The specified library will not be opened or enqueued until the START function is issued. It will not be closed or dequeued until the STOP function is issued. The default RESLIB is null. Any modules needed to create an IMS BMP will be expected to reside somewhere in the standard MVS search order.

Note: It is not necessary to issue an ALLOCATION command, a START command, a SEND command, a STOP command, and then a DEALLOCATION command sequentially in the same REXX exec.

You can issue the ALLOCATION and START commands in the morning, for example, SEND commands throughout the day, and then a STOP and DEALLOCATION at the end of the day.
**IMSCMD Function**

**Comments**
- The IMSCMD() START option can be invoked from a REXX exec using standard REXX syntax. It causes the specified AF/OPERATOR BMP to remain active until an IMSCMD() STOP is issued.
- AF/OPERATOR issues the following message when the command interface with IMS initiates:

  AOP0272 IMSCMD BMP STARTED for IMSID *imsid*

**Examples**
The following REXX exec fragment starts a command interface to an IMS control region named IMSP.

```rexx
/* REXX */
.*
.*
.*
rc = IMSCMD("IMSP", "START")
.*
.*
```

The following REXX exec fragment starts a command interface to an IMS control region names IMST and specifies a PSB and AGN.

```rexx
/* REXX */
.*
.*
.*
rc = IMSCMD("IMST","START","TESTPSB","TESTAGN")
.*
.*
```
Return Codes

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Completed successfully.</td>
</tr>
<tr>
<td>4</td>
<td>BMP is already active for this IMS control region.</td>
</tr>
<tr>
<td>8</td>
<td>AF/OPERATOR is terminating. No new IMS command interfaces can be started at this time.</td>
</tr>
<tr>
<td>12</td>
<td>IMS region controller (DFSRRC00) could not be found.</td>
</tr>
<tr>
<td>16</td>
<td>ATTACH of IMSCMD BMP failed.</td>
</tr>
<tr>
<td>20</td>
<td>AF/OPERATOR internal error.</td>
</tr>
<tr>
<td>24</td>
<td>IMSCMD internal error.</td>
</tr>
<tr>
<td>28</td>
<td>IMSID not active.</td>
</tr>
<tr>
<td>32</td>
<td>Optional RESLIB ALLOCATE/OPEN error.</td>
</tr>
<tr>
<td>36</td>
<td>Internal error.</td>
</tr>
<tr>
<td>40</td>
<td>Internal error.</td>
</tr>
<tr>
<td>44</td>
<td>Internal error.</td>
</tr>
<tr>
<td>48</td>
<td>Internal error.</td>
</tr>
</tbody>
</table>

IMSCMD() SEND

Description
Allows you to issue a command or a noninteractive transaction to a specified IMS control region and receive the response in REXX variables.

Syntax
To issue a command or a noninteractive transaction to the specified IMS control region and receive the complete response in REXX variables, use the format:

```
>> IMSCMD(imsid, SEND ,command ,AUTO [,timeout ,NOAUTO])
```
IMSCMD Function

**imsid**  The system identifier of the IMS control region. IMSID must be four (4) characters or less.

**SEND**  Passes an IMS command to the specified IMS control region.

**command**  An IMS command or transaction code. The maximum command length is 240 characters.

**Notes:**
1. IMS requires that commands be preceded by a command recognition character (such as `/`). The IMSCMD function does not provide a command recognition character for you.
2. Commas are not allowed as delimiters in the IMS command, because they may cause a REXX syntax error. Use blanks as delimiters instead.

**timeout**  Optional. Specify, in seconds, the maximum time to wait for the IMS response. The default is 2 seconds.

**AUTO**  Automatically issue an IMSCMD() START for the default PSB name, KOGIMPSB, if the AF/OPERATOR BMP is not already started. This is the default. The AUTO option is not available for PSB names other than KOGIMPSB.

**NOAUTO**  Do not automatically issue an IMSCMD() START for the default PSB name, KOGIMPSB, if the AF/OPERATOR BMP is not already started.

**tran**  An IMS transaction code.

**rc**  Return code

**Comments**
- The IMSCMD() SEND option can be invoked from a REXX exec using standard REXX syntax. An automatic IMSCMD() START is built into the IMSCMD() SEND option if you use the default PSB name, KOGIMPSB.
- The lines of command responses are returned in consecutively numbered REXX compound variables whose default names are imscmd.1, imscmd.2, and so on. The total number of lines returned is stored in variable imscmd.0.

**Note:** The command interface to IMS does not support sending commands to an IMS DBCTL environment. These commands can be sent to an IMS DBCTL system using the AF/OPERATOR IMSOPER command.
Examples
The following REXX exec fragment sends the command "/DIS TERM ALL" to an IMS control region named IMSP. Output data is returned in REXX variables named imscmd.

```rexx
/* REXX */
.
rc = IMSCMD("IMSP", "SEND", "/DIS TERM ALL")
do i = 1 to imscmd.0
  if pos("A123", imscmd.i) > 0 then say "found it"
end
.
.
```

Return Codes

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Completed successfully.</td>
</tr>
<tr>
<td>4</td>
<td>Neither the DFSCMTI or BMP is active for this control region, and NOAUTO was specified.</td>
</tr>
<tr>
<td>8</td>
<td>BMP is terminating. No new commands can be processed.</td>
</tr>
<tr>
<td>12</td>
<td>Internal error.</td>
</tr>
<tr>
<td>16</td>
<td>Command timed out.</td>
</tr>
<tr>
<td>20</td>
<td>AF/OPERATOR internal error.</td>
</tr>
<tr>
<td>24</td>
<td>Error writing REXX variables.</td>
</tr>
<tr>
<td>36</td>
<td>IMS security has determined that the command is not allowed.</td>
</tr>
<tr>
<td>40</td>
<td>DLI status ‘CH’.</td>
</tr>
<tr>
<td>44</td>
<td>DLI status other than ‘CH’.</td>
</tr>
<tr>
<td>48</td>
<td>Command sent is not a valid IMS command.</td>
</tr>
<tr>
<td>52</td>
<td>No alternate PCB found.</td>
</tr>
<tr>
<td>56</td>
<td>Error on DL/I CHNG, ISRT, or PURG call.</td>
</tr>
</tbody>
</table>
IMSCMD() STOP

Description
Terminates the command interface to the specified IMS control region.

Note: If OG/MVS’s IMS DFSCMTI command processor is installed in the IMS region being accessed by an IMSCMD statement, the START-STOP sub/functions are not required and will be ignored if provided. The START-STOP sub/functions are only used when an IMS BMP is created in the OG/MVS region. When the DFSCMTI command processor is active in the target IMS region, no BMP is created or used in the OG/MVS region.

Syntax

imsid The system identifier of the IMS control region. IMSID must be four (4) characters or less.

STOP Terminates the BMP in the IMS region specified by imsid. You can adjust how the IMSCMD function handles IMS commands in progress by explicitly specifying one of the following:

- **NOFORCE** Optional. Queued IMS commands are allowed to complete normally before terminating the BMP. This is the default if no specification follows the STOP instruction.
- **FORCE** Optional. The BMP terminates. Queued IMS commands are not processed.
Comments

- The IMSCMD() STOP option can be invoked from a REXX exec using standard REXX syntax.

- If the command interface with IMS becomes unavailable, no automatic retry is attempted. AF/OPERATOR issues one of the following message variations when the command interface with IMS terminates.
  
  **AOP0273 IMSCMD BMP STOPPED for IMSID imsid**
  
  or
  
  **AOP0273 IMSCMD BMP ABENDED for IMSID imsid**

  You can automate a restart by writing a trap for this message. A trap can specify a retry interval and the number of retry attempts.

- If the IMSCMD BMP abends, then IMS will /STOP the PSB and transaction associates with the AF/OPERATOR BMP. If this occurs, you (or the automated restart trap) will need to issue IMS commands to first /START the PSB, then /START the transaction, before the AF/OPERATOR BMP can be successfully started using the IMSCMD function.

Examples

The following REXX exec fragment stops a command interface to an IMS control region names IMSP.

```rexx
/* REXX */
.
.
.
rc = IMSCMD("IMSP", "STOP")
.
.
.
```

The following REXX exec fragment starts a command interface to an IMS control region named IMST and specifies a PSB and AGN.

```rexx
/* REXX */
.
.
.
rc = IMSCMD("IMST","START","TESTPSB","TESTAGN")
.
.
.
```
IMSCMD Function

Return Codes

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Completed successfully.</td>
</tr>
<tr>
<td>4</td>
<td>BMP is not active for this control region.</td>
</tr>
<tr>
<td>8</td>
<td>STOP already in progress.</td>
</tr>
<tr>
<td>12</td>
<td>STOP did not complete.</td>
</tr>
<tr>
<td>20</td>
<td>AF/OPERATOR internal error.</td>
</tr>
</tbody>
</table>
NetView Interface Functions

This section lists functions that allow you to:

- send commands to AF/OPERATOR
- return REXX variables values from AF/OPERATOR to the NetView REXX exec

AOCMD()

This function sends commands from NetView to be executed by an AF/OPERATOR specified using the subsystem name. It is an AF/OPERATOR REXX function that runs under NetView.

Syntax

```
AOCMD("command string", "subsys")
```

**command string**

The command you want to be executed by the specified AF/OPERATOR.

**subsys**

The target AF/OPERATOR address space. If the value is null or omitted, the target AF/OPERATOR address space is the first AF/OPERATOR address space started after the system IPL.

Example

This function causes the AF/OPERATOR subsystem with the subsystem name O320 to execute the command EX OGEXEC.

```
rc=AOCMD(“EX OGEXEC”,”O340")
```

The table below describes the function codes that the AOCMD() function may return. Messages, whose numbers match the function code numbers in this table, are issued when the function codes occur.
<table>
<thead>
<tr>
<th>Function Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>AOCMD() processing successful. AORC variable set.</td>
</tr>
<tr>
<td>100</td>
<td>Parameter error - no parameter passed.</td>
</tr>
<tr>
<td>101</td>
<td>Parameter error - too many parameters.</td>
</tr>
<tr>
<td>102</td>
<td>Parameter error - more than 255 characters.</td>
</tr>
<tr>
<td>108</td>
<td>Unable to allocate return data queue.</td>
</tr>
<tr>
<td>110</td>
<td>Error defining return data queue.</td>
</tr>
<tr>
<td>116</td>
<td>Unable to send command request.</td>
</tr>
<tr>
<td>118</td>
<td>NetView Interface terminated while waiting for data.</td>
</tr>
<tr>
<td>119</td>
<td>No data received in required time interval.</td>
</tr>
<tr>
<td>120</td>
<td>Unable to receive command reply.</td>
</tr>
<tr>
<td>121</td>
<td>Error in STIMER routine.</td>
</tr>
<tr>
<td>124</td>
<td>Error setting REXX variables from reply.</td>
</tr>
<tr>
<td>125</td>
<td>Invalid REXX variable name.</td>
</tr>
<tr>
<td>126</td>
<td>Invalid REXX variable value.</td>
</tr>
<tr>
<td>127</td>
<td>Other REXX error.</td>
</tr>
<tr>
<td>128</td>
<td>Error deallocating resources.</td>
</tr>
<tr>
<td>129</td>
<td>Error in queue allocation table.</td>
</tr>
<tr>
<td>132</td>
<td>Module is not APF authorized.</td>
</tr>
<tr>
<td>135</td>
<td>REXX environment blocks not found.</td>
</tr>
<tr>
<td>136</td>
<td>REXX CLIST name not found.</td>
</tr>
<tr>
<td>137</td>
<td>NetView environment blocks not found.</td>
</tr>
<tr>
<td>138</td>
<td>NetView Interface not available for processing.</td>
</tr>
<tr>
<td>140</td>
<td>AF/OPERATOR Version/Release incompatible.</td>
</tr>
<tr>
<td>141</td>
<td>Incoming data incomprehensible.</td>
</tr>
<tr>
<td>142</td>
<td>Incoming token precedes active token.</td>
</tr>
<tr>
<td>144</td>
<td>AF/OPERATOR not available for processing.</td>
</tr>
<tr>
<td>148</td>
<td>Purging inbound data.</td>
</tr>
<tr>
<td>152</td>
<td>Error closing down input processing.</td>
</tr>
<tr>
<td>156</td>
<td>Unable to allocate requisite storage.</td>
</tr>
<tr>
<td>160</td>
<td>NetView error, call type nn, rc=xxxxxxxx.</td>
</tr>
<tr>
<td>164</td>
<td>Inbound data record xxx ... xxx.</td>
</tr>
<tr>
<td>168</td>
<td>Defaults module length incompatible.</td>
</tr>
<tr>
<td>172</td>
<td>Defaults module Version/Release incompatible.</td>
</tr>
<tr>
<td>176</td>
<td>Error loading CNMNETV.</td>
</tr>
<tr>
<td>178</td>
<td>Error loading defaults.</td>
</tr>
<tr>
<td>179</td>
<td>Unable to locate SSCT.</td>
</tr>
<tr>
<td>180</td>
<td>Unable to locate Global area.</td>
</tr>
<tr>
<td>181</td>
<td>Global area eye-catcher invalid.</td>
</tr>
<tr>
<td>195</td>
<td>Command did not complete normally. AORC variable set.</td>
</tr>
<tr>
<td>196</td>
<td>Unknown error.</td>
</tr>
</tbody>
</table>

**Note:** The variable AORC is set for function codes 0 and 195.
NVIDATA()

NVIDATA() is an AF/OPERATOR REXX function. It returns the names and values of AF/OPERATOR REXX variables to the NetView REXX exec that called AF/OPERATOR via the NetView AOCMD() function. The names of the variables are passed as arguments of the function and are assumed to have been previously assigned a value in the AF/OPERATOR REXX environment.

Syntax

```plaintext
>>> NVIDATA(,'varname'[,'varname[,...]])) <<
```

**varname** Name of the AF/OPERATOR REXX variable whose value will be passed to the invoking NetView exec. REXX substitution does not occur in variable names.

Example

```plaintext
FC=NVIDATA(‘VAR1’, ‘VAR2’, ‘VAR3’)
```

The table below describes the function codes that the NVIDATA() function may issue.

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NVIDATA() processing was successful; data variables are returned to originating NetView exec.</td>
</tr>
<tr>
<td>4</td>
<td>NVIDATA() processing was bypassed. AF/OPERATOR cannot find a valid NetView Interface environment.</td>
</tr>
<tr>
<td>8</td>
<td>NVIDATA() processing was not successful. An invalid or null variable name was detected in the function call. If correct names were contained in the function prior to the error, those variables may have been passed.</td>
</tr>
<tr>
<td>12</td>
<td>Internal error encountered in call to send response via NetView Interface.</td>
</tr>
<tr>
<td>16</td>
<td>Internal error encountered; missing REXX Environment Block.</td>
</tr>
<tr>
<td>20</td>
<td>Internal error encountered; AF/OPERATOR environment error.</td>
</tr>
<tr>
<td>24</td>
<td>Internal error encountered attempting to obtain storage for a large variable. This can occur if there is not enough virtual storage to satisfy a request for a very large variable.</td>
</tr>
<tr>
<td>25</td>
<td>Internal error encountered attempting to release storage obtained for a large variable.</td>
</tr>
<tr>
<td>26</td>
<td>Internal error encountered attempting to release storage obtained for a large variable. This error occurred after passing variable data to the NetView Interface.</td>
</tr>
<tr>
<td>27</td>
<td>Internal error encountered calling REXX variable services (IRXEXCOM) after obtaining storage for a large variable.</td>
</tr>
<tr>
<td>28, 32, -1, -2, (and others)</td>
<td>Internal error encountered calling REXX variable services (IRXEXCOM). These are set by IRXEXCOM and returned by this function as 3-byte function codes with leading zeros.</td>
</tr>
</tbody>
</table>
OMEGAVIEW Interface Functions

This section lists functions that allow you to:

- send status information to AF/OPERATOR
- retrieve status information from OMEGAVIEW
- send status information to OMEGAVIEW
- send alerts from OMEGAVIEW to AF/OPERATOR

*Note:* You must use the LOGON command to establish a session between AF/OPERATOR and OMEGAVIEW before you attempt to invoke any OMEGAVIEW Interface REXX function.

The REXX functions listed in this chapter can be invoked only from AF/OPERATOR REXX execs. These functions are the basis for communication between AF/OPERATOR and OMEGAVIEW.

Summary of OMEGAVIEW functions

The table below explains the use of AF/OPERATOR OMEGAVIEW functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STCLOSE()</td>
<td>Delete an access path to an OMEGAVIEW status item</td>
</tr>
<tr>
<td>STCREATE()</td>
<td>Create status items</td>
</tr>
<tr>
<td>STDSTRY()</td>
<td>Destroy OMEGAVIEW status items</td>
</tr>
<tr>
<td>STGET()</td>
<td>Retrieve OMEGAVIEW status item data from items previously opened with STOPEN()</td>
</tr>
<tr>
<td>STOPEN()</td>
<td>Create an access path to an OMEGAVIEW status item</td>
</tr>
<tr>
<td>STSIGNAL()</td>
<td>Update the status color of a status item by changing its value</td>
</tr>
<tr>
<td>STSSTO()</td>
<td>Change a status item’s timeout value</td>
</tr>
<tr>
<td>STSSTOD()</td>
<td>Set a default timeout value for all status items</td>
</tr>
<tr>
<td>STUPDATE()</td>
<td>Modify OMEGAVIEW status item field information, such as action and problem management fields</td>
</tr>
</tbody>
</table>

Sequence Rules

Certain OMEGAVIEW Interface functions are valid only after other functions or commands have been issued. The table below describes the logical order in which to use OMEGAVIEW Interface functions.

<table>
<thead>
<tr>
<th>Before you issue . . .</th>
<th>Issue . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>STCREATE() or STSSTOD()</td>
<td>LOGON (omview)</td>
</tr>
<tr>
<td>STOPEN()</td>
<td>STCREATE() If the item does not already exist</td>
</tr>
</tbody>
</table>
**STCLOSE()**

STCLOSE() closes a previously opened status item and changes its color to turquoise to indicate no status. The access path created by STOPEN() is destroyed. If the STCLOSE() function ends with a return code of 0, the SDMLINE stem variables will be updated. SDMLINE.0 will contain the total number (n) of valid variables. SDMLINE.2 through SDMLINE.n will contain the names of the columns for the closed status item. Non-zero return codes will leave the SDMLINE stem variables in an indeterminate state.

**Syntax**

```
>> STCLOSE(name, item, mode) <<
```

**name**

The name of an AF/OPERATOR--OMEGAVIEW session as defined with the LOGON command.

**item**

A 1- to 20-character name of a status item previously opened via an STOPEN() function call.

**mode**

- **CLEAR**
  
  Clears the status columns for the closed status item so that old data will not be mistaken for current data. This is the default. The closed status item will turn turquoise, indicating NO DATA.

- **NOCLEAR**

  Preserves status data for the closed status item. The closed status item retains the status and color of its last update.

**Example**

This exec closes the status item CICSTIME known to the session OMV1.

```
RET=STCLOSE(OMV1, CICSTIME, CLEAR)
```
**STCREATE()**

STCREATE() creates a new status item that will be controlled via AF/OPERATOR automation events. The status columns SDMSTNAM, SDMDESCR, and SDMRULE are set by the function using the parameters described below. Unless you specify otherwise, the remaining status columns will be set to the values currently stored in the corresponding REXX variables. You should set all your SDM column variables (SDMAGENT, SDMTEXT and so forth) to null, before issuing STCREATE(), to avoid unexpected value carryover. If the STCREATE() function ends with a return code of 0, the SDMLINE stem variables will be updated. SDMLINE.0 will contain the total number (n) of valid variables. SDMLINE.2 through SDMLINE.n will contain the names of the columns for the created status item. Non-zero return codes will leave the SDMLINE stem variables in an indeterminate state.

**Syntax**

```
>>STCREATE(name,item,desc,rule
             'colname')
```

{name}  
The name of an AF/OPERATOR–OMEGAVIEW session as defined with the LOGON command.

{item}  
A 1- to 20-character name for the new status item. This value is stored in the SDMSTNAM column.

{descr}  
A 1- to 30-character description of the status item. This value is stored in the SDMDESCR column.

{rule}  
Defines an OMEGAVIEW status rule. This value is stored in the SDMRULE column.

{colname}  
Optional. One or more column names of 1 to 8 characters in length. Column names are enclosed in quotes and separated by a comma. They supply starter data for the item in the OMEGAVIEW database. Each colname value must already be assigned by a REXX variable.

**Note:** If you have a string of column names that spans multiple lines, REXX includes the blanks from the end of one record to the beginning of the next record as part of the input. To avoid this problem, you must use the concatenation operator (||) as a part of the quoted string as in the example below:

```
STCREATE(A,B,C,D,,
         'E,F,G;' ||,
         'H,I,J')
```

This example produces the correct quoted string ‘E,F,G,H,I,J’.
**Comments**

- Candle recommends using the null status rule. If you prefer to use the threshold rule, see the *AF/OPERATOR User’s Guide* for information.
- Invoke STCREATE() to define any dynamically created status items *before* accessing them with STOPEN().

**Example**

This exec creates the status item CICSTIME (for a CICS™ system known as CICS).

```rexx
/* REXX */
sdmtext='this is test user text'
sdmcnt='this is test item comment'
RET=STCREATE(OMV1,CICSTIME,'CICS overall response time',,
'CRITICAL,GE,2000,WARNING,GE,0500,OK,GE,0000','SDMTEXT','SDMCOUNT')```

**STDSTRY()**

STDSTRY() deletes a previously defined status item. If the STDSTRY() function ends with a return code of 0, the SDMLINE stem variables will be updated. SDMLINE.0 will contain the total number (n) of valid variables. SDMLINE.2 through SDMLINE.n will contain the names of the columns for the destroyed status item. Non-zero return codes will leave the SDMLINE stem variables in an indeterminate state.

**Syntax**

```rexx
> stdstry(name,item) <<
>>
name
The name of an AF/OPERATOR–OMEGAVIEW session as defined with the LOGON command.

item
A 1- to 20-character name of the status item to be deleted.

**Note:** *Unless an item is deleted via STDSTRY(), the OMEGAVIEW database retains its definition.*
Example
This exec removes the status item CICSTIME from the OMEGAVIEW database.

```plaintext
RET=STDSTRY(OMV1,CICSTIME)
```

**STGET()**

STGET() checks the current values of field names associated with a particular status item and returns status information by column name. The STGET() function will not modify the SDMLINE stem variables.

**Syntax**

```plaintext
>> ___ STGET(name,item) _________ >>
```

**name**

The name of an AF/OPERATOR-OMEGAVIEW session as defined with the LOGON command.

**item**

A 1- to 20-character name of a status item previously opened via an STOPEN() function call.

Example
This exec gets column information for the status item CICSTIME and displays the contents of the column name SDMDESCR.

```plaintext
RET=STGET(OMV1,CICSTIME)
say 'sdmdescr= ' sdmdescr
-----------------------------------------------------------------
SDMDESCR= CICS OVERALL RESPONSE TIME
```

**STOPEN()**

STOPEN() opens a previously created status item. If the STOPEN() function ends with a return code of 0, the SDMLINE stem variables will be updated. SDMLINE.0 will contain the total number (n) of valid variables. SDMLINE.2 through SDMLINE.n will contain the names of the columns for the opened status item. Non-zero return codes will leave the SDMLINE stem variables in an indeterminate state.
OMEGAVIEW Interface Functions

Syntax

>> ___ STOPEN(name,item,mode) _________ <<

name

The name of an AF/OPERATOR–OMEGAVIEW session as defined with the LOGON command.

item

A 1- to 20-character name of a status item previously created via an STCREATE() function.

mode

INPUT Specifies that the opened item can be updated by other sessions logged onto OMEGAVIEW.

OUTPUT Specifies that the opened item can be updated only by the session that issued the STOPEN() call. This is the default.

Note: Invoke STCREATE() to define a status item before opening it with STOPEN().

Example

This exec opens the status item CICSTIME known to the session OMV1. All columns are accessible.

RET=STOPEN(OMV1,CICSTIME,INPUT)

ST SIGNAL()

ST SIGNAL() updates the status value of a status item.

Note: STUPDATE() cannot be used in place of ST SIGNAL(). If the SIGNAL() function ends with a return code of 0, the SDMLINE stem variables will be updated. SDMLINE.0 will contain the total number (n) of valid variables. SDMLINE.2 through SDMLINE.n will contain the names of the columns for the updated status item. Non-zero return codes will leave the SDMLINE stem variables in an indeterminate state.

ST SIGNAL cannot be used to change the light value for an aggregated status item, since the light shown in this case is an aggregation of the status items of all of the rolled-up item’s components. Using ST SIGNAL in this case will cause a return code 21, operation not supported.

Note: Null values are not valid for transient text when using ST SIGNAL.
Syntax

```plaintext
>>STSIGNAL(name,item,level text_variable_name)
```

**name**

The name of an AF/OPERATOR–OMEGAVIEW session as defined with the LOGON command.

**item**

A 1- to 20-character name of an opened status item whose level is to be modified.

**level**

The acceptable level values are 0, 10, 20, and 30 for items with the null rule, any integer for items with the threshold rule.

**text_variable_name**

Optional. A 1- to 8-character name of a variable that contains up to 128 characters of text to be displayed with the status item. The variable name must be enclosed in quotes. The value of the variable must be assigned using a REXX variable assignment.

**Note:** You can also use the SAMPLIB routine MVCHANGE to update the status value of a status item.

**Example**

This exec signals the status level of 10 (OK) for the status item CICSTIME in the session OMV1 with a variable called VARNAME, which contains a text description for the status item.

```plaintext
/* REXX */
varname='this is descriptor text'
RET=STSIGNAL(OMV1,CICSTIME,10,'VARNAME')
```

**STSSTO()**

STSSTO() changes the status item’s timeout value. If the STSSTO() function ends with a return code of 0, the SDMLINE stem variables will be updated. SDMLINE.0 will contain the total number (n) of valid variables. SDMLINE.2 through SDMLINE.n will contain the names of the columns for the referenced status item. Non-zero return codes will leave the SDMLINE stem variables in an indeterminate state.
**Syntax**

```
>> ___ STSSTO(name, item, interval) ___
```

**name**

The name of an AF/OPERATOR–OMEGAVIEW session as defined with the LOGON command.

**item**

A 1- to 20-character name of the status item whose level is to be modified.

**interval**

The duration to wait in seconds before the status item times out, assumes a status level of NONE, and turns turquoise. The acceptable values are whole numbers between 0 and 9999.

A value of zero (0) disables the timeout feature for this status item, indicating an infinite timeout.

**Example**

This exec sets the status item timeout interval to 90 seconds for the status item CICSTIME in the session OMV1.

```
RET=STSSTO(OMV1, CICSTIME, 90)
```

**STSSTOD()**

STSSTOD() sets a default timeout value for all status items whose names match the high-level qualifier. If the STSSTOD() function ends with a return code of 0, the SDMLINE stem variables will be updated. SDMLINE.0 will contain the total number (n) of valid variables. SDMLINE.2 through SDMLINE.n will contain the names of the columns for the referenced status item. Non-zero return codes will leave the SDMLINE stem variables in an indeterminate state.

**Syntax**

```
>> ___ STSTOD(name, prefix, interval) ___
```

**name**

The name of an AF/OPERATOR–OMEGAVIEW session as defined with the LOGON command.

**prefix**

The high-level qualifier of a status item name. The timeout interval is for all status items whose high-level qualifier matches this value.
interval

The duration to wait in seconds before all status items matching the high-level qualifier time out. The acceptable values are whole numbers between 0 and 9999.

A value of zero (0) disables the timeout feature for this status item, indicating an infinite timeout.

Example

This exec sets the default status item timeout interval to 90 seconds.

```
RET=STUPDATE(OMV1,WEBUNDPAYRL,90)
```

**STUPDATE()**

STUPDATE() modifies OMEGAVIEW status item information, such as action and problem management fields. See “Status Column Definitions” on page 447 for a table of status column definitions modifiable by STUPDATE(). For example, to add problem management information to a status item, you must modify the SDMAGENT column and, if you wish, the SDMADESC, SDMCOMNT, SDMSFXDT, and SDMXFXDT columns. The STUPDATE() function will not modify the SDMLINE stem variables.

**Note:** STUPDATE() cannot be used in place of STSIGNAL().

**Syntax**

```
>>STUPDATE(name,item,'colname',...)
```

**name**

The name of an AF/OPERATOR–OMEGAVIEW session as defined with the LOGON command.

**item**

A 1- to 20-character name of the status item whose level is to be modified.

**colname**

A 1- to 8-character column name. See “Status Column Definitions” on page 447 for a list of acceptable values for this field.

**Note:** Column names are enclosed in quotes and separated by a comma. The colname value must be assigned via a REXX variable assignment.

**Note:** The OMEGAVIEW Interface can update any status item known to the OMEGAVIEW system with which it is in session. For non-AF/OPERATOR-created status item access, invoke STOPEN() to specify the item’s name.
Example
This exec updates two status columns for the status item CICSTIME in the session OMV1.

```rexx
/* REEXX */
sdmdescr='this is descriptor text'
sdmcomnt='this is a test item comment'
RET=STUPDATE(OMV1,CICSTIME,'SDMDESCR','SDMCOMNT')
```

Notes
1. If the item is new, you must invoke the STCREATE() function before issuing any other function.
2. If the status item is pre-defined or has already been created by OMEGAVIEW, open the item with the STOPEN() function before using any other function.

OMEGAVIEW Return Codes
The following table describes the return codes that OMEGAVIEW may send to AF/OPERATOR through the interface after the AF/OPERATOR REXX exec has executed. See the OMEGAVIEW documentation for a complete listing of OMEGAVIEW return codes.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command processing successful.</td>
</tr>
<tr>
<td>1</td>
<td>Not Found. The status item could not be found in the status database.</td>
</tr>
<tr>
<td>2</td>
<td>Exists. A status item with this name already exists.</td>
</tr>
<tr>
<td>3</td>
<td>Shortage. Storage allocation failed.</td>
</tr>
<tr>
<td>4</td>
<td>Format Error. The status name or rule was improperly formatted.</td>
</tr>
<tr>
<td>5</td>
<td>Bad Handle. The session handle was invalid.</td>
</tr>
<tr>
<td>6</td>
<td>Table Error. The status item could not be deleted from the database.</td>
</tr>
<tr>
<td>7</td>
<td>Destroyed. The status item has already been destroyed.</td>
</tr>
<tr>
<td>8</td>
<td>Bad Variable Name. A variable name in the rule was invalid.</td>
</tr>
<tr>
<td>9</td>
<td>Missing Relation. A relational operator in the rule was expected but not supplied.</td>
</tr>
<tr>
<td>10</td>
<td>Missing Value. A value in the rule was expected but not supplied.</td>
</tr>
<tr>
<td>14</td>
<td>Bad Function Name. A function name in the rule was invalid.</td>
</tr>
<tr>
<td>15</td>
<td>Bad Function Format. The rule contained an improperly formatted function invocation.</td>
</tr>
<tr>
<td>16</td>
<td>Unbalanced Parenthesis. The rule did not properly match opening and closing parentheses.</td>
</tr>
<tr>
<td>17</td>
<td>Bad Integer Value. The rule contained an invalid integer value.</td>
</tr>
</tbody>
</table>
OMEGAVIEW Interface Functions

AF/OPERATOR Return Codes

The following table describes return codes that AF/OPERATOR may return after REXX exec execution using the OMEGAVIEW Interface functions.

**Note:** For all the return codes below, function processing is terminated, and control returns to the REXX exec.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1008</td>
<td>Invalid input argument length.</td>
</tr>
<tr>
<td>1009</td>
<td>Invalid level value specified on STSIGNAL().</td>
</tr>
<tr>
<td>1030</td>
<td>Error encountered while creating a REXX variable.</td>
</tr>
<tr>
<td>1040</td>
<td>STOPEN() has not been issued for this status item.</td>
</tr>
<tr>
<td>1112</td>
<td>Invalid number of input arguments.</td>
</tr>
<tr>
<td>1113</td>
<td>Application name is invalid or was not found.</td>
</tr>
<tr>
<td>1114</td>
<td>Application buffer returned is not valid or is not parseable.</td>
</tr>
<tr>
<td>1115</td>
<td>Invalid mode (INPUT/OUTPUT) specified on STOPEN().</td>
</tr>
<tr>
<td>1117</td>
<td>Error encountered while trying to locate the AF/OPERATOR module vector</td>
</tr>
<tr>
<td>1118</td>
<td>Invalid mode (CLEAR/NOCLEAR) specified on STCLOSE().</td>
</tr>
<tr>
<td>2001</td>
<td>Missing REXX environment block.</td>
</tr>
<tr>
<td>2003</td>
<td>Bad REXX environment-block parameter list.</td>
</tr>
<tr>
<td>2020</td>
<td>REXX communication error encountered.</td>
</tr>
<tr>
<td>2022</td>
<td>Storage shortage. Getmain failed; function processing is terminated.</td>
</tr>
<tr>
<td>2023</td>
<td>IRXEXCOM error.</td>
</tr>
<tr>
<td>2030</td>
<td>Duplicate status item error. A status item with the same status name is already open for this session name.</td>
</tr>
<tr>
<td>2040</td>
<td>Enqueue failed.</td>
</tr>
<tr>
<td>2050</td>
<td>Buffer exceeded.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>2060</td>
<td>The session specified in the function is no longer logged on.</td>
</tr>
<tr>
<td>2065</td>
<td>The session specified in the function lost its associated pseudo-match, leaving it in an unpredictable state.</td>
</tr>
<tr>
<td>3000</td>
<td>A VTAM error was encountered while processing this request.</td>
</tr>
<tr>
<td>3001</td>
<td>The maximum length of an STCREATE column name argument was exceeded.</td>
</tr>
<tr>
<td>3002</td>
<td>A syntax error occurred for an STCREATE column name argument.</td>
</tr>
</tbody>
</table>
This section describes the functions for the Programmerless Open VTAM Interface (POVI), which is an automation interface you can use to automate interactions between AF/OPERATOR and VTAM 3270 applications.

For information on using POVI, see the AF/OPERATOR Configuration and Customization Guide.

**POVIDATA**

Returns screen data for a specified location and length from an environment that was created using the POVILGON function.

**Syntax**

```
POVIDATA(row, column, length, {trace})
```

- **row** The 1- to 2-digit number of the row from which data will be returned.
- **column** The 1- to 3-digit column number from which data will be returned.
- **length** The 1- to 4-digit length of the returned data.
- **trace** Turns on internal trace of primitive functions.

**Function Codes**

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion. Screen data has been pushed onto data stack.</td>
</tr>
<tr>
<td>1</td>
<td>End of data wraps the screen. Screen wrapping is not supported.</td>
</tr>
<tr>
<td>2</td>
<td>Unbind received for session. See AF/OPERATOR log.</td>
</tr>
<tr>
<td>3</td>
<td>Invalid number of arguments.</td>
</tr>
<tr>
<td>4</td>
<td>Invalid argument length. The argument name is pushed onto the data stack.</td>
</tr>
<tr>
<td>7</td>
<td>Environment not active. See the AF/OPERATOR log. The environment name is pushed onto the data stack.</td>
</tr>
<tr>
<td>8</td>
<td>Environment not addressed. The environment name is pushed onto the data stack.</td>
</tr>
<tr>
<td>9</td>
<td>No POVILGON environment is active.</td>
</tr>
<tr>
<td>15</td>
<td>Row number is too large for the model.</td>
</tr>
<tr>
<td>16</td>
<td>Column number is too large for the model.</td>
</tr>
<tr>
<td>24</td>
<td>GLBVPUT error. The variable RESULT is pushed onto the data stack.</td>
</tr>
<tr>
<td>25</td>
<td>GLBVGET error. The variable RESULT is pushed onto the data stack.</td>
</tr>
<tr>
<td>101</td>
<td>Severe error. Contact Candle Support Services.</td>
</tr>
</tbody>
</table>
Comments

- POVIDATA returns function code 9 when no environment had been created using POVILGON.
- POVIDATA returns function code 7 when the environment created using POVILGON was terminated by means other than POVILGOF.
- Screen positions that do not contain visible data (that is, spaces, attributes and nulls) are translated to spaces in the screen buffer.
- If no stacked data is described for a non-zero function code, then a null value is pushed onto the stack by default.

Example

The following example shows how to log onto an application (identified as TSOA during POVI installation) and extract data from the screen buffer.

```
toolsrc = POVILGON(OVIAO, TSOA, TSOA)
IF (toolsrc <> 0) THEN CALL ERROR_HANDLER
 toolsrc = POVIDATA(1, 18, 17)
IF (toolsrc<>0) THEN CALL ERROR_HANDLER
FULL screen_data
IF (screen_data<> "ENTER LOGON ID - ") THEN ..... 
```

POVIDLAY()  

Waits a specified number of seconds, then refreshes the terminal buffer of an environment created using the POVILGON function.

Syntax

```
POVIDLAY(n, {trace})
```

- **n**  
  Wait time in seconds.

- **trace**  
  Turns on internal trace of primitive functions.

Function Codes

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Wait completed and no errors were detected.</td>
</tr>
<tr>
<td>9</td>
<td>No POVILGON environment is active.</td>
</tr>
<tr>
<td>23</td>
<td>Data transmission error. See the AF/OPERATOR log.</td>
</tr>
<tr>
<td>25</td>
<td>GLBVGET error. The variable RESULT is pushed onto the data stack.</td>
</tr>
</tbody>
</table>
Comments
- POVIDLAY can be used when the application needs time to complete screen processing before POVISEND can be used to send more data to the application.
- If no stacked data is described for a non-zero function code, then a null value is pushed onto the stack by default.

POVIFIND()
Locates a string in a screen buffer and returns the row and column of either the start of the string or an offset from the start of the string. The environment for the session must have been created using the POVILGON function.

Syntax
   POVIFIND(TOP|HERE,string,{offset}, {trace})

TOP
   Begins searching for the data at the first screen position.

HERE
   Begins searching for the data at the cursor position.

string
   The string you want to search for. If no optional offset is specified, the row and column of the starting point of the string is returned.

offset
   Adjusts the string’s starting row and column forward by the specified offset. You can use 1 to 4 characters. This argument is optional.

trace
   Turns on internal trace of primitive functions.

Function Codes

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion. The string was found. Row and column have been pushed onto the external data stack. The row is pushed first, followed by the column.</td>
</tr>
<tr>
<td>2</td>
<td>Unbind received for session. See the AF/OPERATOR log.</td>
</tr>
<tr>
<td>3</td>
<td>Invalid number of arguments.</td>
</tr>
<tr>
<td>4</td>
<td>Invalid argument length. The argument name is pushed onto the data stack.</td>
</tr>
<tr>
<td>7</td>
<td>Environment is not active. See the AF/OPERATOR log. The environment name is pushed onto the data stack.</td>
</tr>
<tr>
<td>8</td>
<td>Environment is not addressed. The environment name is pushed onto the data stack.</td>
</tr>
</tbody>
</table>
The offset is relative to 0, which means the beginning of the string is at offset 0.

POVIFIND returns a function code of 21 if the offset from the located data start would occur after the end of the screen. Screen wrapping is not supported.

It is more efficient to search for data from the current cursor position if you know the data occurs later. If you know the location of the data, you should extract the data using Povidata and then compare it to the expected field contents, rather than use POVIFIND.

Searching is case sensitive.

Screen positions which do not contain visible data (that is, spaces, attributes and nulls) are translated to spaces in the screen buffer.

If no stacked data is described for a non-zero function code, then a null value is pushed onto the stack by default.

Examples
The following example assumes that the current screen is the ISPF Main Menu. It uses POVIFIND to locate the input field for the dataset name on the EDIT primary menu and then invokes EDIT. POVIFIND is used to locate the dataset name input field.

```
toolsrc = POVISEND(2, ENTER)
IF (toolsrc <> 0) THEN CALL ERROR_HANDLER
toolsrc = POVIFIND(TOP, "DATASET NAME ===" , 20)
IF (toolsrc <> 0) THEN CALL ERROR_HANDLER
PULL column
PULL row
toolsrc = POVILOC(ROWCOL, row, column)
IF (toolsrc <> 0) THEN CALL ERROR_HANDLER
toolsrc = POVISEND("our.dataset.name", ENTER, "EDIT", 1, 2)
IF (toolsrc <> 0) THEN CALL ERROR_HANDLER
.
.
.
(edit the dataset)
```
POVI Functions

POVILOGOF()

Terminates an environment that was created using POVILOGON and logs off the associated VTAM application.

Syntax

```
POVILOGOF((trace))
```

**trace**

Turns on internal trace of primitive functions.

**Function Codes**

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful logoff.</td>
</tr>
<tr>
<td>9</td>
<td>No POVILOGON environment is active.</td>
</tr>
<tr>
<td>25</td>
<td>GLBVGET error. The variable RESULT is pushed onto the data stack.</td>
</tr>
<tr>
<td>26</td>
<td>LOGOFF command error. The LOGOFF command return code is pushed onto the data stack.</td>
</tr>
</tbody>
</table>

**Comments**

1. When POVILOGOF is used to logoff the AF/OPERATOR virtual terminal from the POVI Playback Manager, any active VTAM 3270 application session is also terminated.
2. If no stacked data is described for a non-zero function code, then a null value is pushed onto the data stack.
POVI Functions

POVILGON()

Creates a POVI environment with the POVI Playback Manager, addresses the environment, and then logs onto a session with a VTAM 3270 application.

Syntax

POVILGON(pbmgrapplid, session id, {environment name}, {model number}, {timeout}, {trace})

pbmgrapplid

The VTAM applid for the POVI Playback Manager. This is defined using the Customizer.

session id

The 1- to 8-character ID assigned during customization and used for a session created with a VTAM 3270 application. These are names described as sessids on the POVI Recording Manager Main Menu.

environment name

The handle for the session with the POVI Playback Manager. A handle is a unique identifier for a session. The default is Mn, where n is the numeric match number with no leading zeros.

model number

The model number for the 3270 device being simulated. The default is 2.

timeout

Specifies in seconds the maximum time that a session is expected to take to respond to a logon request. If this time is exceeded, the session will be logged off and the appropriate return code set. If 0 is specified, then requests will wait for a response and never time out. The maximum allowed is 9999 seconds. The default TIMEOUT is specified by means of the LOGON_RESPTIME startup parameter.

trace

Turns on internal trace of primitive functions.

Function Codes

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion. VTAM 3270 application session is ready for processing.</td>
</tr>
<tr>
<td>5</td>
<td>Invalid argument. The argument name is pushed onto the data stack.</td>
</tr>
<tr>
<td>6</td>
<td>A required argument is missing. The argument name is pushed onto the data stack.</td>
</tr>
<tr>
<td>10</td>
<td>Another environment has already been addressed using POVILGON and is currently active. The environment name is pushed onto the data stack.</td>
</tr>
<tr>
<td>11</td>
<td>CT/Engine™ error message is returned. The message is pushed onto the data stack.</td>
</tr>
</tbody>
</table>
### POVI Functions

#### Comments

1. Function codes of 5, 6, and 10 are programming errors. Function values 18, 19 and 20 typically result from installation errors.

2. POVILGON will de-address any currently active or queued environments which have not been started using POVILGON.

3. POVILGON returns function code 10 when it is used to create a second environment before the first environment has been ended using POVLGOF.

4. Since duplicate environments names are prohibited, only one environment with the default name can be created. Environment names are global entities for the entire product so they must be managed. For example, “match1” and “match2” cannot concurrently create environments having the same name. However, a given environment name would be serially reusable by the matches.

5. No sessions are left active after a non-zero function code is returned.

6. If no stacked data is described for a non-zero function code, then a null value is pushed onto the stack.

#### Examples

The following example shows a logon to the application with sessid TSOA.

```
IF (POVILGON(OVIAO, TSOA, TSOA, 2) <> 0) THEN CALL ERROR_HANDLER
    
    (use the session)
```
POVILOC()

Positions the cursor for the simulated 3270 screen. The environment must have been created using the POVILGON function.

Syntax

POVILOC(type, arg1, {arg2}, {trace})

*type* Specifies where you want to position the cursor.

- **NEXTDATA**
  Locates the specified data and positions the cursor at an offset from the start of that data. The search begins from the current cursor position.

- **FIRSTDATA**
  Locates the specified data and positions the cursor at an offset of that data. The search begins at the top of the screen.

- **ROWCOL**
  Positions the cursor at the specified row and column.

*arg1* Depends on the value you used for the type argument.

- **NEXTDATA**
  Data to search for.

- **FIRSTDATA**
  Data to search for.

- **ROWCOL**
  Row where you want to position the cursor.

*arg2* Depends on the value you used for the type argument.

- **NEXTDATA**
  An offset from the start of the located data. This is optional.

- **FIRSTDATA**
  An offset from the start of the located data. This is optional.

- **ROWCOL**
  Column where you want to position the cursor.

*trace* Turns on internal trace of primitive functions.
Function Codes

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion. Cursor has been repositioned.</td>
</tr>
<tr>
<td>2</td>
<td>Unbind received for session. See the AF/OPERATOR log.</td>
</tr>
<tr>
<td>3</td>
<td>Invalid number of arguments.</td>
</tr>
<tr>
<td>5</td>
<td>Invalid argument. The argument name is pushed onto the data stack.</td>
</tr>
<tr>
<td>6</td>
<td>A required argument is missing. The argument name is pushed onto the data stack.</td>
</tr>
<tr>
<td>7</td>
<td>Environment not active. See the AF/OPERATOR log. The environment name is pushed onto the data stack.</td>
</tr>
<tr>
<td>8</td>
<td>Environment not addressed. The environment name is pushed onto the data stack.</td>
</tr>
<tr>
<td>9</td>
<td>No POVILGON environment is active.</td>
</tr>
<tr>
<td>14</td>
<td>An internal invocation of the POVIFIND function has failed. The POVIFIND function is pushed onto the data stack.</td>
</tr>
<tr>
<td>23</td>
<td>Data transmission error. See the AF/OPERATOR log.</td>
</tr>
<tr>
<td>24</td>
<td>GLBVPUT error. The variable RESULT is pushed onto the data stack.</td>
</tr>
<tr>
<td>25</td>
<td>GLBVGET error. The variable RESULT is pushed onto the data stack.</td>
</tr>
<tr>
<td>101</td>
<td>Severe error. Contact Candle Support Services.</td>
</tr>
</tbody>
</table>

Comments

1. It is more efficient to specify the exact screen location for cursor positioning using ROWCOL than to unnecessarily search the screen buffer using NEXTDATA or FIRSTDATA.

2. The FIRSTDATA and NEXTDATA options are available for infrequent situations such as the appearance of pop-up menus, which may be output to different locations on a 3270 screen each time.

3. Searching is case sensitive so be sure to specify the correct case for the search argument.

4. If no stacked data is described for a non-zero function code, then a null value is pushed onto the stack by default.

Examples

The following example searches for a field labeled with “DATASET NAME ===>” and positions the cursor at an offset of 20 character positions after the start of the label.

toolsrc=POVILOC(NEXTDATA,"DATASET NAME ==>", 20)
IF (toolsrc<0) THEN CALL ERRHNDLR
POVIPBS()

Initiates playback of a Keystroke Recorder script.

Syntax

POVIPBS(scriptname, {trace})

**scriptname**  The name of a script that was recorded using the POVI Keystroke Recorder.

**trace**  Turns on internal trace of primitive functions.

**Function Codes**

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Script playback complete.</td>
</tr>
<tr>
<td>9</td>
<td>No POVILGON environment is active.</td>
</tr>
<tr>
<td>23</td>
<td>Data transmission error. See the AF/OPERATOR log.</td>
</tr>
<tr>
<td>25</td>
<td>GLBVGET error. The variable RESULT is pushed onto the data stack.</td>
</tr>
</tbody>
</table>

**Comments**

1. The VTAM 3270 application screen data is not forwarded to the controlling EXEC for the duration of playback. Once playback has ended, you can access the current screen using POVIFIND or POVIDATA.

2. If no stacked data is described for a non-zero function code, then a null value is pushed onto the stack by default.

POVIRSTR()

Restores a session environment for use by POVI REXX tools from a handle created using POVISAVE.

Syntax

POVIRSTR(handle, {trace})

**handle**  A handle that had been created using POVISAVE for a POVI REXX function session environment that is to be restored.

**trace**  Turns on internal trace of primitive functions.
POVI Functions

Function Codes

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>POVI REXX function session environment has been restored based on information in the handle.</td>
</tr>
<tr>
<td>19</td>
<td>AFADDR command failed. See the AF/OPERATOR log. The AFADDR command return code is pushed onto the data stack.</td>
</tr>
<tr>
<td>24</td>
<td>GLBVPUT failed.</td>
</tr>
<tr>
<td>27</td>
<td>Invalid handle.</td>
</tr>
</tbody>
</table>

Comments
1. Handles may be passed match to match using shared variables, effectively passing sessions between matches.
2. Handles may be retained by a specific match so that it may switch from session to session.
3. POVISAVE may be used to create a handle for a POVI session not started using POVILGON so that it can be restored by POVIRSTR and subsequently used by POVI REXX functions.

POVISAVE()

Returns a handle for either the current POVI session that was started using POVILGON, or for the specified session that was started using the native LOGON command.

This handle can be used as a parameter with the POVIRSTR function to restore the session. Once the session has been restored in this way, it may be used by the POVI REXX functions.

Syntax

POVISAVE(env_name model#, {trace})

env_name Environment name for a session that was started using the native LOGON command.
model# Model number for a session started using the native LOGON command.
trace Turns on internal trace of primitive functions.

Function Codes

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Handle could not be created.</td>
</tr>
<tr>
<td>&lt;&gt;0</td>
<td>Current session environment has been de-addressed but can be restored using POVIRSTR and the returned handle.</td>
</tr>
</tbody>
</table>
Comments
1. If the currently addressed environment was not created using the POVILGON REXX function, then env_name and model# must be specified. In this case, if these parameters are specified a result of 0 will be returned.
2. When the current environment was established using POVILGON, the parameters should be omitted.
3. POVISAVE will de-address the current environment and return a handle which can be used by POVIIRSTR to reestablish the session environment so that it may be used by POVI REXX functions.
4. Handles created by POVISAVE may be used by a single match to switch between two or more sessions.
5. Handles created by POVISAVE may be passed to other matches via shared variables, allowing sessions to be effectively passed between matches.
6. Handles creating using POVISAVE may be created for sessions that were started using the native LOGON command (and therefore unusable by POVI REXX functions), then used by POVIIRSTR to restore the session environment so that it can be used by the POVI REXX functions. This approach can be used so that the superior RAS of the POVI REXX functions can be extended to existing POVI automation without re-writing the entire automation application.

POVISEND()

Writes data to the current cursor position and optionally simulates an Attention Identifier (AID) key or function key. Use the AID key option to verify the contents of the screen.

The environment must have been created using the POVILGON function.

Syntax

POVISEND(data, key, {waitdata, {waitrow, waitcol}}, CONFIDENTIAL, {trace})

data

The data to be sent. This argument may contain a null character only if the key argument contains a non-null character.
**POVI Functions**

**key**
A key to be simulated. This argument may contain a null character only if the data argument contains a non-null character. Valid keys include AID keys, which send data to the application, and local function keys, which do not send data to the application. The valid AID keys are:
- ENTER
- CLEAR
- PA1 to PA3
- PF1 to PF24
The valid local function keys are:
- HOME
- UP
- DOWN
- LEFT
- RIGHT
- TAB
- BACKTAB
- ERASEEOF
- ERASEINP

You cannot specify the following parameters, waitdata, waitrow, and waitcol, unless you have specified an AID key. These parameters are used to verify the screen contents, which can help you detect situations that could cause your automation to fail.

**waitdata**  Data to be searched for at the row and column specified in waitrow and waitcol. The data search occurs on the screen returned after a key has been simulated. When waitrow and waitcol are null, the entire screen will be searched.

**waitrow**  The row where you want to search for the waitdata. When waitrow contains a value, then waitcol and waitdata must contain a value.

**waitcol**  The column where you want to search for the waitdata. When waitcol contains a value, then waitrow and waitdata must contain a value.

**CONFIDENTIAL**
Causes the POVI trace facilities to not trace the data. Use this if you want to prevent the tracing of userids and passwords.

**trace**  Turns on internal trace of primitive functions.
Function Codes

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion. Data or key was sent to the VTAM 3270 application. If the waitdata, waitrow, and waitcol were established, then the function ended successfully and the data was found in the specified location.</td>
</tr>
<tr>
<td>5</td>
<td>Invalid argument. The argument name was pushed onto the data stack.</td>
</tr>
<tr>
<td>6</td>
<td>A required argument is missing. The argument name is pushed onto the data stack.</td>
</tr>
<tr>
<td>9</td>
<td>No POVILGON environment active.</td>
</tr>
<tr>
<td>11</td>
<td>CT/Engine error message returned. The message is pushed onto the data stack.</td>
</tr>
<tr>
<td>12</td>
<td>Terminal input error detected.</td>
</tr>
<tr>
<td>13</td>
<td>An internal invocation of the POVIDATA function failed. The POVIDATA function code is pushed onto the data stack.</td>
</tr>
<tr>
<td>14</td>
<td>An internal invocation of the POVIFIND function failed. The POVIFIND function code is pushed onto the data stack.</td>
</tr>
<tr>
<td>22</td>
<td>Waitdata not found as specified by waitrow and waitcol.</td>
</tr>
<tr>
<td>23</td>
<td>Data transmission error. See the AF/OPERATOR log.</td>
</tr>
</tbody>
</table>

Comments

1. Function code 22 may indicate one of the following:
   - A programming error resulting in data being sent to a protected screen location. The TERMINAL INPUT ERROR message is returned from CT/Engine.
   - Unexpected application behavior, such as asynchronous messages sent to the terminal, restoration messages for archived datasets, and maintenance to the application caused differences in screen appearance.

2. Searching is case sensitive.
3. If no stacked data is described for a non-zero function code, then a null value is pushed onto the data stack by default.
Examples
The following example uses POVISEND to send the userid for a TSO logon and to verify that the password prompt was returned:

```plaintext
toolsrc = POVILGON(OVIAO, TSOA, TSOA, 3)
IF (toolsrc <> 0) THEN CALL ERRHNDLR
toolsrc = POVISEND(TOP, "ENTER LOGON ID - ", 0)
IF (toolsrc<>0) THEN CALL ERRHNDLR
 toolsrc=POVISEND("USER012",ENTER,"ENTER PASSWORD - ")
IF (toolsrc<>0) THEN CALL ERRHNDLR
```

POIVSET()
Sets a CT/Engine variable to a specified value.

Syntax

```
POIVSET(vblname, value, {trace})
```

- **vblname** The name of the variable you want to set.
- **value** The new value of the variable.
- **trace** Turns on internal trace of primitive functions.

Function Codes

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The variable was successfully set.</td>
</tr>
<tr>
<td>9</td>
<td>No POVILOGON environment is active.</td>
</tr>
<tr>
<td>23</td>
<td>Data transmission error. See the AF/OPERATOR log.</td>
</tr>
<tr>
<td>25</td>
<td>GLBVGET error. The variable RESULT is pushed onto the data stack.</td>
</tr>
</tbody>
</table>

Comments

1. POIVSET can be used to set CT/Engine variables which will be referenced in POVI Keystroke Recorder scripts.
2. POIVSET can be used to set variables that control CT/Engine options for the current session only. Many of these options can be set globally for all sessions but POIVSET provides more granular control of the options.
3. When you need to set variables to be substituted in Keystroke Recorder scripts, use the variable name PASSWORD to inhibit tracing of the variable’s new value.
4. If no stacked data is described for a non-zero function code, then a null value is pushed onto the stack by default.
The REDIRECT function acts on the output of all AF/OPERATOR commands and functions which return data in variables, allowing you to redirect the output data to REXX variables or to the REXX data stack.

Purpose
The REDIRECT function supports enhanced control of the variables returned by AF/OPERATOR-supplied REXX functions, such as IMSCMD, which return data in REXX variables by default. The REDIRECT function can also redirect the output of AF/OPERATOR commands which normally return data in AF/OPERATOR line variables, to REXX variables, are:

- The maximum number of variables that can be returned is increased. REXX variables requested using REDIRECT return all available variables, by default. AF/OPERATOR line variables return a maximum of 999 lines of data.
- No data alteration occurs. Contiguous blanks that may appear in the data are preserved instead of being compressed into a single blank character. AF/OPERATOR line variables compress multiple blank characters into one.
- Simplification of your REXX execs. Data in REXX variables is accessible to the REXX exec without further processing. Data in AF/OPERATOR line variables must first be retrieved, using the GLBVGET function, then accessed.

REDIRECT()

Description
Redirects the output of subsequent AF/OPERATOR commands and AF/OPERATOR-supplied REXX functions which return data in variables, into REXX variables whose names you can specify or places this data on the REXX data stack.

Syntax

```plaintext
>> REDIRECT ( OFF varname STACK max CONCAT ) <<
```
null

Specify REDIRECT with a null parameter, as follows:

REDIRECT()

to return the name of the variable that has been specified with a previous call to the REDIRECT function. If REDIRECT is not in effect, the word OFF is returned.

OFF

This reserved keyword disables a previous call to the REDIRECT function. The output of subsequent AF/OPERATOR commands and AF/OPERATOR-supplied REXX functions is returned in the default variable type and name associated with each command or function. For example, the AF/OPERATOR OPER command returns data in AF/OPERATOR line variables whose default names are line1, line2, and so on. The AF/OPERATOR-supplied IMSCMD function returns data in REXX compound variables whose default names are imscmd.1, imscmd.2, and so on.

varname

Specify the name to be assigned to variables that will hold output data returned by subsequent AF/OPERATOR commands and AF/OPERATOR-supplied REXX functions. The variable name must be a valid REXX variable name. It can be from 1-242 characters in length. Output can be redirected to either REXX compound or REXX prefixed variables depending on how you specify the name:

- If the name you specify ends with a period (.), the name becomes the stem of a REXX compound variable. Lines of output are returned in consecutively numbered REXX compound variables of the form varname.1, varname.2, and so on. The total number of lines returned is stored in varname0.

- If the name you specify does not end with a period, the name becomes the prefix of a REXX prefixed variable. Lines of output are returned in consecutively numbered REXX prefixed variables of the form varname1, varname.2, and so on. The total number of lines returned is stored in varname0.

STACK

This reserved keyword places redirected output on the REXX data stack instead of in REXX variables.

max

You can adjust the maximum number of REXX variables, returned by subsequent AF/OPERATOR commands and AF/OPERATOR-supplied REXX functions by specifying one of the following:

- An asterisk in quotation marks:

      "*"

specifies that all available lines of output are returned. If a max specification is omitted, or codes as "*", up to 999999999 lines of data are returned.

- A positive whole number(from 1–999,999,999), specifies the maximum number of output lines to return. The rest are ignored. The specification is useful if you are looking for data that occurs on a particular line of output, for example, a line near the beginning of a command’s output display summarizing the lines that follow.

CONCAT

Concatenates redirected output and returns it as a single variable or REXX data stack entry. For example, a function that otherwise returns 25 variables as data.0, data.1, ..., data.25, returns only data.0 (containing a count of 1) and data.1, when CONCAT is specified. This specification is efficient when your objective is to search the returned output for a single string.
Comments

- The REDIRECT function can be invoked from a REXX exec using standard REXX syntax. Place the REDIRECT function in the exec before the AF/OPERATOR command or function whose output is to be redirected.

- Redirection is maintained at an exec level. Each REXX exec starts with redirection disabled, even if it was invoked from another REXX exec that had changed the redirection status. When an exec ends, the redirection status of its caller is restored.

- One you enable output redirection with a REDIRECT function call, the specification remains in effect until one of the following occurs:
  - The exec containing the REDIRECT function call ends. Output redirection is only valid in the exec in which it occurs.
  - A subsequent REDIRECT function call specifies REDIRECT(OFF).
  - A subsequent REDIRECT function call specifies a different REXX variable name.

- The REDIRECT function returns its current value or status as a character string before setting a new value. For example:
  - If you specify a null value and output redirection is in effect, the function returns the current redirection variable name.
  - If you specify a null value and output redirection is not in effect, the function returns the word OFF.
  - If you specify a value, the function returns the current redirection value (either a variable name or the word OFF), before implementing the value you specify.
Example of Processing

The following REXX exec shows how the REDIRECT function can simplify execs containing AF/OPERATOR commands which return data in variables. The first time OPER is issued, output redirection is not in effect. The OPER command returns data in AF/OPERATOR line variables prefixed with the name line. This is the default. The REDIRECT function call redirects the output of the OPER command to REXX stem variables named data.1, data.2, and so on. Note that the GLBVGET function call is not used when output redirection is in effect.

```rexx
/* REXX */
address afhost "OPER 'D A, L' RESP"
rc = glbvget('LINE#')
do i=1 to LINE # by 1
   rc = glbvget("LINE"| ||i)
end
if LINE# > 2 then do
   if pos("JES2", LINE2) > 0 then do
      say "JES2 is up"
   end
end
rc = REDIRECT("data.", "*")
address afhost "OPER 'D A, L' RESP"
if data.0 > 2 then do
   if pos("JES2", data.2) > 0 then do
      say "JES2 is up"
   end
end
```

The following REXX exec fragment shows how the REDIRECT function can be used with another Candle-supplied REXX function, IMSCMD. When output redirection is not in effect, the IMSCMD function returns data in REXX stem variables names imscmd. This is the default. The REDIRECT function call, using the CONCAT option, redirects the output of the IMSCMD function call to a single REXX stem variable named output.1. This single variable is then searched for the string A123. This is an efficient way to search output for a single string, eliminating a loop of code.
RETURN Codes

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Completed successfully.</td>
</tr>
<tr>
<td>04</td>
<td>Invalid function syntax.</td>
</tr>
<tr>
<td>08</td>
<td>Invalid environment.</td>
</tr>
<tr>
<td>16</td>
<td>Internal logic error.</td>
</tr>
</tbody>
</table>

/* REXX */
.
.
.
rc = IMSCMD("IMSPROD", "SEND", "/DIS TERM ALL")
do i = 1 to imscmd.0
   if pos("A123", imscmd.1) > 0 then say "found it"
end
rc = REDIRECT("output.", "***", CONCAT)
rc = IMSCMD("IMSPROD", "SEND", "/DIS TERM ALL")
if pos("A123", output.1) > 0 then say "found it"
.
."
System Functions

ACTIVE(procname.taskname)

This function accepts the name of a procedure (an MVS address space) followed optionally by a taskname. If you want to start the same procedure twice with each instance having a different taskname, then use the taskname parameter to specify a unique name. If the procedure is active, then the function returns 1 (true). Otherwise, the function returns 0 (false).

AFDISPAT

This function redispaces the AF/OPERATOR task scheduler, suspending the current match and allowing other matches to be processed. The current match will resume as soon as possible. This resumption will occur almost immediately if no other matches are ready to start or resume.

Syntax

rc=AFDISPAT("")

AFSHOW

The function allows any AF/OPERATOR SHOW command to be issued. Results are returned in a local REXX stem variable called AFSHOW.n, where AFSHOW.0 contains the number of variables returned in total and AFSHOW.1 through AFSHOW.n contain the results of the inquiry.

Syntax

rc=AFSHOW('SHOW command text')

Examples

/* REXX */
RC = AFSHOW('SHOW TRAP')
IF RC = 0 THEN DO
DO CT =1 TO AFSHOW.0
SAY AFSHOW.CT
END
END

/* REXX */
RC = AFSHOW('SHOW TRAP FORMAT("&&STATUS &&TYPE &&NAME")')
IF RC = 0 THEN DO
DO CT =1 TO AFSHOW.0
SAY AFSHOW.CT
END
END
**System Functions**

**ARM('request')**

This function permits you to enable or disable the LISTEN exit to monitor for automatic restart management (ARM) events without having to recycle AF/OPERATOR. You can also use the ARM function to issue the ARM READY request in the event that READY was deferred at product startup.

**Note:** To use this ARM REXX function's READY request, you must have previously specified ENABLED(ELEMENT(elementname)) on the ARM startup parameter. The element name is the name by which AF/OPERATOR is registered with ARM. ARM processing is available only when running under MVS/SP Version 5.2 or later.

- **REQUEST** Identifies the function requested, which can be one of the following:
  - **LISTEN** Starts monitoring for ARM events
  - **NOLISTEN** Stops monitoring for ARM events
  - **READY** Issues a call to ARM services to indicate that AF/OPERATOR is ready for work.

**Return codes**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Request processed successfully</td>
</tr>
<tr>
<td>8</td>
<td>Syntax error occurred</td>
</tr>
<tr>
<td>12</td>
<td>Minimum environment not available</td>
</tr>
<tr>
<td>16 - 24</td>
<td>Internal error has occurred</td>
</tr>
<tr>
<td>1nnn</td>
<td>Return code from ENFREQ or IXCARM + 1000</td>
</tr>
</tbody>
</table>

**CONSOLE(option,address)**

This function accepts an option string of D, M or U. If you specify M, the function returns the ID of the master console. If you specify D, then you must supply the unit address of the specified console. In this case, the function returns that console’s ID. U indicates the function should return the ID of the specified console, whether or not it is active. The address must be an alphanumeric string containing characters 0 to 9 and A to F. The address argument is only valid with option D or U. This function returns the ID of the specified console.
CPULIM()

Allows you to specify a CPU limit for a match. The value you specify overrides the default CPU limit set by the CPULIM startup parameter.

Syntax

```rexx
>>>-CPULIM ( seconds ) >><
```

seconds  Sets, in seconds, a new CPU limit for the match. You can specify from 1 to 32767 seconds.

Comments

REXX execs that use more than the default CPU limit set by the CPULIM startup parameter can use the CPULIM function to change their CPU limit to a higher value.

If CPULIM is coded without a `seconds` specification, it returns the current CPU limit for the match.

When you specify a new CPU limit using the CPULIM function:

- The new limit takes effect immediately.
- The function also returns the previous CPU limit for the match.
- The function creates a global match variable called AOCPULIM which contains the new limit for the match. The default CPU limit for all matches is already available as a system variable called AOCPULIM. These variables can be displayed using the SHOW VARS command.

Example

The following code fragment shows how the CPULIM function can be used.

```rexx
/* REXX */
.
.
.
REQUIRED_CPULIM_VALUE = 200
.
IF CPULIM() < REQUIRED_CPULIM_VALUE THEN DO
   OLD_CPULIM_VALUE = CPULIM(REQUIRED_CPULIM_VALUE)
   SAY "OLD CPULIM VALUE:" OLD_CPULIM_VALUE
   SAY "NEW CPULIM VALUE:" REQUIRED_CPULIM_VALUE
END
.
.
.
```
DEQUEUE Function

Description
DEQUEUE() releases an ENQ on a specified resource name.

Syntax

\[
>> \text{DEQUEUE}(qname, rname, )
\]

Note: The DEQUEUE function interface to the IBM DEQ services, and the input parameters are primarily the same as the input parameters for these services.

**qname** A 1-to-8 character identifier, used in conjunction with **rname**, to uniquely identify the resource being DEQUEUED.

**rname** A 1-to-255 character identifier, used in conjunction with **qname**, to uniquely identify the resource being DEQUEUED.

**scope** Defines the level of the ENQUEUE. Possible values are:

- **STEP** Limits the scope of the resource to one address space.
- **SYSTEM** Specifies that the scope is between address spaces on the same system.
- **SYSTEMS** Specifies that the scope can be between multiple systems.

Note: Only the owning match can dequeue a particular resource.

Examples
The following example dequeues the particular named resource, which is currently enqueued with STEP scope. The value of RET_CODE indicates the success or failure of the request:

```rexx
/*- REXX -*/
"RET_CODE=DEQUEUE(‘r-name’, ‘q-name’, ‘STEP’)"
```
Return Codes
The DEQUEUE function returns the following codes:

<table>
<thead>
<tr>
<th>Return code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal completion</td>
</tr>
<tr>
<td>8</td>
<td>Parameter list error</td>
</tr>
<tr>
<td>24</td>
<td>Internal error - missing SYSWRK</td>
</tr>
<tr>
<td>28</td>
<td>Internal error - missing ENVBLOCK</td>
</tr>
<tr>
<td>32</td>
<td>Internal error - Subtask terminated</td>
</tr>
<tr>
<td>36</td>
<td>Not APF-authorized. This code is returned if you attempt to invoke the DEQUEUE function in an OG<em>TSO environment. The DEQUEUE function is not unavailable under OG</em>TSO.</td>
</tr>
<tr>
<td>40</td>
<td>Internal error - Failed to attach subtask</td>
</tr>
<tr>
<td>2xx</td>
<td>DEQ return code, where xx is the non-zero return code from the IBM DEQ service.</td>
</tr>
<tr>
<td>208</td>
<td>Unable to DEQ, not enqueued</td>
</tr>
</tbody>
</table>

### ENQUEUE

**Description**
ENQUEUE() obtains an ENQ on a specified resource name or tests the availability to ENQ on a specified resource.

**Syntax**

```plaintext
=> ENQUEUE(qname, rname, STEP, EXCL, \time, \USE, \TEST, \CHNG)
```

**Note:** The ENQUEUE function interface to the IBM ENQ services, and the input parameters are primarily the same as the input parameters for these services.

- **qname** A 1-to-8 character identifier, used in conjunction with rname, to uniquely identify the resource being ENQUEUED.
- **rname** A 1-to-255 character identifier, used in conjunction with qname, uniquely identify the resource being ENQUEUED.
- **scope** Defines the level of the ENQUEUE. Possible values are:
Comments:

- Enqueues are obtained for the duration of a match. When the match terminates, each enqueue the match owns is released and each enqueue the match is still waiting for is canceled.

- When enqueuing on multiple resources in different tasks, always enqueue the individual resources in the same sequence to avoid each task locking out the other (deadly embrace). For example, assume the two tasks are executing at the same time, and that both tasks would like to enqueue on resources A and B. If both tasks request enqueues for A followed by B, the situation is avoided where task1 owns resource A but is requesting and waiting for resource B, while task2 owns resource B and is requesting and waiting for resource A.

Examples

The following example obtains an EXCLUSIVE enqueue with STEP scope against the particular named resource. If the resource is not immediately available, the match will wait until it becomes available. The value of RET_CODE identifies the success or failure of the request:
The following example obtains an EXCLUSIVE enqueue with SYSTEM scope against the particular named resource. If the resource is not immediately available, the match will wait for a maximum of 10 seconds before abandoning the request. The value of RET_CODE indicates the success or failure (including timeout) of the request:

```rexx
/* REXX */
"RET_CODE=ENQUEUE(‘r-name’,’q-name’,’STEP’,’EXCLUSIVE’)"
```

The following example determines if a SHARED enqueue with SYSTEMS scope against the particular named resource can be obtained. If the enqueue cannot be obtained, the value of RET_CODE will indicate the availability of the resource:

```rexx
/*- REXX */
"RET_CODE=ENQUEUE(‘r-name’,’q-name’,’SYSTEM’,’EXCL’,’WAIT’,’10’)"
```

### Return Codes

Execution of the ENQUEUE function returns the following codes:

<table>
<thead>
<tr>
<th>Return code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Parameter list error</td>
</tr>
<tr>
<td>24</td>
<td>Internal error - missing SYSWRK</td>
</tr>
<tr>
<td>28</td>
<td>Internal error - missing ENVBLOCK</td>
</tr>
<tr>
<td>32</td>
<td>Internal error - Subtask terminated</td>
</tr>
<tr>
<td>36</td>
<td>Not APF-authorized. This code is returned if you attempt to invoke the ENQUEUE function in an OG<em>TSO environment. The ENQUEUE function is not available under OG</em>TSO.</td>
</tr>
<tr>
<td>40</td>
<td>Internal error - Failed to attach subtask</td>
</tr>
<tr>
<td>2xx</td>
<td>ENQ return code, where xx is the non-zero return code from the IBM ENQ service.</td>
</tr>
</tbody>
</table>
Return codes specific to the WAIT option are:

<table>
<thead>
<tr>
<th>Return code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal completion - enqueue obtained</td>
</tr>
<tr>
<td>100</td>
<td>Enqueue timeout expired</td>
</tr>
<tr>
<td>104</td>
<td>Internal error - Subtask terminated</td>
</tr>
<tr>
<td>208</td>
<td>This match already owns the enqueue</td>
</tr>
</tbody>
</table>

Return codes specific to the USE option are:

<table>
<thead>
<tr>
<th>Return code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal completion - enqueue is available</td>
</tr>
<tr>
<td>204</td>
<td>Resource is enqueued elsewhere</td>
</tr>
<tr>
<td>208</td>
<td>This match already owns the enqueue</td>
</tr>
</tbody>
</table>

Return codes specific to the CHNG option are:

<table>
<thead>
<tr>
<th>Return code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal completion - ‘control’ changed to EXCL</td>
</tr>
<tr>
<td>204</td>
<td>Cannot change, resource being shared with other tasks</td>
</tr>
<tr>
<td>208</td>
<td>Cannot change, already enqueued</td>
</tr>
</tbody>
</table>

**JOBSTEP(address-space-name)**

This function accepts the name of an MVS address space. If the address space is active, then the function returns the name of the currently executing job step. Otherwise, a null string is returned.

**JOURNAL**

Use the JOURNAL REXX function if you want to journal your own data rather than the matched data from a trap. When invoking JOURNAL, you can specify a string of up to 128 characters to write on the journal.

In the REXX EXEC, you can code:

```rc = JOURNAL('text','color','highlight','intensity','help')```

where:

- **text** Text for the SLF journal.
- **color** Optional color attribute. Valid values are blue, red, pink, green, turq, yellow, and white.
- **highlight** Optional highlight attribute. Valid values are none, blink, rvideo, and underline.
- **intensity** Optional intensity attribute. Valid values are normal and high.
- **help** Optional help panel name.
The return code \( rc \) from the REXX function may be:

<table>
<thead>
<tr>
<th>Return code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Completed successfully</td>
</tr>
<tr>
<td>8</td>
<td>Syntax error</td>
</tr>
<tr>
<td>12</td>
<td>SLF not active</td>
</tr>
<tr>
<td>16</td>
<td>Internal error</td>
</tr>
</tbody>
</table>

**LOG(ON|OFF|ACTIVE)**

This function lets you activate or deactivate the logging of a match. The initial logging status of a match is determined by the AF LOG/NOLOG setting and by the LOG/NOLOG options of the TRAP ADD command. You can dynamically activate message logging from a REXX EXEC using the LOG() function even if the originating TRAP carried the NOLOG attribute.

When LOG(ON) or LOG(OFF) is issued, the function returns 1 (true) if the operation was successful. Otherwise, the function returns 0 (false). When you execute the LOG without a keyword, the function returns 1 (true) if logging is active for the match. If logging is not active for the match, the function returns 0 (false). When you execute LOG with the ACTIVE operand, LOG(‘ACTIVE’), the function returns ON if logging is active for the match. If logging is not active for the match, the function returns OFF.

**LOGMSG(text,OGMSGL/USERLG)**

This function accepts a character string up to 480 characters. LOGMSG writes a line of text from a REXX exec to the RKOGLOGM (unless the optional second parameter is specified) without impacting the external environment.

The second parameter specifies a log file to which the message is to be written. If this parameter is omitted, the value defaults to OGMSGL, which directs the message to the OG message log, RKOGLOGM. If USER is specified, the message is directed to the current user DASD message log, USERLG##, if there is one. If there is no USERLG##, the message is directed to the OG message log. This function always writes to a log unless logging has been suppressed globally by the AF NOLOG command.

When the text is successfully written, the function returns 1 (true). Otherwise, the function returns 0 (false).

**MATCH(text,pattern)**

This function accepts a text string and a pattern string, each of which is 0 to 254 characters in length. The text string may contain any EBCDIC character. The pattern string may contain any pattern characters allowed by the AF/OPERATOR pattern matching facility (see “Pattern Matching” on page 83). The function returns 1 (true) if the text string matches the pattern string according to the rules of AF/OPERATOR pattern matching. Otherwise, the function returns 0 (false). If the text string is null, then only a pattern of * produces a match. If the pattern string is null, then 0 (false) will always be returned.
System Functions

MSGLIM()

Description
The purpose of this function is to permit the message limit to be set dynamically on a per match basis. The new limit takes effect immediately. This means that the default limit set by the MSGLIM initialization parameter can be set to a lower, more reasonable value. Those REXX execs that use more than the default can use the MSGLIM function to change the value.

The default value is available as a system variable called AOMSGLIM. If the MSGLIM function is called to set a new message limit, it also creates a global match variable called AOMSGLIM that contains the new limit. You can display these variables using the SHOW command.

Syntax

```
rc=MSGLIM COUNT
```

none When coded without a parameter, MSGLIM returns the current message limit for the match.

COUNT Specifies the new limit on the number of messages this match can issue. The minimum value is 10 and the maximum value is 32767, which is the same as the MSGLIM initialization keyword. Specifying 0 turns off message counting.

In addition to setting the new message limit, the MSGLIM function also returns the previous message limit for the match.

Caution: Use caution when specifying 0 to turn off message counting.

PLXQUERY()

Description
This function extracts information from XCF about a specified Sysplex resource. The results are returned in a local REXX stem variable called PLXQUERY.n, where PLXQUERY.0 contains the number of variables returned in total and PLXQUERY.1 through PLXQUERY.n contain the results of the inquiry.

The variable name, PLXQUERY, can be altered using the Candle-supplied REDIRECT REXX function.
Syntax

```
rc=PLXQUERY('ARMSTATUS', elementname)
   | 'GROUP', groupname, membername
   | 'SYSPLEX')
```

**ARMSTATUS**

(or ARMS). Extract details relating to ARM elements.

When used without further ARM element name qualification or when the special `elementname` value, ARMS_ALLDATA, is used, details relating to all ARM elements on the Sysplex are returned.

**elementname**

Optional ARM element name.

If you optionally qualify ARMSTATUS with an ARM element name, details relating to the named individual ARM element are returned.

**GROUP**

(or GRP). Extract details relating to XCF groups and members.

When used without further XCF group name qualification or when the special `groupname` value, NO_GRPNAME, is used, details relating to all XCF groups on the Sysplex are returned.

**groupname**

Optional XCF Group name.

If GROUP is qualified only by an XCF group name, details relating to all individual members of that specific group are returned.

**membername**

Optional XCF member name.

If GROUP is fully qualified to the member level, details relating to the specific member within the specific group are returned.

**SYSPLEX**

Extract details relating to systems in the Sysplex.
**Format of the returned information**

The returned data contained in REXX variables PLXQUERY.1 through PLXQUERY.n has the following format according to the type of data that is requested.

**If ARM element information is returned:**

<table>
<thead>
<tr>
<th>Columns</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-16</td>
<td>ARM element name</td>
</tr>
<tr>
<td>18-25</td>
<td>System where element was initially registered</td>
</tr>
<tr>
<td>27-34</td>
<td>System where element is currently registered</td>
</tr>
<tr>
<td>36-45</td>
<td>Element status. Value can be: AVAILABLE, STARTING, FAILED, RESTARTING, RECOVERING, or UNKNOWN.</td>
</tr>
<tr>
<td>47-62</td>
<td>Restart group</td>
</tr>
<tr>
<td>64-71</td>
<td>Job name</td>
</tr>
<tr>
<td>73-80</td>
<td>Element type (If there is no element type, field contains spaces.)</td>
</tr>
<tr>
<td>82-85</td>
<td>ASID (Hexadecimal address space ID)</td>
</tr>
<tr>
<td>87-91</td>
<td>Level (Leading spaces are used.)</td>
</tr>
<tr>
<td>93-95</td>
<td>Total number of restarts since initially registered. (Leading spaces used.)</td>
</tr>
<tr>
<td>97-106</td>
<td>Date when initially registered in MM/DD/YYYY format</td>
</tr>
<tr>
<td>108-115</td>
<td>Time when initially registered in HH:MM:SS format</td>
</tr>
<tr>
<td>117-126</td>
<td>Date of the first restart (or spaces if never restarted) in MM/DD/YYYY format.</td>
</tr>
<tr>
<td>128-135</td>
<td>Time of the first restart (or spaces if never restarted) in HH:MM:SS format</td>
</tr>
<tr>
<td>137-146</td>
<td>Date of the latest restart (or spaces if never restarted) in MM/DD/YYYY format.</td>
</tr>
<tr>
<td>148-155</td>
<td>Time of the latest restart (or spaces if never restarted) in HH:MM:SS format</td>
</tr>
</tbody>
</table>

**If XCF group information is returned (for example, no groupname was specified):**

<table>
<thead>
<tr>
<th>Columns</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>XCF group name</td>
</tr>
<tr>
<td>10-13</td>
<td>Count of members in this group (Leading spaces used.)</td>
</tr>
</tbody>
</table>
If XCF member information is returned:

<table>
<thead>
<tr>
<th>Columns</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-16</td>
<td>XCF member name</td>
</tr>
<tr>
<td>18-25</td>
<td>Member status. The value can be CREATED, ACTIVE, QUIESCED, FAILED, or UNKNOWN.</td>
</tr>
<tr>
<td>27-34</td>
<td>System name</td>
</tr>
<tr>
<td>36-43</td>
<td>Job name</td>
</tr>
<tr>
<td>45-54</td>
<td>Date of last status change in MM/DD/YY format</td>
</tr>
<tr>
<td>56-63</td>
<td>Time of last status change in HH:MM:SS format</td>
</tr>
<tr>
<td>65-66</td>
<td>Length of the user state field that can be a 2-digit value from 0 - 32 bytes.</td>
</tr>
<tr>
<td>68-99</td>
<td>User state field (variable length field of maximum 32 bytes)</td>
</tr>
</tbody>
</table>

If Sysplex system information is returned:

<table>
<thead>
<tr>
<th>Columns</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-8</td>
<td>System name</td>
</tr>
<tr>
<td>10-17</td>
<td>Time of last status update in HH:MM:SS format</td>
</tr>
</tbody>
</table>

Examples

The following REXX example shows how to display all active XCF groups:

```rexx
/* REXX */
RC = PLXQUERY('GROUP')
IF RC = 0 THEN DO
    DO CT = 1 TO PLXQUERY.0
       SAY PLXQUERY.CT
    END
END
```

Example output:

```
SYSGRS    8
SYSGRS2   1
SYSGRS3   6
.
.
```

The following REXX example shows how to display all members within an active XCF group:
The following REXX example shows how to display details of a specific ARM element.

```rexx
/* REXX */
RC = PLXQUERY('GROUP','KOGABC123')
IF RC = 0 THEN DO
  DO CT = 1 TO PLXQUERY.0
    SAY PLXQUERY.CT
  END
END

AFXCSYSA     ACTIVE   SYSA  TASK0027 10/19/2000  01:50:24  24 XX..

/* REXX */
RC = PLXQUERY('ARMSTATUS','KOG_ARM123')
IF RC = 0 THEN DO
  DO CT = 1 TO PLXQUERY.0
    SAY PLXQUERY.CT
  END
END

KOG_ARM123  SYSG  SYSG   STARTING  DEFAULT  TASK1234  0096 2  0
10/19/2000  03:35:19
```

The following REXX example shows how to display systems within the Sysplex.

```rexx
/* REXX */
RC = PLXQUERY('SYSPLEX')
IF RC = 0 THEN DO
  DO CT = 1 TO PLXQUERY.0
    SAY PLXQUERY.CT
  END
END
```
Example output

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success. PLXQUERY.n variables are set.</td>
</tr>
<tr>
<td>4</td>
<td>Specified resource not found.</td>
</tr>
<tr>
<td>8</td>
<td>Parameter list error.</td>
</tr>
<tr>
<td>12</td>
<td>IXCQUERY error. IXCQUERY return and reason codes are set in PLXQUERY.n variables.</td>
</tr>
<tr>
<td>16</td>
<td>Severe internal error.</td>
</tr>
<tr>
<td>20</td>
<td>Reserved.</td>
</tr>
<tr>
<td>24</td>
<td>Internal error - $RXVARS.</td>
</tr>
<tr>
<td>28</td>
<td>Internal error - Unrecognized format in IXCQUERY answer area.</td>
</tr>
<tr>
<td>32</td>
<td>Internal error - Answer area GETMAIN retry limit hit.</td>
</tr>
<tr>
<td>92</td>
<td>Internal error - Bad $ESTAE setup.</td>
</tr>
<tr>
<td>96</td>
<td>Internal error - $ESTAE invoked.</td>
</tr>
</tbody>
</table>

When return code 12 is set, the IXCQUERY return code and reason code are set in the PLXQUERY.1 variable text. The format of the variable text is:

**IXCQUERY ERROR RETCODE=XXXXXXXX, RSNCODE=XXXXXXXX**

where XXXXXXXXX is the hexadecimal representation of the codes from IXCQUERY.

A complete list of IXCQUERY return codes can be found in IBM’s *Sysplex Services Reference*, GC28-1772.
PPRCCMD

Allows certain peer-to-peer remote copy commands to be issued.

Syntax

```c
rc=PPRCCMD('FREEZE', DEV-, P_SSID, P_SER-------, SSID, S_SER )
```

Parameters

- **FREEZE** Stop all updates to the secondary volume of this volume pair.
- **RUN** Resume updates to the secondary volume of this volume pair.
- **ESTPAIR** Establish a primary and secondary volume pairing.
- **DELPAIR** Remove a primary and secondary volume pairing.
- **SUSPEND** Suspend operations between primary and secondary volumes
- **RECOVER** Force the secondary volume into a simplex state
- **QUERY** Query the PPRC status of one volume of a pair.

NO The default, indicates that paths are not to be displayed. If omitted, NO is assumed.

YES Indicates that paths are to be displayed.
System Functions

Candle-Supplied REXX Functions 157

Return codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal completion</td>
</tr>
<tr>
<td>16</td>
<td>Internal error</td>
</tr>
<tr>
<td>Other</td>
<td>Directly from the IBM ANTRQST service</td>
</tr>
</tbody>
</table>

Special ANTRQST Error Codes

The local REXX stem variables called PPRCCMD.n are set by the PPRCCMD function. PPRCCMD.0 contains the number of variables returned in total and PPRCCMD.1 through PPRCCMD.n contain the additional information.

The stem variables are set under the following circumstances:

- When a ‘QUERY’ request is successful, the local REXX stem variables called PPRCCMD.n are set and contain the results of the query.
- When any return code greater than 16 is set, the local REXX stem variables called PPRCCMD.n are set and contain additional diagnostic information. The PPRCCMD.1 variable generally contains text containing the ANTRQST return and reason codes, except when one of the special ANTRQST error codes is returned (see the table below).

The variable name PPRCCMD can be altered using the Candle-supplied REDIRECT REXX function.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>7140</td>
<td>PDELPAIR error. An ‘ANTP0NNNNN’ message is returned in the stem variable.</td>
</tr>
<tr>
<td>7180</td>
<td>PSUSPEND error. An ‘ANTP0NNNNN’ message is returned in the stem variable.</td>
</tr>
<tr>
<td>7200</td>
<td>PESTPAIR error. An ‘ANTP0NNNNN’ message is returned in the stem variable.</td>
</tr>
<tr>
<td>7220</td>
<td>PRECOVER error. An ‘ANTP0NNNNN’ message is returned in the stem variable.</td>
</tr>
<tr>
<td>7240</td>
<td>PQUERY error. An ‘ANTP0NNNNN’ message is returned in the stem variable.</td>
</tr>
<tr>
<td>7260</td>
<td>PRUN error. An ‘ANTP0NNNNN’ message is returned in the stem variable.</td>
</tr>
<tr>
<td>7280</td>
<td>PFREEZE error. An ‘ANTP0NNNNN’ message is returned in the stem variable.</td>
</tr>
</tbody>
</table>

For other codes, the variable contents follows the format below:

`ANTRQST ERROR RETCODE=XXXXXXX, RSNCODE=XXXXXXX.`

where XXXXXXXX is the hexadecimal representation of the codes from ANTRQST. The complete list of ANTRQST return codes are found in IBM’s DFSMSdfp Advanced Services, SC26-4921.
REPID(address-space-name)

This function returns a character string equal to the reply ID of the first outstanding WTOR for
the job identified by its argument. The reply ID can be 2–4 characters long (in MVS releases
prior to ESA 4.1, the reply ID is always 2 characters long). If the job is not active in the system
or does not have an outstanding WTOR, REPID() returns a null.

REPINFO()

The REPINFO function provides detailed information about outstanding WTORs.
Returns detailed information about all outstanding WTORs for the job identified by the first
argument, or for all jobs if an asterisk is specified as the first argument.

Syntax

>> ___ REPINFO(ADDRESS_SPACE_NAME, STEM) _____________ >>

Parameters

ADDRESS_SPACE_NAME

Name of the job for which information about outstanding WTORs is to be returned.
This parameter is required and must be either a valid jobname or an asterisk (*) to
indicate that all outstanding WTORs for all jobs are to be returned.

STEM.

Root name of the REXX compound variable that contains the detailed information
about the outstanding WTORs.

Note: The ending period is required to denote a stem.
This parameter is optional. If specified, it must be a valid REXX variable name from 1
through 230 characters in length. If STEM. is not specified, the default name,
AOREPINFO. is used. Assuming that the default name, AOREPINFO., is used, the
function returns detailed information in the following REXX compound variables.

AOREPINFO.@REPLYID.1 ... NN
The 2-4 character reply ID of the outstanding WTOR. (In MVS
releases prior to ESA 4.1, the reply ID is always two characters long.)

AOREPINFO.@JOBNAME.1 ... NN
The 1-8 character address space name of the job that issued the
WTOR.

AOREPINFO.@JOBID.1 ... NN
The 8-character job number of the job that issued the WTOR.

AOREPINFO.@SYSID.1 ... NN
The 1-8 character system name of the system on which the WTOR
originated..

AOREPINFO.@MSGTEXT.1 ... NN
The text of the WTOR message. The text can be up to 128 characters
in length.

AOREPINFO.@TIME.1 ... NN
The 8-character timestamp that indicates when the WTOR was
issued.

AOREPINFO.@DATE.1 ... NN
The 5-character Julian date that indicates when the WTOR was
issued.
Comments

- Each outstanding WTOR selected generates one of each of the classes of variables described above. Thus, if the specified job has four outstanding WTORS, then four sets of variables are created and numbered from 1 through 4.
- Additionally, the variable AOREPINFO.0 is created to contain the number of sets of variables. In the example above, AOREPINFO.0 would contain the value 4.

Examples

The following example returns information for all outstanding WTORS for job IMSPROD in the default REXX stem variables AOREPINFO.

```
MY_RC = REPINFO("IMSPROD")
```

The following example returns information for all outstanding WTORS for all jobs in the REXX stem variable Current_Outstanding_WTORS.

```
MY_RC = REPINFO("*", "CURRENT_OUTSTANDING_WTORS.")
```

Return codes

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Completed successfully.</td>
</tr>
<tr>
<td>04</td>
<td>No WTORS were selected.</td>
</tr>
<tr>
<td>08</td>
<td>No WTORS were outstanding.</td>
</tr>
<tr>
<td>12</td>
<td>Invalid function arguments.</td>
</tr>
<tr>
<td>16</td>
<td>Invalid root name was specified.</td>
</tr>
<tr>
<td>20</td>
<td>Invalid variable name was specified.</td>
</tr>
<tr>
<td>24</td>
<td>Error trying to create variables.</td>
</tr>
<tr>
<td>28</td>
<td>Invalid REXX environment.</td>
</tr>
<tr>
<td>32</td>
<td>Invalid AF/Operator environment.</td>
</tr>
<tr>
<td>36</td>
<td>Task is not running in authorized mode.</td>
</tr>
<tr>
<td>40</td>
<td>Invalid communication task.</td>
</tr>
<tr>
<td>92</td>
<td>Error trying to establish ESTAE.</td>
</tr>
<tr>
<td>96</td>
<td>ESTAE encountered an error.</td>
</tr>
<tr>
<td>&gt;256</td>
<td>Internal logic error.</td>
</tr>
<tr>
<td>Negative</td>
<td>Internal logic error.</td>
</tr>
</tbody>
</table>
RSTORAGE(address,length,data,input_format,output_format)

address  A virtual storage address of the data to be retrieved. The address argument must be an alphanumeric string containing characters 0 to 9 and A to F that represent an MVS virtual address in hexadecimal. This is an EBCDIC representation of a hexadecimal storage address.

length  An integer indicating the number of bytes to be retrieved from the specified storage address. The length argument should be a positive integer. If no length is specified, the length of the data returned is one byte long.

data  Optional. You can supply data that will be used to overwrite storage. If data is supplied, after retrieving the information from the specified storage address, RSTORAGE will overwrite the storage contents starting at location address.

input_format  Optional. Can be the keyword DECIMAL or a truncation of the keyword DECIMAL. Enclose the character or characters in quotes. If input_format is specified then the address parameter is changed as follows: address specifies an EBCDIC representation of a decimal storage address.

output_format  Optional. Can be the keyword DECIMAL or a truncation of the keyword DECIMAL. Enclose the character or characters in quotes. If output_format is specified then the length parameter and return variable are changed as follows: length can specify a number in the range 1 to 4. The return variable will be the EBCDIC representation of a decimal number. If no length parameter is specified, the length of the data used to generate the return value is four bytes long.

The function returns the contents of storage in the current address space, beginning at the specified address for the length indicated by the length argument. If a length of zero is specified, then a null result is returned. If the specified address causes an addressing exception or storage fetch violation, or is not a valid address within the system, then a null character string is returned.

The input_format and output_format parameters are intended for use where addresses are being manipulated. They avoid the need to use additional REXX code (the C2D function) to convert the address to and from decimal format.

RSTORAGE is a Candle-supplied REXX function; STORAGE is a TSO REXX function. If you alter storage by using the RSTORAGE or STORAGE functions, messages AOP0270 and AOP0271 are written to the message log. These messages provide an audit trail indicating the address and length of the alteration.
SYMVGGET(input_varname, output_varname)

This function performs system symbol substitution on a text string and sets the resultant value in a local REXX variable. If execution was successful, the output variable contains the result after symbol substitution. The format is

\[
\text{return code} = \text{SYMVGGET('input\_varname', 'output\_varname')}
\]

where:

input_varname

Name of a local REXX variable that contains the source text for system symbols substitution. The source text can be a maximum of 1,024 bytes long.

output_varname

The name of a local REXX variable that will be set to contain the resultant text after system symbols substitution. The resultant text can be a maximum of 1,024 bytes long. If the text exceeds 1,024 bytes after substitution occurs, the result is truncated and a return code 4 is set.

Note: In the case of the variables above, the variable name is passed, not the value.

\[
\begin{align*}
\text{INPVAR} & = '\&SYSNAME.' \\
\text{RET} & = \text{SYMVGGET('INPVAR', 'OUTVAR')} \\
\text{WTO 'TEST - &SYSNAME.= OUTVAR "'"
\end{align*}
\]

Return codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful execution. Output variable contains the result after symbol substitution</td>
</tr>
<tr>
<td>4</td>
<td>Truncation occurred during symbol substitution. The result is truncated at the maximum length of 1,024 bytes.</td>
</tr>
<tr>
<td>8</td>
<td>Input variable was not found.</td>
</tr>
<tr>
<td>16</td>
<td>Parameter syntax error occurred.</td>
</tr>
<tr>
<td>20</td>
<td>Input text exceeded the maximum length permitted.</td>
</tr>
<tr>
<td>32</td>
<td>Internal error occurred.</td>
</tr>
<tr>
<td>64</td>
<td>Internal error occurred.</td>
</tr>
<tr>
<td>1xx</td>
<td>A non-zero return code 1xx occurred after an IRXEXCOM FETCH operation.</td>
</tr>
<tr>
<td>2xx</td>
<td>A non-zero return code 2xx occurred after an IRXEXCOM STORE operation.</td>
</tr>
<tr>
<td>6xx</td>
<td>An internal error resulted in a non-zero return code 6xx from KOGSYSYM.</td>
</tr>
</tbody>
</table>
TESTBIT(string,bit-number)

This function accepts an arbitrary string and a bit position relative to zero. If the bit at the specified position of the string is ON, then 1 (true) is returned. Otherwise 0 (false) is returned. For a string of length $n$, bit-number must be in the range:

$$0 \leq \text{bit-number} \leq (8^n)-1$$

If bit-number is outside this range, a syntax error will result.

TRPQUERY

Description

Returns a list of the currently active matches that were created from a particular trap or returns the status of a trap.

The results are returned in a local REXX stem variable called ‘TRPQUERY’, where TRPQUERY.0 contains the number of variables returned in total, and TRPQUERY.1 through TRPQUERY.n contain the results of the inquiry.

The variable name TRPQUERY can be altered through the use of the Candle-supplied REDIRECT REXX function.

Syntax

```
rc=TRPQUERY(traptype,trapname ) =>
   ,MATCHES
   ,STATUS
```

Note: Only those matches created from the currently defined trap of the specified name will be returned. Any active matches created by a trap of the same name that has subsequently been deleted and redefined will not be returned.

- **traptype**: A valid AF trap type: WTO, CMD, TOD, XOM, XOC, XOI, XO2 or DOM.
- **trapname**: A valid 1- through 24-character trap name.
- **MATCHES**: Returns a list of the currently active matches that were created from a particular TRAP. (This is the default.)
- **STATUS**: Returns the status of a particular TRAP. The returned value can be ENABLED or DISABLED. Additionally, in the case of a TOD trap only, the value can be PENDING.
Examples

The following REXX example shows how to list the currently active matches that were created from a WTO trap called TRAP1.

```
/* REXX */
RC = TRPQUERY('WTO','TRAP1')
IF RC = 0 THEN DO
  DO CT = 1 TO TRPQUERY.0
    SAY TRPQUERY.CT
  END
END
```

Example output:

```
1273
471
459
9
```

The following REXX example shows how to extract the current status of a CMD TRAP called TRAP2.

```
/* REXX */
TRPNAME = 'TRAP2'
RC = TRPQUERY('CMD',TRPNAME,'STATUS')
IF RC = 0 THEN DO
  SAY 'CMD TRAP ' TRPNAME ' STATUS IS ''TRPQUERY.1'''
END
```

Example output:

```
CMD TRAP TRAP2 STATUS IS "ENABLED"
```
### Return Codes

Execution of the TRPQUERY function returns the following codes:

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success. TRPQUERY.n variables are set.</td>
</tr>
<tr>
<td>4</td>
<td>There were no active matches found that were created from this trap.</td>
</tr>
<tr>
<td>8</td>
<td>Specified trap not found.</td>
</tr>
<tr>
<td>16</td>
<td>Parameter list error.</td>
</tr>
<tr>
<td>24</td>
<td>Internal error - $RXVARS.</td>
</tr>
<tr>
<td>28</td>
<td>Severe internal error.</td>
</tr>
<tr>
<td>32</td>
<td>Severe internal error.</td>
</tr>
<tr>
<td>36</td>
<td>Severe internal error.</td>
</tr>
<tr>
<td>40</td>
<td>Severe internal error.</td>
</tr>
</tbody>
</table>
Variable Functions

Format
There are two ways in which REXX variable-handling functions can be issued:

\[ \text{RET} = \text{GLBVGET('varname')} \]

or

\[ \text{RET} = \text{`GLBVGET(`varname')} \]

The second method should result in faster interpretation because it bypasses the search for internal parameters. All other methods besides those stated above may produce inconsistent results. You may, however, see examples of both syntaxes in our manuals.

GLBVGET(var1,var2,...,var20,'NOTRUNC')
This function accepts a list of up to 20 names of AF/OPERATOR global variables. The function obtains the values of the variables and places them into REXXX variables of the same name. If the REXXX variable does not exist, then it will be created. Otherwise, it will be assigned the obtained value. If any of the AF/OPERATOR variables cannot be accessed (for example, because the variable does not exist), then the function will return the value 0 (false). However, the function will continue to try to set other variables if some do exist. If all requested values are obtained, then the value 1 (true) is returned.

If you have stored a variable with leading or trailing blanks, or both, and you want to avoid having the blanks stripped from the variable when the variable is returned, you must specify 'NOTRUNC' as the last parameter.

Note: The GLBVGET function does not accept compound (stem) variable names.

Please see the example below and follow the steps.

1. Fetch the global match variable AOJNAME.
2. Test the status returned by GLBVGET to see whether fetch was successful.
3. Issue message depending on success or failure of the GLBVGET function call.

```/*REXX*/
  status=GLBVGET('AOJNAME')
  IF status = 1 THEN
    SAY "This EXEC was invoked by" aojname
  ELSE
    SAY "Unable to get job name"
```
Variable Functions

GLBVPUT(var1, var2, ..., var20)

This function accepts a list of up to 20 names of REXX variables and assigns them to AF/OPERATOR global variables of the same name. If the AF/OPERATOR variable does not exist, then it will be created. If any of the REXX variables do not exist, or if a corresponding AF/OPERATOR variable cannot be set, then a value of 0 (false) is returned. However, the function will continue to try to set other variables if some do not exist. If all requested values are set, then a value of 1 (true) is returned.

Please see the example below and follow the steps.

1. Define a local variable named BARN and initialize it with “Pig.”
2. Use GLBVPUT function to place the contents of BARN into an AF/OPERATOR global variable of the same name.
3. Test the status returned by the GLBVPUT function and issue a message if the operation failed.
4. Change the value of the local variable BARN to “Cow.”
5. Call GLBVGET to restore the contents of BARN from the AF/OPERATOR global variable.
6. Display the contents of BARN, if GLBVGET was successful.

/*REXX*/
  barn=“Pig”
  status=GLBVPUT(‘BARN’)
  IF status = 0 THEN
      SAY “Unable to store variable BARN”
  barn=“Cow”
  status=GLBVGET(‘BARN’)
  IF status = 1 THEN
      SAY “The contents of BARN = “ barn
  ELSE
      SAY “Unable to fetch contents of BARN”

This should produce the message: “The contents of BARN = PIG.”
**GLBVDEL(var1, var2, ..., var20)**

This function accepts a list of up to 20 names of AF/OPERATOR global variables. The named variables are destroyed. If any of the named variables cannot be destroyed (for example, because the variable does not exist or is protected from user change), then a value of 0 (false) is returned. If all named variables are destroyed, then a value of 1 (true) is returned. However, the function will continue to try to delete other variables if some do not exist.

Please see the example below and follow the steps.

1. Define a local variable named BARN and initialize it with “Horse.”
2. Use GLBVPUT function to place the contents of BARN into an AF/OPERATOR global variable of the same name.
3. Test the status returned by the GLBVPUT function and issue a message if the operation failed.
4. Delete the global variable BARN using GLBVDEL.
5. Change the contents of the local variable BARN to “Hen.”
6. Call GLBVGET to restore the contents of BARN from the AF/OPERATOR global variable which should fail since the variable was deleted from the global pool.
7. Display the contents of BARN. Since the GLBVGET failed, the contents of local variable BARN is unchanged (“Hen”).

```rexx
/*REXX*/
barn="Horse"
status=GLBVPUT('BARN')
IF status = 0 THEN
   SAY "Unable to store variable BARN"

status=GLBVDEL('BARN')
IF status = 0 THEN
   SAY "Unable to delete variable BARN"

barn="Hen"
status=GLBVGET('BARN')
IF status = 0 THEN
   SAY "Unable to fetch contents of BARN"
ELSE
   SAY "Contents of BARN= " barn
```
**GLBVUPD(var,control-value)**

This function accepts the name of an AF/OPERATOR global variable and an arbitrary value. The value of the AF/OPERATOR variable is obtained and compared against the control value. If they are equal, then the value of a REXX variable of the same name is obtained and assigned to the AF/OPERATOR variable, and a value of 1 (true) is returned. Otherwise, the AF/OPERATOR variable remains unchanged and a value of 0 (false) is returned. The control value provides a serialization mechanism to insure that a previously obtained global AF/OPERATOR variable (see GLBVGET) has not been changed by another process before it could be updated by this process.

See the example and the corresponding steps that detail how it functions.

1. Define a local variable named BARN and initialize it with “Chicken.”
2. Use GLBVPUT function to place the contents of BARN into an AF/OPERATOR global variable of the same name.
3. Test the status returned by the GLBVPUT function and issue a message if the operation failed.
4. Save the contents of the local variable BARN, then change it to “Egg.”
5. Use GLBVUPD function to update the contents of global variable BARN.
6. Test the status returned by the GLBVUPD function and issue a message if the operation failed.
7. Change the contents of the local variable BARN back to “Chicken.”
8. Call GLBVGET to restore the contents of BARN from the AF/OPERATOR global variable.
9. Display the contents of BARN. You should see “Egg”!

```/* REXX */
barn="Chicken"
status=GLBVPUT('BARN')
IF status = 0 THEN
   SAY "Unable to store variable BARN"
savebarn=barn
barn="Egg"
status=GLBVUPD('BARN',savebarn)
IF status = 0 THEN
   SAY "Unable to update variable BARN"
barn="Chicken"
status=GLBVGET('BARN')
IF status = 0 THEN
   SAY "Unable to fetch contents of BARN"
ELSE
   SAY "Contents of BARN= " barn```
SYSVGET(var1,var2,...,var20)

This function works like the GLBVGET function, except that it operates on AF/OPERATOR system variables. See GLBVGET for details.

SYSVPUT(var1,var2,...,var20)

This function works like the GLBVPUT function, except that it operates on AF/OPERATOR system variables. See GLBVPUT for details.

SYSVDEL(var1,var2,...,var20)

This function works like the GLBVDEL function, except that it operates on AF/OPERATOR system variables. See GLBVDEL for details.

SYSVUPD(var,control-value)

This function works like the GLBVUPD function, except that it operates on AF/OPERATOR system variables. See GLBVUPD for details.
The AF/OPERATOR REXX communications API (application programming interface) functions in this section allow AF/OPERATOR to manage communication tasks for you.

**COMSDRCV**
Permits sending commands to, and receiving responses from, linked systems. All you need to do is specify the linkid of the partner system, the data, and some information about the data.

**LNKCHECK**
Permits you to ensure that a link is active before sending data.

**COMADMIN**
Controls ownership of an individual connection when using TCP/IP connections without the AF Packet Header. It can also be used to assign an identifier to any individual inbound connection.

### COMSDRCV
Sends data and optionally receives a response from another AF/OPERATOR system or from a distributed application such as AF/Remote. The link between the systems must be predefined with a LINK DEFINE command.

**Syntax**

```
rc=COMSDRCV('linkid','data','datatype','replYLENGTH','SESSION','timeout')
```

**Parameters**

- **linkid**  
The LINK identifier. This can be the LINKID used on the LINK DEFINE command to define the connection, or should be omitted if this is an inbound connection initiated from a remote system (unless an identifier has been assigned using the COMADMIN REXX function (see “COMADMIN” on page 178)). The maximum length of the linkid is 8 characters.

- **data**  
The data to be sent over the connection. The data can be text or it can be contained in a REXX variable. You must also specify its type (see datatype).

  **Note:** If you are using the RECEIVE datatype, you must omit a value for the data parameter.

- **datatype**  
Identifies the type of data in the data field. Depending on the type of link you define, specify one or more of the following types.

  **Note:** SEND and RECEIVE datatypes are only permitted over connections that specify AFPACKET(OFF) when using LINK DEFINE.
**COMMUNICATION FUNCTIONS**

**CMDRSP**  
The data is any valid MVS or subsystem command which will generate a response. The maximum length that can be sent is 126 characters.

**DATARPLY**  
This parameter is used in a REXX exec running on an AF/OPERATOR server, to send response data back to the originating system. There is no AF/OPERATOR restriction on the length of the outbound data.

*Note:* In a COM match, the linkid for DATARPLY is ignored, because the response data is automatically returned to the originating system. In a non-COM match, a linkid is permitted.

**EXEC**  
The data is any valid AF/OPERATOR command (such as an EXEC, WTO, or OPER command) to be executed on the partner system. The maximum length that can be sent is 180 characters.

**USRDATA**  
The data is of a user-defined type. Specify USRDATA when transferring data to a distributed application. There is no AF/OPERATOR restriction on the length of the transferred data.

*Note:* USRDATA must be specified in order to initiate an APPC communication session with AF/REMOTE. No other datatype is valid.

**WTOR**  
The data is a WTO with response. The maximum length that can be sent is 122 characters.

**Rerouting WTOs and Commands:**

**WTORT**  
The data is a WTO to be routed through an intermediate system and forwarded to its final destination. The first 8 bytes of data must contain the linkid on the intermediate system over which the WTO/command will be rerouted to its final destination. The rest of the data is considered WTO/command text. If the intermediate linkid is less than 8 characters, pad it with blanks. The linkid over which the WTO/command is being forwarded must be predefined and started with LINK DEFINE/LINK START commands on the intermediate system. The maximum length of the WTO/command that can be sent is 126 characters. (The maximum total length including the linkid is 134 characters.)

**CMDRT**  
The data is any MVS or subsystem command to be routed through an intermediate system and forwarded to its final destination. The first 8 bytes of data must contain the linkid on the intermediate system over which the WTO/command will be rerouted to its final destination. The rest of the data is considered WTO/command text. If the intermediate linkid is less than 8 characters, pad it with blanks. The linkid over which the WTO/command is being forwarded must be predefined and started with LINK DEFINE/LINK START commands on the intermediate system. The maximum length of the WTO/command that can be sent is 126 characters. (The maximum total length including the linkid is 134 characters.)
**CMDRSPRT**  The data is any MVS or subsystem command from which a response is requested. The command is routed through an intermediate system and forwarded to its final destination. The first 8 bytes of data must contain the linkid on the intermediate system over which the WTO/command will be rerouted to its final destination. The rest of the data is considered WTO/command text. If the intermediate linkid is less than 8 characters, pad it with blanks. The linkid over which the WTO/command is being forwarded must be predefined and started with LINK DEFINE/LINK START commands on the intermediate system. The maximum length of the WTO/command that can be sent is 126 characters. (The maximum total length including the linkid is 134 characters.)

**When LINK DEFINE specifies AFPACKET(OFF).**  See also Comments for SEND/RECEIVE (page 173).

**SEND**  Used to send data over a connection established between two systems. Use is only permitted over connections having a LINK DEFINE specification of AFPACKET(OFF).

**RECEIVE**  Used to receive data over a connection established between two systems. Use is only permitted over connections having a LINK DEFINE specification of AFPACKET(OFF).

**replylength**  Optional. If you want to receive a response, specify the maximum length, in bytes, of the response data string. That is, the length of the response generated by the sent data (for example, by a CMDRSP, CMDRSPRT, EXEC, WTOR, or DATARPLY request). The maximum replylength that can be specified is 32K.

**session**  Optional, depending on how the link is defined. You can explicitly specify which session on an AF/OPERATOR to AF/OPERATOR link is used to transmit data. The default AF/OPERATOR send only session will be used if no session specification is made. You can alter the session default by specifying the following.

**SENDRCV**  Send and receive data in a two-way alternate session on the same link. If you specify a SENDRCV session here, ensure that SENDRCV is also specified on the LINK DEFINE command for this link. If the corresponding definition is not present on the LINK DEFINE command, the SENDRCV session does not exist. Distributed applications, such as AF/Remote, do not support SENDRCV.

**timeout**  Number of seconds to wait for data to be received before the operation will time out. If you omit this argument, an infinite wait time is assumed.
Comments

- The COMSDRCV function can be invoked from a REXX exec, running on an AF/OPERATOR system, using standard REXX syntax. COMSDRCV uses a positional parameter list. If parameters are omitted, they must be indicated by commas.

- The COMSDRCV function is not intended as a file transfer program.

- A replylength should be specified for datatypes that return responses (CMDRSP, CMDRSPRT, EXEC, WTOR and DATARPLY request) if you intend to receive a response. If replylength is not specified for a command that generates a response (datatype CMDRSP, CMDRSPRT, or WTOR), a syntax error occurs. If replylength is not specified for an AF/OPERATOR command (datatype EXEC or DATARPLY request) that generates a response, data is discarded. In most cases the maximum replylength that can be specified is 32K. The replylength for datatype WTOR is restricted to a maximum of 119 bytes. This restriction is imposed by the MVS operating system. Do not specify a replylength for datatype WTOR; this generates a syntax error.

  replylength can also be used on a SEND to imply that a RECEIVE is to be issued once the data has been sent.

- Response data is stored in a single REXX compound variable whose name is COMDATA.1. To remain consistent with other AF/OPERATOR-supplied REXX functions, the total number of variables returned is stored in variable COMDATA.0.

- In the string of response data returned by AF/OPERATOR for datatype CMDRSP, CMDRSPRT, or EXEC, each 150 bytes represent 1 line.

- When AF/OPERATOR initiates a session to a distributed application, outbound data is in EBCDIC format.

- The maximum length of outbound data for datatype USRDATA, though not restricted by AF/OPERATOR, is restricted by the partner application’s maximum length for received data.

- Use of DATARPLY can be used by any match type. It requires that a reply has been requested by a partner and is still outstanding. A linkid is permitted to be used with DATARPLY in a non-COM match.

- A CMDSDRCV EXEC function that causes an OPER with RESP command to be executed on a remote system will also cause an AOCASE REXX variable to be built when the reply from the remote system is received. See “Determining the status of the response lines for zero return codes” on page 313 for a description of the values for the AOCASE variable.

Comments for SEND/RECEIVE

- AF/OPERATOR uses stream sockets. Stream sockets treat communications simply as a continuous stream of bytes.

- RECEIVE sets a REXX variable called COMDATA.1, that contains the data received. Its value can be shorter than the specified replylength value, and another RECEIVE may be necessary. If the full length expected is not received at once, then you must issue another RECEIVE and continue to loop through this process until all of the data is transferred.
If there is no data available to receive, then the REXX function causes the match to be suspended until there is at least 1 byte.

Examples

The following exec sends data to a session identified by LINKID RMT.

```rexx
/* REXX */
ADDRESS APHOST
"LINK DEFINE LINKID(RMT) MODENAME(LU62) PLUNAME(SYSTEM1) TPNAME(RMTPGM)"
IF rc <> 0 THEN SAY "LINK DEFINE FAILED"
ELSE DO
  "LINK START L(RMT)"
  IF rc <> 0 THEN SAY "LINK START FAILED"
  ELSE DO
    LINKID = "RMT"
    rc = LNKCHECK(LINKID)
    IF rc <> 0 THEN SAY "LINK NOT READY"
    ELSE DO
      DATA = 'user defined'
      rc = COMSDRCV(LINKID,DATA,'USRDATA',200)
      IF rc <> 0 THEN SAY "COMSDRCV ERROR"
      ELSE SAY "DATA SENT OK"
    END
  END
END
RETURN rc
```

The following exec sends data to a session identified by linkid RMT, and requests a response of 200 bytes.

```rexx
ADDRESS APHOST
"LINK DEFINE L(RMT) M(LU62) P(SYSTEM1) TP(RMTPGM)"
IF rc <> 0 THEN SAY "LINK DEFINE FAILED"
ELSE DO
  "LINK START LINKID(RMT)"
  IF rc <> 0 THEN SAY "LINK START FAILED"
  ELSE DO
    LINKID = "RMT"
    rc = LNKCHECK(LINKID)
    IF rc <> 0 THEN SAY "LINK NOT READY"
    ELSE DO
      DATA = 'user defined'
      rc = COMSDRCV(LINKID,DATA,'USRDATA',200)
      IF rc <> 0 THEN SAY "COMSDRCV ERROR"
      ELSE SAY "COMDATA.1"
    END
  END
END
RETURN rc
```
## Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful data transfer (a default send only session was requested).</td>
</tr>
<tr>
<td>-1</td>
<td>Invalid or null linkid parameter.</td>
</tr>
<tr>
<td>-2</td>
<td>Invalid or null data parameter.</td>
</tr>
<tr>
<td>-3</td>
<td>Invalid data parameter length.</td>
</tr>
<tr>
<td>-4</td>
<td>Invalid or null datatype parameter.</td>
</tr>
<tr>
<td>-5</td>
<td>Invalid session parameter.</td>
</tr>
<tr>
<td>-6</td>
<td>Invalid replylength parameter.</td>
</tr>
<tr>
<td>-7</td>
<td>Invalid parameter list.</td>
</tr>
<tr>
<td>-8</td>
<td>Invalid timeout parameter</td>
</tr>
<tr>
<td>4</td>
<td>Incomplete data received. The length of the response data exceeded the replylength specification.</td>
</tr>
<tr>
<td>5</td>
<td>Program error received from a SENDRCV session.</td>
</tr>
<tr>
<td>8</td>
<td>Internal error. Contact Candle Support Services.</td>
</tr>
<tr>
<td>12</td>
<td>Link is terminated.</td>
</tr>
<tr>
<td>16</td>
<td>Timeout period has expired.</td>
</tr>
<tr>
<td>20</td>
<td>Request cannot be sent while conversational DATARPLY is owed to another link.</td>
</tr>
<tr>
<td>24-36</td>
<td>Internal error. Contact Candle Support Services.</td>
</tr>
<tr>
<td>500</td>
<td>Internal error. Contact Candle Support Services.</td>
</tr>
<tr>
<td>999</td>
<td>Processing abended. Contact Candle Support Services.</td>
</tr>
<tr>
<td>1004</td>
<td>Linkid not defined.</td>
</tr>
<tr>
<td>1008</td>
<td>Linkid defined but not active.</td>
</tr>
<tr>
<td>1012</td>
<td>Datatype is invalid or is not permitted to be sent over this link.</td>
</tr>
<tr>
<td>1016</td>
<td>Match does not own this link.</td>
</tr>
<tr>
<td>1020-1099</td>
<td>Internal error. Contact Candle Support.</td>
</tr>
<tr>
<td>1109</td>
<td>Message not sent—link termination in progress.</td>
</tr>
<tr>
<td>2008</td>
<td>EXEC processing STOP requested.</td>
</tr>
<tr>
<td>2012</td>
<td>EXEC processing no free match.</td>
</tr>
<tr>
<td>2016</td>
<td>EXEC processing internal error.</td>
</tr>
<tr>
<td>2020</td>
<td>EXEC processing replylength requested for datatype WTORT or DATARPLY.</td>
</tr>
<tr>
<td>2024-2998</td>
<td>EXEC processing internal error.</td>
</tr>
<tr>
<td>2124</td>
<td>No subsystem consoles available.</td>
</tr>
<tr>
<td>2824</td>
<td>EXEC processing error. The exec (or REXX script) executing on the partner system may contain a syntax error or a REXX exec needed a response but the response was not sent.</td>
</tr>
<tr>
<td>2999</td>
<td>The COM match has abended.</td>
</tr>
<tr>
<td>3004</td>
<td>CMDRSP processing STOP requested.</td>
</tr>
<tr>
<td>3008</td>
<td>CMDRSP timed out before the requested response data could be returned.</td>
</tr>
<tr>
<td>3012-3999</td>
<td>CMDRSP processing internal error.</td>
</tr>
<tr>
<td>4000-4999</td>
<td>WTOR processing internal error.</td>
</tr>
<tr>
<td>6000</td>
<td>Unknown type of data. Possibly, data was sent to AF/OPERATOR with a datatype specification of USRDATA.</td>
</tr>
</tbody>
</table>
**LNKCHECK**

Determines whether the specified link session is active.

**Syntax**

```
rc=LNKCHECK('linkid','NOWAIT' ,',sessiontype') =>
```

**Parameters**

- **linkid**: The ID of the partner system. This is the same `linkid` as defined on the LINK DEFINE command. The maximum length of the `linkid` is 8 characters.
- **NOWAIT**: Forces serialization of LNKCHECK requests.
- **sessiontype**: Optional. If no specification is made, AF/OPERATOR checks only the one-way (send only) session that is the LINK DEFINE command’s default. If a SENDRCV session was defined, you can explicitly specify which session, on the specified link, to check by adding one of the following keywords.
  - **SENDRCV**: Checks only the two-way (half-duplex) session.
  - **BOTH**: Checks the default (send only) session and the SENDRCV sessions.

**Comments**

- The LNKCHECK command is not intended for heartbeat checking. The AF/OPERATOR link component performs heartbeat checking by default, so that you don’t have to.
- You can use LNKCHECK to ensure that the specified session is active, before sending data.
Example

/* REXX */
ADDRESS APHOST
"LINK DEFINE LINKID(RMT) MODENAME(LU62) PLUNAME(SYSTEM1) TPNAME(RMPGM)"
IF rc <> 0 THEN SAY "LINK DEFINE FAILED"
ELSE DO
   "LINK START LINKID(RMT)"
   IF rc <> 0 THEN SAY "LINK START FAILED"
   ELSE DO
      LINKID = "RMT"
      rc = LNKCHECK(LINKID)
      IF rc <> 0 THEN SAY "LINK NOT READY"
      ELSE DO
         DATA = "user defined"
         rc = COMSDRCV(LINKID,DATA,'USRDATA')
      END DO
   END DO
.
.
.

Function Codes

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The specified session is active.</td>
</tr>
<tr>
<td>-1</td>
<td>Invalid or null linkid specified.</td>
</tr>
<tr>
<td>-2</td>
<td>Invalid sessiontype specified.</td>
</tr>
<tr>
<td>-3</td>
<td>Linkid inactive.</td>
</tr>
<tr>
<td>-4</td>
<td>Default (send only) session inactive.</td>
</tr>
<tr>
<td>-5</td>
<td>SENDRCV session inactive.</td>
</tr>
<tr>
<td>-6</td>
<td>Invalid parameter list.</td>
</tr>
<tr>
<td>4</td>
<td>Linkid not defined.</td>
</tr>
<tr>
<td>8</td>
<td>Link is not active.</td>
</tr>
<tr>
<td>12</td>
<td>Invalid type.</td>
</tr>
<tr>
<td>16</td>
<td>Internal error.</td>
</tr>
<tr>
<td>20</td>
<td>No ssct extension.</td>
</tr>
<tr>
<td>36</td>
<td>Either a) the specified LINKID is locked by another user performing a LINK START, LINK STOP, or LNKCHECK with NOWAIT; or b) the internal LNKCHECK timer has expired after 5 seconds of waiting for the LINKID to be freed.</td>
</tr>
</tbody>
</table>
COMADMIN

Controls ownership of an individual connection when using TCP/IP connections without the AF Packet Header. It can also be used to assign an identifier to any inbound TCP/IP connection.

Syntax

```
rc=COMADMIN( 'GIVE', connection_id, timeout )
rc=COMADMIN( 'TAKE', connection_id, timeout )
rc=COMADMIN( 'ASSIGN', connection_id )
```

Parameters

**GIVE**

Passes ownership of the connection named to another match. See “GIVE and TAKE requests” on page 179.

**TAKE**

Takes ownership of the named connection. See “GIVE and TAKE requests” on page 179.

**ASSIGN**

Available to be used only by the CONNECT EXEC match, assigns a connection identifier to the connection for which the match was created. An inbound connection can be assigned an identifier that effectively corresponds to a LINKID as if it were used on the LINK DEFINE command. The identifier assigned follows the same basic naming conventions as those for a LINKID. (The identifier can have a maximum length of 8 characters, consisting of alpha-numeric and national characters ($, # and @)). This permits the inbound connection identifier to be used with the COMSDRCV REXX function as well as with the GIVE and TAKE operations.

*Note* A connection can only be assigned an identifier once.

**connection Id**

The unique identifier associated with or being associated with this connection. For GIVE and TAKE operations, this identifier can be that assigned to a LINK by means of a LINK DEFINE or it can be the ID associated with an inbound connection by the previous use of an ASSIGN operation.

**timeout**

Used only with the GIVE operation, this is the number of seconds that the GIVE operation will wait for the new owner to TAKE ownership of the named connection. The default is 30 seconds.
Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Operation completed successfully</td>
</tr>
<tr>
<td>4</td>
<td>Timeout period elapsed</td>
</tr>
<tr>
<td>8</td>
<td>Connection was terminated during a GIVE request.</td>
</tr>
<tr>
<td>16</td>
<td>Syntax error; invalid connection-ID.</td>
</tr>
<tr>
<td>20</td>
<td>Named connection on GIVE or TAKE request was not located</td>
</tr>
<tr>
<td>24</td>
<td>Ownership not applicable for the named connection.</td>
</tr>
<tr>
<td>28</td>
<td>Match does not own the connection; cannot GIVE.</td>
</tr>
<tr>
<td>32</td>
<td>Cannot TAKE connection; the current owner is not relinquishing ownership.</td>
</tr>
<tr>
<td>36</td>
<td>ASSIGN requests are only permitted in a CONNECT_EXEC match.</td>
</tr>
<tr>
<td>40</td>
<td>Identifier being assigned already exists.</td>
</tr>
<tr>
<td>44</td>
<td>Logon no longer exists. It has probably terminated.</td>
</tr>
<tr>
<td>48</td>
<td>Internal error. Not a duplex connection.</td>
</tr>
<tr>
<td>52</td>
<td>Internal error. Protocol invalid.</td>
</tr>
<tr>
<td>88</td>
<td>Internal error. REXX environment block missing.</td>
</tr>
<tr>
<td>92</td>
<td>Internal error. Bad $ESTAE setup.</td>
</tr>
<tr>
<td>96</td>
<td>Internal error. $ESTAE invoked.</td>
</tr>
</tbody>
</table>

Connection ownership

AF permits only one match to access a particular AF PACKET(OFF) connection at any one time. This match is referred to as the owner.

For outbound connections, the initial owner is the match that performs the LINK START command for that connection. For inbound connections, the initial owner is the CONNECT_EXEC match that is executed in response to a connection being made to a server.

If a match ends while it owns an active connection, the connection is automatically terminated.

GIVE and TAKE requests

The GIVE and TAKE request types allow ownership of an active connection to be transferred from one match to another. In most situations, it probably won’t be necessary to transfer connection ownership. Usage of GIVE and TAKE is totally dependent upon the design of the application using the communications functions.

A match that owns a connection needn’t ever send or receive data over the connection. It could immediately GIVE the link to another match. The TAKE could be performed by a
straightforward TOD(*) or another long running match that acts as a manager and could be given many links to handle.

Possible usage situation - consolidation
Rather than have many instances of the CONNECT_EXEC running simultaneously, a single match could be made to control all connections to one specific port.

When the CONNECT EXEC instance begins to execute, it would immediately GIVE ownership of the connection. The controller match is posted or polls and takes the connection. It now has full control over the connection. This instance of the CONNECT_EXEC can no longer use COMSDRCV to send or receive data over this connection.

Example:
The owning match can relinquish ownership using GIVE:

/* REXX */
"LINK DEFINE LINK(ABC123) HOST(FRED) PORT(9999) AFPACKET(OFF)"
"LINK START LINK(ABC123)"

RC = COMADMIN('GIVE', 'ABC123', 120)
Another match can now TAKE ownership:

/* REXX */
RC = COMADMIN('TAKE', 'ABC123')

MY_DATA = 'TEST DATA @' TIME()
RC = COMSDRCV('ABC123', MY_DATA, 'SEND')

The ASSIGN request
When the COMSDRCV REXX function is used within a COM match, it does not need to identify the return route. It simply omits the first argument. The CONNECT_EXEC executes as a special type of COM match and can send and receive data using the NULL linkid specification.

Example:

/* REXX */
RC = COMSDRCV(,, 'RECEIVE', 200)
SAY 'TESTRECV - NUMBER OF BYTES RECEIVED:' LENGTH(COMDATA.1)
SAY 'TESTRECV - DATA RECEIVED:' COMDATA.1

RC = COMSDRCV(COMDATA.1, 'SEND') /* Echo received data */
If ownership of an inbound connection is to be transferred to another match, the ASSIGN request **must** be used to give the connection a unique identifier. Doing so is **required** in this situation because, once ownership has passed from the CONNECT EXEC match, the return route is not automatically known. Using the ASSIGN request is also necessary because it could be possible to pass more than one inbound connection into a single match.

**Example:**

```rexx
/* REXX */
RC = COMSDRCV(,,'RECEIVE',200)
SAY 'TESTRECV - NUMBER OF BYTES RECEIVED:' LENGTH(COMDATA.1)
SAY 'TESTRECV - DATA RECEIVED:' COMDATA.1

RC = COMADMIN('ASSIGN','XYZ123')

IF RC = 0 THEN DO
  RC = COMSDRCV('XYZ123',,'RECEIVE',200)
  SAY 'TESTRECV - NUMBER OF BYTES RECEIVED:' LENGTH(COMDATA.1)
  SAY 'TESTRECV - DATA RECEIVED:' COMDATA.1

  RC = COMADMIN('GIVE','XYZ123',120)
  SAY 'TESTRECV - GIVE RETURN CODE:' RC
END
ELSE SAY 'TESTRECV - ASSIGN FAILED RETURN CODE:' RC
```
Introduction

This chapter discusses how AF/OPERATOR commands are issued and provides an alphabetical list of AF/OPERATOR commands, including their syntax, comments, examples, and related information. REXX functions, such as OMEGAVIEW functions and POVI functions, are listed in “Candle-Supplied REXX Functions” on page 93. The POVI command primitives are listed separately under “Programmerless Open VTAM Interface Commands” on page 423.

Note: Default operands are underlined. Since many AF/OPERATOR product defaults can be changed based on parameter library selections made at your site during the configuration process, defaults at your site may differ from those listed in this section. For more information about command syntax, please see the Conventions section at the beginning of this manual.

Chapter Contents

Issuing Commands .......................................................... 187
AF .................................................................................. 189
AF (no operand) ............................................................ 192
AF AUTOREF and AF NOAUTOREF .................................. 193
AF CANCEL ................................................................. 195
AF CMD and AF NOCMD ................................................ 197
AF COMTRACE ......................................................... 199
AF DOM and AF NODOM ............................................... 204
AF FREE ...................................................................... 206
AF LOG and AF NOLOG ................................................ 207
AF MATINFO ............................................................... 208
AF QUIET and AF NOQUIET ......................................... 209
AF RECOVER and AF NORECOVER ................................. 211
AF RELOAD(HOSTNAME) .............................................. 212
AF RXSEARCH .......................................................... 213
AF SSIRESET and AF SSIDRAIN .................................... 214
AF STATS ..................................................................... 215
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AF STOP and AF KILL</td>
<td>217</td>
</tr>
<tr>
<td>AF TODSTART</td>
<td>219</td>
</tr>
<tr>
<td>AF WARN and AF PROCESS</td>
<td>220</td>
</tr>
<tr>
<td>AF WTO and AF NOWTO</td>
<td>222</td>
</tr>
<tr>
<td>AFADDR</td>
<td>223</td>
</tr>
<tr>
<td>AFCALL</td>
<td>226</td>
</tr>
<tr>
<td>ALLOC (AL)</td>
<td>228</td>
</tr>
<tr>
<td>BEEP</td>
<td>231</td>
</tr>
<tr>
<td>CLOSE (CL)</td>
<td>234</td>
</tr>
<tr>
<td>COMM</td>
<td>236</td>
</tr>
<tr>
<td>COMM DISPLAY</td>
<td>237</td>
</tr>
<tr>
<td>COMM RELEASE</td>
<td>245</td>
</tr>
<tr>
<td>COMM STOP</td>
<td>252</td>
</tr>
<tr>
<td>COMM SUSPEND</td>
<td>255</td>
</tr>
<tr>
<td>CP</td>
<td>258</td>
</tr>
<tr>
<td>DELETE (DEL)</td>
<td>259</td>
</tr>
<tr>
<td>DOM</td>
<td>260</td>
</tr>
<tr>
<td>EXEC (EX)</td>
<td>262</td>
</tr>
<tr>
<td>EXECDROP</td>
<td>264</td>
</tr>
<tr>
<td>EXECLOAD</td>
<td>266</td>
</tr>
<tr>
<td>EXECMAP</td>
<td>267</td>
</tr>
<tr>
<td>FREE (FR)</td>
<td>268</td>
</tr>
<tr>
<td>HELP (HE)</td>
<td>270</td>
</tr>
<tr>
<td>IMSOPER</td>
<td>272</td>
</tr>
<tr>
<td>JES3OPER (J3OPER)</td>
<td>273</td>
</tr>
<tr>
<td>LINK</td>
<td>274</td>
</tr>
<tr>
<td>LINK DEFINE</td>
<td>275</td>
</tr>
<tr>
<td>LINK DELETE</td>
<td>280</td>
</tr>
<tr>
<td>LINK DISPLAY</td>
<td>283</td>
</tr>
<tr>
<td>LINK START</td>
<td>286</td>
</tr>
<tr>
<td>LINK STOP</td>
<td>289</td>
</tr>
<tr>
<td>LOGOFF</td>
<td>292</td>
</tr>
<tr>
<td>LOGON</td>
<td>293</td>
</tr>
<tr>
<td>MLWTO</td>
<td>298</td>
</tr>
<tr>
<td>NVALERT</td>
<td>301</td>
</tr>
<tr>
<td>NVISHOW</td>
<td>304</td>
</tr>
<tr>
<td>NVISTART</td>
<td>306</td>
</tr>
<tr>
<td>NVISTOP</td>
<td>307</td>
</tr>
<tr>
<td>OPEN (OP)</td>
<td>308</td>
</tr>
</tbody>
</table>
OPER . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  .  310
PEER . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  316
PEER START . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  317
PEER STOP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  318
PEERLINK . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  320
PEERLINK DEFINE . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  322
PEERLINK DELETE. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  323
PEERLINK DISPLAY . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  324
PEERLINK START . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  325
PEERLINK STOP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  326
Programmerless Open VTAM Interface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  327
READ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  .  328
SHOW (SH) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  330
SHOW ENQUEUE . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  332
SHOW FILES . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  334
SHOW HOSTNAME . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  335
SHOW LOGONS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  336
SHOW MAINT . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  337
SHOW MATCHES . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  338
SHOW OG . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  339
SHOW RSVS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  343
SHOW SSCONS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  346
SHOW SSCTS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  347
SHOW STATS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  349
SHOW SVCS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  351
SHOW TRAPS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  353
SHOW TRAPS TITLE FORMAT . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  355
SHOW VARS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  358
TPUT (TP) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .   360
TRAP (TR) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .   362
TRAP ADD CMD . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  364
TRAP ADD TOD . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  371
TRAP ADD WTO . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  376
TRAP ADD xtype . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  386
TRAP CHANGE CMD . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  390
TRAP CHANGE WTO . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  396
TRAP CHANGE xtype . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  403
TRAP DELETE. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  407
TRAP DISABLE . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  409
TRAP ENABLE .......................................................... 411
WAIT ................................................................. 413
WRITE .............................................................. 417
WTO ................................................................. 418
WTOR ............................................................... 421
Programmerless Open VTAM Interface Commands ................................ 423
  \@TYPE .......................................................... 424
  \#TYPE ........................................................ 425
  \K ............................................................... 426
  \LOC ........................................................... 427
  \RC ............................................................. 428
  \S ............................................................... 429
  \SEE .......................................................... 430
  \SET .......................................................... 431
  \STATUS ....................................................... 432
  \TYPE ........................................................ 433
  \WAIT ......................................................... 434
Programmerless Open VTAM Interface Operator Commands .................... 435
  ECHO .......................................................... 436
  LOGOFF ......................................................... 437
  MONITOR ....................................................... 438
  REFRESH ....................................................... 439
  VSHOW ......................................................... 440
Issuing Commands

There are several ways to issue AF/OPERATOR commands:

- Any command can be issued by a trap when it is triggered.
- Commands can be issued in the execution of a REXX exec.
- Some commands can be issued interactively from a TSO terminal.
- Some commands can be issued interactively from a console.

Issuing Commands from a Trap

A trap can issue any AF/OPERATOR command. Include the command in the ACTION field of the TRAP ADD command that defines the trap (or in a TRAP CHANGE command). For example,

```
TRAP ADD(QUIET) CMD('QUIESCE *') +
   ACTION('WTO "QUIESCE NOT ALLOWED"') SUPPRESS ENABLE
```

defines a trap which, when triggered, issues the following command:

```
WTO 'QUIESCE NOT ALLOWED'
```

If you issue the TRAP command from the console, the command in the ACTION field of the TRAP command cannot exceed 128 characters, excluding the single quotes around it.

Issuing Commands from a REXX exec

You can store REXX execs in a partitioned dataset for later execution by an EXEC command. “Understanding the Programming Environment” on page 21 presents an overview of REXX execs.

Notes:

1. The language elements in AF/OPERATOR (such as the product control commands in this chapter), can be used in REXX execs.
2. The examples in this chapter are written in REXX exec format. See “Sample REXX Language Address” on page 25 for more information.

Issuing Host Commands

When issuing a command that you want to address to the host environment, you must enclose the command in double quotes. To designate a literal value in REXX, use single quotes.

Issuing Commands from a TSO Terminal

The AF/OPERATOR command processor can receive commands from a TSO terminal when it is emulating an operator console—as it does, for example, through OMEGAMON with the OCMD command, or IBM’s SDSF® (System Display and Search Facility). Any TSO terminal capable of issuing operator commands can control the AF/OPERATOR system just as it would control any started task or job in the system. The command processor uses TPUT to return the command response to the TSO user who issued the command in a JES2 environment.
Issuing Commands

By using the TSO/E session manager, you can make command responses scrollable. This avoids repeating the command to reread a large display.

Issuing Commands from an Operator Console

Issuing automation commands is like issuing MVS commands. The AF/OPERATOR processor automatically recognizes the commands and sends responses back to the same console that issued them.

Enter a command at the operator’s console as you would a standard MVS command, by typing the text and pressing the Enter key. A console command cannot exceed 128 characters (or the number of characters you can enter at your console). There is no provision for continuing a command to new physical lines.

Separating Multiple Values in a Command

When you issue a command that has multiple values, you can use either a comma or a space to separate the values. For example, both of the following are valid:

```
TRAP ADD CONSOLE(1 2 55)
TRAP ADD CONSOLE(1,2,55)
```
AF controls and displays various AF/OPERATOR settings. (Can be issued from the console.) This product control command sets the operation mode for AF/OPERATOR, stops AF/OPERATOR, and performs many other control functions.

This section gives a general description of the AF product control command. Because of the complexity of the command, its subtypes are described separately in subsequent sections, in alphabetical groupings.

Syntax

```
subsys AF <option/s>
```

Options:
```
subsys

AUTOREF | NOAUTOREF
CANCEL(number)
CMD | NOCMD
COMTRACE(action)
DOM | NODOM
DUB_AS_PROCESS
FREE(ddname)
LOG | NOLOG
MATINFO
QUIET | NOQUIET
RECOVER | NORECOVER
RELOAD(HOSTNAME)
RXSEARCH
SSIRESET | SSIDRAIN
STOP | KILL
TODSTART
WARN | PROCESS
WTO | NOWTO
```

(No operand) The AF product control command issued with no operand displays AF/OPERATOR’s current operating mode and status.

subsys This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

AUTOREF Automatically a REXX exec before executing it. Thus ensures that you pick up the newest version of an exec.

NOAUTOREF Do not automatically reload a REXX exec before executing it; use the version already in storage.
CANCEL(number) Terminates a specified match. To determine the number of the match, use the SHOW MATCHES command.

COMTRACE(action) Activates or deactivates a trace facility that provides diagnostic information about communications services. Used at the request of Candle Customer Support.

CMD Specifies that AF/OPERATOR will monitor operator commands for possible matches with enabled traps.

NOCMD Stops AF/OPERATOR from monitoring operator commands for possible matches with enabled traps.

Important: This command prevents you from issuing AF/OPERATOR commands from the consoles and from trapping commands. You can, however, issue commands using the MVS MODIFY command (F task, command). For example, F task,AF CMD allows command processing.

DOM AF DOM starts or resumes delete operator message (DOM) processing. (Can be issued from the console.)

NODOM AF NODOM suspends delete operator message (DOM) processing.

DUB_AS_PROCESS Changes the default OS/390 UNIX System Services setting for DUB so that any new subtask is treated as a separate process.

FREE Deallocates and then reallocates an AF/OPERATOR reserved sysout dataset. You must specify the ddname; for example, RKOGLOGM.

LOG Writes a message to the AF/OPERATOR message log every time a trap is triggered.

NOLOG Closes the AF/OPERATOR message log but does not free it. No message is written to the AF/OPERATOR log when a trap is triggered. Also, the results of the trap action are not logged.

MATINFO Enables or disables the message indicating the total CPU time used for a completed match.

QUIET Suppresses all messages from AF/OPERATOR to the operator console and SYSLOG, except those that are in response to an operator command. The following messages are not suppressed by AF QUIET:

!AOP0120 - !AOP0159
!AOP0200 - !AOP0299
!AOP0501 - !AOP0502
!AOP0505
!AOP0767 - !AOP0769
!AOP2021
!AOP3001 - !AOP3116
!AOP3230 - !AOP3239

NOQUIET Specifies that messages from AF/OPERATOR will not be suppressed.

RECOVER Enables abend recovery by AF/OPERATOR. AF/OPERATOR analyzes, reports on, and recovers from system and user abends.

NORECOVER Disables abend recovery by AF/OPERATOR. Any subsequent system or AF/OPERATOR abend will terminate AF/OPERATOR.

Important: Use only at the request of Candle Customer Support.
Comments

To provide downwards compatibility with previous versions, the AO command functions as a command synonym for the AF command.

Once you have disabled console command processing with AF NOCMD, you must issue AF CMD using the MVS MODIFY command (F task, command) to re-enable it. For example, F task,AF CMD resumes command processing.

Examples

The following product control commands show some of its uses.

- **AF CANCEL(1234)**
  
  cancels match 1234.

- **AF STOP**
  
  ends AF/OPERATOR after all active matches end.
AF (no operand)

AF with no operand displays AF/OPERATOR’s current operating mode and options in effect. (Can be issued from the console.)

Syntax

```
subsys AF ->
```

*subsys* This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The *subsys* name is assigned by the SUBSYS startup parameter.

Response

AF/OPERATOR issues messages AOP150 and AOP151 to provide status information including:

- The name of the product job or started task name
- Whether the product is in process mode or WARN mode
- The event types (commands, WTOs, etc.) currently being trapped.
- The POVI trigger character. Note that the POVI trigger character (\) is displayed in three different ways, depending on where the AF command was invoked. Under TSO, the backslash appears as a colon (:). In the SYSLOG, it appears as a blank. In RKOGLOGM, the backslash (\) appears.

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>AF/OPERATOR mode and option settings are successful.</td>
</tr>
<tr>
<td>12</td>
<td>Dataset name does not exist, or file with ddname is in use.</td>
</tr>
<tr>
<td>16</td>
<td>Invalid syntax.</td>
</tr>
</tbody>
</table>

Comments

To provide downwards compatibility with previous versions, the AO command functions as a command synonym for the AF command.
AF AUTOREF and AF NOAUTOREF

Description
AF AUTOREF specifies that each time a REXX exec is executed, it will be reloaded to reflect the latest changes to it. (Can be issued from the console.) AUTOREF also removes execs from storage when their use count reaches zero.

Syntax

```
subsys AF AUTOREF -->
```

subsys This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

NOAUTOREF Option
AF NOAUTOREF specifies that REXX execs will not be reloaded automatically before being executed. The version already in storage will be used. AF NOAUTOREF also retains execs in storage when their use count reaches zero.

Syntax

```
subsys AF NOAUTOREF -->
```

subsys This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

Comments
- To provide downwards compatibility with previous versions, AO AUTOREF functions as a command synonym for AF AUTOREF.
- With NOAUTOREF, whenever you revise a REXX exec you need to remove the old version of the exec from storage with an EXECDROP command. The new version will be loaded into storage the next time the exec is executed.
- AUTOREF is useful when AF/OPERATOR is first installed, and you are creating and revising a great number of REXX execs. However, AUTOREF requires much more system overhead to reload REXX execs before executing them. When your trap library is stable, you may want to run under the NOAUTOREF option.
AF AUTOREF and AF NOAUTOREF

- After testing your AF/OPERATOR REXX execs, turn AUTO REFRESH off. NOAUTOREF optimizes performance of REXX execs and reduces the amount of I/O to your exec libraries by keeping the execs loaded in storage.

- Alternatively, you can specify AUTOREF and as execs you develop become stable, selectively load those execs into storage with the EXECLOAD command. This will improve performance.

- The EXECMAP command lists the status of REXX execs known to AF/OPERATOR.
AF CANCEL

Description
AF CANCEL terminates a matched trap. (Can be issued from the console.)

If a matched trap is hung (that is, it will never continue processing), you may use AF CANCEL to terminate it. This termination does not affect the processing of AF/OPERATOR.

Syntax

```
subsys AF CANCEL(number) DUMP FORCE
```

- **subsys**: This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The `subsys` name is assigned by the SUBSYS startup parameter.

- **number**: Specifies the match entry number displayed by the SHOW MATCH command. AF/OPERATOR assigns each match a unique number.

- **DUMP**: Specifies that an SVC dump will be taken and stored in the SYS1.DUMPx data set.

- **FORCE**: Terminates all matched traps, including the traps that operate AF/OPERATOR. *Use this command with extreme caution.*

Response
A message such as the following appears:

```
!AOP0140 MATCH 3 CANCEL COMMAND ACCEPTED
```

If no active match corresponds to the number entered, AF/OPERATOR will respond with an error message and a U222 abend is generated. The following error message appears:

```
!AOP0141 MATCH 3 NOT FOUND
```

Comments
- When a hung match is cancelled, jobs waiting in response to the WAIT option on the TRAP command are allowed to continue, and all temporary wait traps due to the WAIT host command are deleted.

- Execute the following commands to cancel a hung match:

  1. **SHOW MATCH** to identify the hung match and obtain the match’s number, which is then inserted into the AF CANCEL command.

  2. **AF CANCEL(nnnnn)** The variable AOMATNUM can be used in the AF CANCEL command instead of obtaining the number of the current match by the SHOW MATCH command. This will cause AF CANCEL to cancel the current outstanding match.

  3. **CLOSE ddname FORCE**
AF CANCEL

- To provide downwards compatibility with previous versions, the AO CANCEL command functions as a command synonym for the AF CANCEL command.

**Example**

```plaintext
AF CANCEL(3)
```

This command cancels match entry number three.
AF CMD and AF NOCMD

Description
AF CMD specifies that AF/OPERATOR will monitor operator commands for possible matches with enabled traps. (Can be issued from the console.)

Syntax
```
subsys AF CMD ->
```

subsys This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

Comments
- CMD is the default. AF CMD is normally used to reverse the effect of an earlier AF NOCMD product control command. Since it is not advisable to issue AF NOCMD, there is no need to issue AF CMD.
- To provide downwards compatibility with previous versions, the AO CMD command functions as a command synonym for the AF CMD command.

NOCMD Option
AF NOCMD stops AF/OPERATOR from monitoring operator commands for possible matches with enabled traps. (Can be issued from the console.)

Caution: This command makes command traps inoperable. It is not advisable to use this command.

Syntax
```
subsys AF NOCMD ->
```

subsys This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.
**Comments**

- Since all AF/OPERATOR console commands are executed as a result of command trap action, issuing AF NOCMD will prevent AF/OPERATOR from processing such commands. It will not respond to commands issued from the console; however, it will respond to commands entered via modify. You can use the MVS MODIFY (F) command to pass commands to AF/OPERATOR using the following syntax:

  `F task,command`

  To enable console command processing after it has been disabled, issue AF CMD using the MVS MODIFY command.

  `F task,AF CMD`

- To provide downwards compatibility with previous versions, the AO NOCMD command functions as a command synonym for the AF NOCMD command.
AF COMTRACE

Description
AF COMTRACE activates or deactivates a communication trace facility that provides diagnostic information. The AF COMTRACE command can be issued from an MVS console, from a TSO simulated console, or from a REXX exec running under AF/OPERATOR.

Syntax

```
>> AF COMTRACE(action)
```

```
>> LINKID(ii) 
```

```
>> CTYPE(cc) 
```

```
>> TPNAME(tt) XCFID(GROUP(gg),(MEMBER(mm)) 
```

```
>> PORT(pppp) 
```

```
>> STYPE(rr) 
```

Parameters

COMTRACE(action)
Indicate what action the trace facility should perform. Valid values for action are.

- **ON** Activate trace facility.
- **OFF** Deactivate trace facility.
- **DISPLAY** Display trace facility status.

LINKID(ii)
Specifies the linkid of the AF/OPERATOR client. Valid values for ii are: a unique linkid, 1 to 8 characters in length. This is the same LINKID as on the LINK DEFINE command.

CTYPE(cc)
Specifies the component type to trace on the client side. More than one type can be specified. Valid cc values are:

- **ALL** Explicitly requests tracing of all components on the client side.
- **WTO** WTO component.
- **CMD** Command component.
- **EXEC** REXX exec component.
- **WTORT** WTO reroute component.
TPNAME(tt)
    Specifies the APPC transaction program name of the AF/OPERATOR server. This is the same TPNAME as on the COMM START command.

STYPE(rr)
    Specifies the component type to trace on the server side. More than one can be specified. Valid rr values are:

    ALL  Explicitly requests tracing of all components on the server side.
    WTO  WTO component.
    CMD  Command component.
    EXEC REXX exec component.
    WTORT WTO reroute component.
    USRDATA User data component.
    CMDRSP Command response component.
    WTOR WTOR component.
    DATARPLY Data reply component.
    DATARERR Data reply error component.

XCFID(GROUP(gg),MEMBER(mm))
    Identifies an AF/OPERATOR XCF server selected for tracing. The GROUP and MEMBER should match those previously specified on a COMM START command.

PORT(pppp)
    Identifies an AF/OPERATOR TCP/IP server selected for tracing. The PORT should match one previously specified on a COMM START command.
Comments

- The AF COMTRACE command can be issued from an MVS console, from a TSO simulated console, or from a REXX exec.
- You do not need to use the AF COMTRACE command, but you may be asked to perform a communication trace at the request of Candle Customer Support if you have a problem running the communication or link component.
- A COMTRACE action must be specified, there is no default.
- If no LINKID or TPNAME is specified, all communication links are traced or displayed.

Note: Tracing all links may degrade system performance for systems with many defined links.

- If a component is specified without a LINKID or a TPNAME, that component is traced for all LINKIDs or TPNAMEs.
- If a LINKID or TPNAME is specified without a component, all components associated with that LINKID or TPNAME are either traced or displayed.
- CTYPE and STYPE accept lists of components, separated by commas.
- This command uses the MVS generalized trace facility. GTF must be active for tracing to occur. The results are displayed in the GTF trace dataset.

Example

The following example activates the trace facility for a specific communication link, LINKID(RMT).

```rexx
/* REXX */
ADDRESS APHOST
"AF COMTRACE(ON) LINKID(RMT)"
IF rc <> 0 THEN SAY "COMMUNICATION TRACE FACILITY FAILED"
ELSE DO
  say "COMMUNICATION TRACE FACILITY ACTIVATED"
  "AF COMTRACE(DISPLAY)"
END
RETURN rc
```

The following example activates the trace facility for all communication links.

```rexx
/* REXX */
ADDRESS APHOST
"AF COMTRACE(ON)"
IF rc <> 0 THEN SAY "COMMUNICATION TRACE FACILITY FAILED"
ELSE
  "AF COMTRACE(DISPLAY)"
RETURN rc
```

The following example activates the trace facility for a specific component (the WTO component) of a specific link.
The following example activates the trace facility for a specific AF/OPERATOR server, identified by its APPC transaction program name, PEERTEST.

```/* REXX */
ADDRESS AF HOST
 "AF COMTRACE(ON) LINKID(RMT) CTYPE(WTO)"
 IF rc <> 0 THEN SAY "COMMUNICATION TRACE FACILITY FAILED"
 ELSE DO
   say "COMMUNICATION TRACE FACILITY ACTIVATED"
   "AF COMTRACE(DISPLAY)"
 END
RETURN rc
```

The following example activates the trace facility for a specific AF/OPERATOR server’s WTO component.

```/* REXX */
ADDRESS AFHOST
 "AF COMTRACE(ON) TPNAME(PEERTEST)"
 IF rc <> 0 THEN SAY "COMMUNICATION TRACE FACILITY FAILED"
 ELSE DO
   say "COMMUNICATION TRACE FACILITY ACTIVATED"
   "AF COMTRACE(DISPLAY)"
 END
RETURN rc
```

The following example activates tracing of the WTO component for a specific XCF server.

```/* REXX */
"AF COMTRACE(ON) XCFID(GROUP(KOGXCF1),MEMBER(SYSASRVR)) STYPE(WTO)"
```

The following example activates tracing of the WTO component for a specific TCPIP server.

```/* REXX */
"AF COMTRACE(ON) PORT(1234) STYPE(WTO)"
```
## Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Trace facility activated or deactivated successfully.</td>
</tr>
<tr>
<td>4</td>
<td>Trace facility activated or deactivated unsuccessfully.</td>
</tr>
<tr>
<td>8</td>
<td>Syntax error.</td>
</tr>
</tbody>
</table>
**AF DOM and AF NODOM**

**Description**
AF DOM starts or resumes the delete operator message (DOM) processing. (Can be issued from the console.)

**Syntax**

```
subsys AF DOM -->
```

**subsys** This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The `subsys` name is assigned by the SUBSYS startup parameter.

**Comments**
Since DOM processing is the default, use the AF DOM product control command primarily to undo the effect of an AF NODOM product control command.

**NODOM Option**
AF NODOM suspends the delete operator message (DOM) processing. (Can be issued from the console.)

**Syntax**

```
subsys AF NODOM -->
```

**subsys** This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The `subsys` name is assigned by the SUBSYS startup parameter.

**Response**
AF/OPERATOR issues messages AOP0983 and AOP0984 to warn that DOM processing has been suspended.
AF DUB_AS_PROCESS

Description
AF DUB_AS_PROCESS changes the default OS/390 UNIX System Services setting for DUB so that any new subtask is treated as a separate process. (Can be issued from the console.)

Syntax

```
subsys AF DUB_AS_PROCESS =>
```

subsys This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

This command has the same function as the AF/OPERATOR startup parameter of the same name. By default, a user of OS/390 UNIX System Services is treated as a thread within the individual process of the address space. Certain programs may require that the user execute as a separate process within a multitasking address space. This command is provided as an alternative to users that start AF/OPERATOR with SUB=MSTR. Do not issue the AF DUB_AS_PROCESS command if you have specified the DUB_AS_PROCESS startup parameter. Do not use the DUB_AS_PROCESS startup parameter when SUB=MSTR is specified.

A program invoked using the AFCALL command can fail if it uses OS/390 UNIX System Services and is unable to coexist with another active user of OS/390 UNIX System Services within the AF/OPERATOR address space.

Note:  This command should be issued before any TCP/IP links are started, since the AF/OPERATOR TCP/IP client/server tasks created using the LINK START and COMM START commands can use OS/390 UNIX System Services.
Description
AF FREE deallocates and then reallocates an AF/OPERATOR reserved sysout dataset or
DASD dataset. This makes it possible to print RKOGLOGM without shutting down
AF/OPERATOR. (Can be issued from the console.)

Syntax

```
subsys FREE(aoddname) — >>
```

- **subsys** This operand is only required when issuing the command from the console. This is
  the name of the AF/OPERATOR address space where you want the command to
  execute. The subsys name is assigned by the SUBSYS startup parameter.

- **aoddname** The ddname of the file to be freed.

Comments

- If you direct the message log to the SYSOUT dataset, AF FREE makes it possible to run
  AF/OPERATOR for long periods of time without dedicating large amounts of JES spool
  space for AF/OPERATOR’s internal datasets. Because RKOGLOGM is allocated through
  normal JCL, the first time AF FREE is issued the spool space is not purged. Subsequent
  AF FREE product control commands (RKOGLOGM) will purge the tracks.

- If you direct the message log to DASD, you can use AF FREE to deallocate the current
  dataset so you can use it for other purposes. For example, you can specify up to 99
  DASD datasets, so when one dataset fills up, the log is automatically redirected to the next
  dataset in sequence. When the last dataset fills up, the log switches back to the first one.
  So if you issue the AF FREE command, logging will automatically switch to the next
  dataset in sequence.

- You can automate the process of freeing these datasets by setting a trap for a Lines
  Exceeded message from JES and issuing an AF FREE product control command from
  that trap.

- Note that if there is severe spool space shortage in the system, AF/OPERATOR may not
  be able to reallocate an additional spool dataset until more space becomes available.
  This may temporarily halt AF/OPERATOR execution.

- The AF FREE command will search for, and free, up to 25 datasets concatenated to one
  ddname when processing a FREE(ddname) statement.

- To provide downwards compatibility with previous versions, the AO FREE command
  functions as a command synonym for the AF FREE command.
AF LOG and AF NOLOG

Description
AF LOG activates AF/OPERATOR’s message logging facility, which records AF/OPERATOR activity in the AF/OPERATOR message log. (Can be issued from the console.)

Syntax

```plaintext
AF LOG -->
```

```plaintext
subsys
```

This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The `subsys` name is assigned by the SUBSYS startup parameter.

Comments
- AF LOG is the default. AF LOG generates two messages:
  - AOP0150
  - AOP0151
- To provide downwards compatibility with previous versions, the AO LOG command functions as a command synonym for the AF LOG command.

NOLOG Option
AF NOLOG specifies that messages normally sent to the AF/OPERATOR message log will not be recorded in the log. (Can be issued from the console.)

Syntax

```plaintext
AF NOLOG -->
```

```plaintext
subsys
```

This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The `subsys` name is assigned by the SUBSYS startup parameter.

Comments
- If you issue an AF FREE (RKOGLOGM) command after issuing the AF NOLOG command, logging will recommence after the log has been freed and reallocated. You must issue another NOLOG command to stop sending messages to the AF/OPERATOR message log.
  - AF NOLOG generates two messages:
    - AOP0150
    - AOP0151
- To provide downwards compatibility with previous versions, the AO NOLOG command functions as a command synonym for the AF NOLOG command.
AF MATINFO

Description
AF MATINFO displays a message indicating the amount of CPU time used for a normally completed match. (Can be issued from the console.)

Syntax
```
subsys AF MATINFO (ON (NOWTO)) (OFF) (WTO) <>
```

`subsys` This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

`ON` Specifies that message !AOP0097 indicating the total CPU time used for a completed match will be displayed in RKOGLOGM.

`NOWTO` Directs the MATINFO message to the message log RKOGLOGM. This is the default setting.

`WTO` Directs the MATINFO message to SYSLOG. This allows it to be trapped upon.

`OFF` Specifies that no completed match message will be displayed.

Example
```
AF MATINFO(ON(WTO))
```
In the above example, this message is written to SYSLOG. This means it can be trapped by AF/OPERATOR. If you specify the NOWTO option, the message is written to AF/OPERATOR message log RKOGLOGM.
AF QUIET and AF NOQUIET

Description
AF QUIET suppresses messages from AF/OPERATOR to the operator console and SYSLOG, except responses to an operator command. The AF QUIET command allows the Quietable Messages List to be dynamically updated. (Can be issued from the console.)

Syntax

```
subsys AF QUIET (MSG (nnnn ))
```

subsys This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

MSG(nnnn) nnnn represents the number of the message affected by the operation. It can be an individual message number, a list of messages, or a range of message numbers.

Comments
- The default list of messages not suppressed by AF QUIET mode are:
  
  - !AOP0120 - !AOP0159
  - !AOP0200 - !AOP0299
  - !AOP0501 - !AOP0502
  - !AOP0505
  - !AOP0767 - !AOP0769
  - !AOP2021
  - !AOP3001 - !AOP3116
  - !AOP3230 - !AOP3239

- To provide downwards compatibility with previous versions, the AO QUIET command functions as a command synonym for the AF QUIET command.

- The SHOW QUIET/NOQUIET command lists the current messages that will not be suppressed by QUIET mode.

The following example shows a sample of this list:

```
!AOP0210 --- LIST OF QUIET MODE ......
...... EXCEPTION MESSAGES ---
!AOP0210 CURRENT MODE : NOQUIET
!AOP0210 NOQUIET MESSAGES : NN NN NN-NN ......
!AOP0210 NOQUIET MESSAGES : NNN-NNN NNNN-NNNN
```

The display is limited to approximately 13 detail lines.
NOQUIET Option

AF NOQUIET specifies that messages from AF/OPERATOR will not be suppressed. (Can be issued from the console.)

Syntax

```
| subsys | AF NOQUIET | (MSG(nnnn)) |
```

- **subsys**
  
  This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

- **MSG(nnnn)**

  nnnn represents the number of the message affected by the operation. It can be an individual message number, a list of messages, or a range of message numbers.

Comments

- NOQUIET is the default. AF NOQUIET is normally used to reverse the effect of an earlier AF QUIET product control command.
- To provide downwards compatibility with previous versions, the AO NOQUIET command functions as a command synonym for the AF NOQUIET command.
- When the MSG suboperand is specified on an AF QUIET command it will cause the particular message to be suppressed when executing in QUIET mode.
- When specified on an AF NOQUIET command, it will cause the particular message to not be suppressed when executing in QUIET mode.

Example

The following example causes messages 100, 180 through 199, 3300, and 3301 not to be suppressed by quiet mode:

```
AF NOQUIET(MSG(100 180-199 3300 3301))
```

The following example causes message 201 to be suppressed by QUIET mode:

```
AF QUIET(MSG(201))
```

A special value of ‘RESET’ will restore the original default list of messages. For example:

```
AF NOQUIET(MSG(RESET))
```
AF RECOVER and AF NORECOVER

Description
AF RECOVER enables abend recovery by AF/OPERATOR. (Can be issued from the console.)

Syntax

```
subsys AF RECOVER -->
```

`subsys` This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The `subsys` name is assigned by the SUBSYS startup parameter.

Response
When you specify AF RECOVER, AF/OPERATOR analyzes, reports on, and recovers from system and user abends.

NORECOVER Option
AF NORECOVER disables abend recovery by AF/OPERATOR. Any subsequent system or AF/OPERATOR abend will terminate AF/OPERATOR. (Can be issued from the console.)

Important: Use only at the request of Candle Customer Support.

Syntax

```
subsys AF NORECOVER -->
```

`subsys` This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The `subsys` name is assigned by the SUBSYS startup parameter.

Comments
To provide downwards compatibility with previous versions, the AO RECOVER and AO NORECOVER commands function as command synonyms for the AF RECOVER and AF NORECOVER commands.
AF RELOAD(HOSTNAME)

Description
AF RELOAD(HOSTNAME) dynamically rebuilds the trusted hostnames list from the HOSTNAME member of the RKANPAR DD.

Syntax

```
subsys AF RELOAD (HOSTNAME) <
```

subsys This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

HOSTNAME The trusted hostname list is rebuilt from the HOSTNAME member of the RKANPAR PDS.
Description

AF RXSEARCH determines REXX search order used to locate functions and subroutines that are called from an exec. (Can be issued from the console.)

*Note:* This command does not affect the search sequence for the REXX functions that are invoked in the same EXEC in which this command is issued.

Syntax

```
subsys AF RXSEARCH (LOAD)
```

**subsys**

This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The *subsys* name is assigned by the SUBSYS startup parameter.

**LOAD**

Search load libraries first. If the function or subroutine is not found, search for a REXX exec.

**REXX**

Search for a REXX exec. If the exec is not found, search the load libraries.

Example

```
AF RXSEARCH(LOAD)
```
AF SSIRESET and AF SSIDRAIN

Description
AF SSIRESET starts or restarts command (CMD), WTO, and DOM processing. (Can be issued from the console.)

Syntax
```
subsys
AF SSIRESET -><
```

Comments
Since CMD, WTO, and DOM processing are the defaults, use the AF SSIRESET product control command primarily to undo the effect of an AF SSIDRAIN product control command.

SSIDRAIN Option
AF SSIDRAIN suspends CMD, WTO, and DOM processing. (Can be issued from the console.)

Syntax
```
subsys
AF SSIDRAIN -><
```

**subsys**
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The *subsys* name is assigned by the SUBSYS startup parameter.

Response
- AF/OPERATOR issues messages AOP983 and AOP0984 to warn that CMD, WTO, and DOM processing have been suspended.
- The SSIRESET and SSIDRAIN options allow you to reset the CMD, WTO, and DOM options with a single command. These options perform no other special processing.
AF STATS

Description
The AF STATS command enables AF/OPERATOR to gather SMF data from which statistical reports can be generated. Using the AF STATS command, you can select any of the following events to monitor:
- Match start and end
- EXEC start and end
- File open, I/O, and close
- Messages suppression and rerouting

In addition, the SHOW STATS command allows you to determine if the system is collecting the SMF events you have selected.

Syntax
```
subsys AF STATS(Collect(OPT,...OPT),SMFREC(SMF#)) ->>
```

subsys
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

SMF#
This is the SMF record number to be used for the AF/OPERATOR data. The default is 200. Check with your systems programmer to determine which numbers are being collected by SMF and are not currently in use by another application.

OPT
Can be one or more of the following:

- NONE
  No SMF data will be collected.
- ALL
  All SMF data will be collected.
- MATCH
  Only MATCH data will be collected.
- EXEC
  Only EXEC data will be collected.
- FILE
  Only FILE data will be collected.
- IMMED
  Only immediate update information will be collected for message rerouting and suppression.
Comments
The options may be specified in any combination and in any order with the following considerations:

- If NONE is specified, it takes precedence over any other options specified at the same time.
- If ALL is specified, it takes precedence over any other options specified at the same time with the exception of NONE.
- We recommend that the AF/OPERATOR startup parameter contain the option of NONE and the required SMF record number. The AF command can then be used to turn SMF recording on or off as needed without having to respecify the SMF record number as shown in the examples below.

Example
The following is an example of the AF/OPERATOR startup parameter. For further information on this parameter, see the Startup parameters section of the Customization and Configuration Guide.

```
PARM:STATS(COLLECT(NONE),SMFREC(200))
```

The following example shows how to use the AF command to turn on SMF recording.

```
CMD:AF STATS(COLLECT(MATCH,FILE))
```

With the Statistics enhancement enabled, you can now use the SHOW STATS command to determine if the system is collecting the SMF events that you have specified.

```
SHOW STATS
!AOP2021 OMC GATEWAY ACTIVITY SUMMARY REPORT
!AOP2021 OMC GATEWAY ---WTO/WTOR COMMAND DOM
!AOP0280 # MONITORED : 23396 487 4514
!AOP0280 # SELECTED : 14 3 0
!AOP0280 % SELECTED : 0 0 0
!AOP0280 # SUPPRESSED : 14 3 0
!AOP0280 % SUPPRESSED : 0 0 0
!AOP0280 # ALTERED : 14 0 0
!AOP0281 # TIME-OF-DAY EVENTS TRIGGERED: 3
!AOP0280 SMF STATS(COLLECT(ALL),SMFREC(210))
!AOP0280 SMF STATS CURRENTLY: ACTIVE
```
AF STOP and AF KILL

Description
AF STOP shuts down AF/OPERATOR after all active matches are complete. (Can be issued from the console.)

Syntax

```
AF STOP — >>
```

**subsys**  
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The *subsys* name is assigned by the SUBSYS startup parameter.

AF/OPERATOR issues message AOP0999 to tell you that the product started task has ended and message AOP0004 to tell you that the product message log has stopped.

Comments
Do not use the AF STOP product control command unless you want to shut down AF/OPERATOR.

- If you want AF/OPERATOR to remain active but to take no action when traps are triggered, put AF/OPERATOR in warn mode.
- If you want to prevent AF/OPERATOR from trapping particular events, use the TRAP DISABLE command to disable those specific traps.
- To provide downwards compatibility with previous versions, the AO STOP command functions as a command synonym for the AF STOP command

AF KILL Option
AF KILL shuts AF/OPERATOR down immediately and creates a dump of the AF/OPERATOR address space. (Can be issued from the console.)
AF STOP and AF KILL

Syntax

```
subsys AF KILL -->
```

**subsys** This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

Response

The dump issues a U995 abend code.

AF/OPERATOR issues message AOP0995 to tell you that the product started task has been killed, and messages such as the following appear:

```
IEF450I taskname stepname - ABEND S000 U0995
IEF403I taskname - ENDED
```

Comments

To provide downwards compatibility with previous versions, the AO KILL command functions as a command synonym for the AF KILL command.
AF TOSTART

Description
AF TOSTART begins scheduling of restored TOD traps added by RELOAD(TODS). It is also used when you specify the TODISYNC(Y) option during initialization. The TODISYNC option lets you specify whether you want to schedule restored TOD traps immediately or delay them until you issue the AF TOSTART command. (Can be issued from the console.) See the AF/OPERATOR Configuration and Customization Guide for a description of the RELOAD and TODISYNC startup parameters.

Syntax

```
subsysAF TOSTART —>>
```

subsys This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

Response
Messages such as the following appear:

```
!AOP0062 RESTORED TOD TRAP SCHEDULING IS NOW ACTIVE.
!AOP0064 AF TOSTART IGNORED. RESTORED TOD SCHEDULING IS ALREADY ACTIVE.
```

Comments
- Use AF TOSTART only if TODISYNC(Y) was specified in the startup procedure. If restored TOD suspension was not requested or had been resumed by a prior request, the AF TOSTART command will be ignored and will issue the following warning message:

```
!AOP0063 RESTARTED TOD TRAP SCHEDULING IS SUSPENDED.
USE AF TOSTART TO RESUME
```

- Specifying the TODISYNC(Y) option postpones execution of the checkpointed (that is, those with the catchup option) TOD traps until you use AF TOSTART. Please remember: If the delay is requested, the only way to start scheduling of the restored TOD traps is by issuing the AF TOSTART command.

- To provide downwards compatibility with previous versions, the AO TOSTART command functions as a command synonym for the AF TOSTART command.
Description
AF WARN puts AF/OPERATOR into warn mode, in which no trap action is taken. (Can be issued from the console.)

AF/OPERATOR enters warn mode after all active matches are complete.

Syntax

```
subsys           AF WARN     -><
```

This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The sub sys name is assigned by the SUBSYS startup parameter.

Comments
- AF/OPERATOR can operate in one of two modes: warn mode or process mode. In warn mode, when a trap is triggered, AF/OPERATOR does only the following:
  - Monitors commands, messages, and system dates and times for triggering events.
  - Logs the match to ddname RKOGLOGM.
  - Writes a message to SYSLOG each time a match for an enabled trap occurs. No WTOs, WTO Rs, or OPERs are issued.

**Important**
When running in WARN mode, any logic in your REXX (or CFILE) command procedures which depends on the reply to a WTO will fail. Also, while such command procedures may be executed, any AF/OPERATOR commands they issue, other than the ones listed as being available in WARN mode, will fail.

Warn mode is very helpful during the initial installation and testing of AF/OPERATOR applications.
- In warn mode, there can be no route/descriptor code changes or message suppression.
- In warn mode, command traps have no effect. Consequently, most AF/OPERATOR commands are not available, since they are defined as command traps. Commands available in warn mode include:
  - AF
  - DELETE
  - EXEC
  - SHOW
  - TRAP

Use AF PROCESS to exit warn mode, or AF STOP to stop AF/OPERATOR.
To provide downwards compatibility with previous versions, the AO WARN command functions as a command synonym for the AF WARN command.

**AF PROCESS Option**

AF PROCESS puts AF/OPERATOR into process mode, in which all enabled traps function fully. (Can be issued from the console.)

**Syntax**

```
AF PROCESS -- >
```

**subsys**

This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The **subsys** name is assigned by the SUBSYS startup parameter.

**Comments**

- AF/OPERATOR can operate in one of two modes: warn mode or process mode.
- Use the AF PROCESS command if you are running in WARN mode and want to change all enabled traps to become fully functional.
- In process mode, AF/OPERATOR is fully operational. AF/OPERATOR monitors commands, messages, and the date and time for triggering events. When traps are triggered, the matches are logged in the AF/OPERATOR message log (RKOGLOGM). All trap actions are performed, both immediate actions or extended actions.
- To display AF/OPERATOR’s current operating mode, enter the command
  
  ```AF```
- To provide downwards compatibility with previous versions, the AO PROCESS command functions as a command synonym for the AF PROCESS command.
AF WTO and AF NOWTO

Description
AF WTO specifies that AF/OPERATOR will monitor messages to operators for possible matches with enabled traps. (Can be issued from the console.)

Syntax

subsys AF WTO ->

subsys This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

Comments
- Since this is the default AF/OPERATOR condition, the AF WTO product control command is principally used to undo the effect of an AF NOWTO product control command.
- To provide downwards compatibility with previous versions, the AO WTO and AO NOWTO commands function as a command synonyms for the AF WTO and AF NOWTO commands.

AF NOWTO Option
AF NOWTO stops AF/OPERATOR from monitoring messages to operators for possible matches with enabled traps. (Can be issued from the console.)

Syntax

subsys AF NOWTO ->

subsys This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.
Description
AFADDR directs commands to an alternate environment.

AFADDR is downwardly compatible with the AF/OPERATOR ADDRESS command. (See “Sample REXX Language Address” on page 25 for a discussion of the distinction between the AF/OPERATOR AFADDR command and the REXX ADDRESS command.)

Note: Unlike REXX, AF/OPERATOR has no default environment; for AF/OPERATOR to direct commands elsewhere, both a logon and an AFADDR command must be issued. Only non-AF/OPERATOR commands are redirected by this command.

Syntax

AFADDR applname command

applname The name of an application session as it was specified in a previous LOGON command.

command A command to the application session specified by applname.

Comments
The three basic forms of the AFADDR command perform three different functions:

- Directing subsequent commands to an alternate environment.
  After the command
  
  \[AFADDR \text{ applname1}\]
  
  has been issued, every subsequent line of text, that is not part of an AF/OPERATOR command, is directed to the \text{applname1} environment, until another AFADDR command is issued with the following form:
  
  \[AFADDR \text{ applname2}\]
  
  where \text{applname2} is different from \text{applname1} or AFADDR alone as described below.

- Directing subsequent commands to the previous environment.
  After an AFADDR \text{applname} command has been issued, the command
  
  \[AFADDR\]
  
  without an operand restores the previous environment. Subsequent non-AF/OPERATOR text is directed as it was before the AFADDR command was issued.

- Directing a command to an environment immediately.
Only one REXX exec can access a particular environment at a time. The environment is locked when the AFADDR command is issued and unlocked when the match ends or an AFADDR for another environment is issued as part of the same match.

The AFADDR command contains a mechanism to serialize access to an addressed environment. This feature is particularly useful in the Programmerless Open VTAM Interface (POVI) environment for responding to multiple events that occur simultaneously. For example, let’s say that a trap has been written to fire whenever a job abends and then execute the following sequence. (This example assumes that a previous sequence logged onto the POVI environment with an applname of TSOSESS and started the TSO session.)

```
AFADDR TSOSESS
\RC INFOMGMT
```

\RC is a POVI command that invokes a prerecorded POVI procedure called INFOMGMT.
When a single job abends, a match is created.

```
AFADDR TSOSESS
\RC INFOMGMT
MATCH1
Time
>>>=
```

When two jobs abend at the same time, two matches are created. Match 2 is addressing the same environment and is suspended while Match 1 finishes, at which point it continues.

```
AFADDR TSOSESS
\RC INFOMGMT
MATCH1
Time
>>>=
```

```
AFADDR TSOSESS
\RC INFOMGMT
MATCH2
(match 2 is suspended)
Time
>>>=
```

(match 2 continues when match1 ends)
Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Link is established. The AFADDR command automatically enqueues on a resource that is already in use. Unlike the LOGON command, it does not set AOLASTCC to 16 if the link is in use by another resource. Once the link is freed, the AFADDR command sets AOLASTCC to 0 and processing continues.</td>
</tr>
<tr>
<td>16</td>
<td>SSCT extension is not available, error addressing previous environment, or invalid syntax.</td>
</tr>
</tbody>
</table>

Example

Assume that at AF/OPERATOR system startup,

- The UNDFTEXT startup parameter directs lines of text that are not automation commands to the AF/OPERATOR log.
- A LOGON command assigns the applname OM1 to an OMEGAMON session.

The REXX exec ADDRTEXT follows:

```
AFADDR OM1 'MCPU01' /*Line1*/
' MCPU01' /*Line2*/
AFADDR OM1 /*Line3*/
' MCPU01' /*Line4*/
AFADDR /*Line5*/
' MCPU01' /*Line6*/
```

When ADDRTEXT is executed:

1. MCPU01 is sent to OM1, but AF/OPERATOR is unchanged.
2. MCPU01 is sent to the AF/OPERATOR log since the environment is unchanged.
3. The environment is changed to OM1.
4. MCPU01 is sent to OM1.
5. AF/OPERATOR is re-established.
6. MCPU01 is sent to the AF/OPERATOR log.
AFCALL

Description
AFCALL passes execution of a command file to a non-AF/OPERATOR program and returns execution to AF/OPERATOR when the program is completed.

Syntax

```
AFCALL program  
               PARMS('parmstring')  
               FILENAME(filename)
```

**program** Specifies the load module name.

**PARMS("parmstring")**
The parameter string that will be passed to the called program.

**FILENAME(filename)**
The filename used in the ALLOC and OPEN commands for the load module library that contains the program.

Comments
- If the FILENAME option is not specified, AF/OPERATOR searches for the load module—first in LINKLIST libraries, then in LPA.
- If a FILENAME option is specified, only the library containing the specified file is searched (not LINKLIST or LPA).
- The load module being called must reside in an APF authorized library, otherwise a system abend 306 will occur.
- The return code from the AFCALL command is the return code passed back from the called program. This return code is put into the AOLASTCC variable.

Example
The following two examples illustrate how a module that resides in LINKLIST or LPA is invoked via the AFCALL procedural instruction:

```
AFCALL MYCODE PARMS('&AOJNAME &AOGDATE &AOTIME')
```

```
AFCALL PROG1
```
The following example illustrates how a module that doesn’t reside in LINKLIST or LPA is invoked with the AFCALL procedural instruction:

```
ALLOC DD(TEMP) DSN('USER.LOAD.LIB') SHR
OPEN TEMP INPUT DDNAME(TEMP)
AFCALL PGM PARMS('PARM1, PARM2, PARM3') FILENAME(TEMP)
CLOSE TEMP
FREE DD(TEMP)
```

**Note:** The dataset which contains the module should be allocated and opened for input prior to the invocation.
**Description**

ALLOC dynamically defines and allocates a dataset or concatenates a list of datasets to one ddname.

**Syntax**

```
ALLOC DDNAME(ddname) SYSOUT(class)
```

**Options:**

- SPACE(primary quantity,secondary quantity)
- BLOCK(value)
- TRACKS/CYLINDERS
- UNIT(TYPE)
- VOLUME(VOLSER)
- RCFM( )
- LRECL( )
- BLKSIZE( )
- DSORG( )
- DIR( )
- RECALL/NORECALL

**DDNAME(ddname)**

The ddname for the dataset or list of datasets. It consists of 1 to 8 alphanumeric characters, starting with an alphabetic or national character.

**SYSOUT**

The dataset is a system output dataset.

- **class**
  
  Use this option to allocate a SYSOUT file and assign it a class. Files allocated with the SYSOUT option cannot be opened for input processing.

- **writer**
  
  Use this option to specify the member name of a program in the system library that is to write the SYSOUT dataset. This program will be used instead of JES2 or JES3.
DEST(“dest”) Specifies a remote destination or workstation to which SYSOUT data sets are to be routed.

DSN(“dsname”) Use this option to allocate a disk file and assign it a name or to allocate a list of disk files and assign them names. You can concatenate a maximum of 25 dataset names to the specified ddname. Each dataset name must be enclosed in quotes.

OLD Allocates the disk file for exclusive use.

MOD Allows you to append data to an existing disk file or to allocate a new disk file if it did not previously exist.

NEW Allocates a new disk file.

SHR Allocates the disk file for sharing with other tasks. This is the default.

RECALL If the specified dataset has been archived by a DASD storage management product, such as IBM’s DFHSM, wait for a RECALL. This is the default.

NORECALL If the specified dataset has been archived, do not wait for a RECALL.

SPACE(primary quantity, secondary quantity) Specify dataset characteristics. The following space allocation options are available.

- BLOCK(“value”)
- TRACKS
- CYLINDERS

UNIT Device type on which the specified dataset resides or will reside.

VOLUME Volume name where the specified dataset resides or will reside.

RECFM Record format of the specified dataset.

LRECL Logical record length of the specified dataset.

BLKSIZE Block size and average record length. If specified with the BLOCK keyword, then used as block size only.

DSORG Data set organization.

DIR Number of directory blocks required for a new partitioned dataset.

Comments
- You must place quotes around a dsname you are allocating with the ALLOC command.
- The options for the ALLOC command have the same meaning as the corresponding options for the TSO ALLOC command.
- Using the SYSOUT writer option precludes the need to add a special internal reader DD statement to your AF/OPERATOR startup JCL. In addition, the internal reader DD card will automatically specify the fixed record format, 80-byte LRECL and 80-byte BLKSIZE required by JES2 and JES3 for submitted JCL streams.
Note that AF/OPERATOR allocates SYSOUT datasets dynamically with the FREE=CLOSE parameter. The dataset is freed as soon as the CLOSE command is issued so you need not issue a FREE command for SYSOUT datasets. Such a command returns an error message indicating that the dataset is no longer allocated.

The ALLOC command will wait while HSM recalls the dataset unless NORECALL is specified. Since the allocation is done while running under the AF/OPERATOR main task, the entire automation task will wait for the recall, that is, no new work will be processed until the recall completes.

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Dataset allocation is successful.</td>
</tr>
<tr>
<td>12</td>
<td>Dataset name does not exist, or file with DD name is in use, or resource is already allocated as OLD, or the dataset is migrated and the ALLOC NORECALL option is specified.</td>
</tr>
<tr>
<td>16</td>
<td>Invalid syntax.</td>
</tr>
</tbody>
</table>

Example

Following are some examples of the ALLOC command as it looks when embedded in a REXX exec.

```
"ALLOC DDNAME(REPORT) SYSOUT(A)"

"ALLOC DDNAME(OUTFILE) DSN('VALID.DS.NAME') SHR"

"ALLOC DDNAME(FILE1) DSN('MY.DATA1', 'MY.DATA2', 'MY.DATA3') SHR"

"ALLOC DDNAME(INFILE) DSN('VALID.DSNAME'), "UNIT(3390) VOL(TSO001) OLD"

"ALLOC DDNAME(OUTFILE) SYSOUT(A) DEST(N1R10)"

"ALLOC DDNAME(OGIRDR) SYSOUT(X,INTRDR)"

"ALLOC DDNAME(NEWFILE) DSN('MY.NEW.DATASET'), "NEW TRACKS SPACE(15,15), "LRECL(80) BLKSIZE(6160) RECFM(F B), "DSORG(PO) DIR(5)"

"ALLOC DDNAME(DD1) DSN('MY.DATA') SHR NORECALL"
```
**BEEP**

**Description**
BEEP initiates automatic beeper paging by sending a WTOR to a console connected with AF/REMOTE. The WTOR contains the telephone number of a beeper service, the beeper (or pager) number of a person to be paged, and the text of a message. AF/REMOTE phones the beeper service, pages the designated person, and transmits the message.

**Syntax**
```
bEEP subys PHONE(‘number’) TYPE (M) PAGER(‘number’)

MESSAGE(‘message’)

REPLY(‘reply’)

ROUT(code, n:n:n:n)

NOHCOPY

HCOPY

TIMEOUT(‘reply’)
```

**subsys**
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subys name is assigned by the SUBSYS startup parameter.

**PHONE(“number”)**
The telephone number of the paging service. It may contain only numeric characters, commas, parentheses, and dashes; no embedded blanks are allowed. All non-numeric characters are ignored during the dialing.

**TYPE**
Specifies the type of device that will receive the AF/REMOTE call. The valid values are:

- **M** a modem
- **T** a telephone
BEEP

PAGER(number)

The number that identifies the person to whom the page is directed. It may contain only numeric characters and must be at least one character long.

MESSAGE("message")

A string to be transmitted to the paging service. The string can contain any printable characters, including blanks.
It can have a maximum length of 60 characters, minus the sum of the number of characters in the phone and pager numbers.

ROUT(code, s, n: n, n-n)

Specifies the routing of the WTOR that is the paging request. The value of routecode is an integer from 1 to 128. These installation-specific codes direct the WTOR to the console connected with AF/REMOTE®.

REPLY(varname)

A character variable containing the reply returned from this WTOR instruction.

NOHCOPY

Prevents output from going to the hardcopy log.

HCOPY

Sends output to the hardcopy log. This is the default.

TIMEOUT sec - Specifies the amount of time to wait before deleting the operator message. The value can be an integer from 5 to 3600.

reply - Supplies a default reply. The reply can be up to 119 characters. If no value is specified, the default is a single character blank.

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Automatic beeper paging has been initiated.</td>
</tr>
<tr>
<td>16</td>
<td>Invalid phone number, page number, or syntax.</td>
</tr>
</tbody>
</table>

Comments

BEEP can be used effectively as follows:

1. Define an AF/OPERATOR trap that, when triggered by a critical event, executes a BEEP command with a message describing the event. For example,
   
   Prod CICS Abend

2. Include in the BEEP command the beeper number of someone who should respond to the triggering event.

3. When the event occurs, the person who can respond will be paged.

Note: Modern pagers only accept alphanumeric text. “Touch Tone”® pagers only accept digits.
Example

The following is an example of this command embedded in a REXX exec.

```
"BEEP PHONE('1213-207-1400') TYPE(M) PAGER(9999)",
"MESSAGE('Prod CICS Abend') ROUT(3)"
```

The format of the WTOR generated by a BEEP command is:

```
nnAOOBEEPbccccbbbbnnnnnbbmsg
```

- Text of message
- Beep number
- Service number
- Type
- Message ID
- Reply ID
CLOSE (CL)

Description
CLOSE closes a previously opened file.

Syntax

```
CLOSE filename
```

- **filename** The name of the opened file.
- **TEMP** Performs a TYPE=T close. After a TYPE=T close, a subsequent READ or WRITE can be made without an intervening OPEN command.
  
  **Note:** CLOSE TEMP is ignored for PDS members and is not intended to be used for PDS members.

- **FORCE** Ignores match number integrity or REXX exec integrity while processing this CLOSE command.

Comments
- CL is a command synonym for CLOSE.
- You can use a variable to contain the filename or ddname.
- Use the TEMP operand to ensure that all data in memory buffers is written out and that a temporary end-of-file marker is placed at the end of the data. The FORCE operand allows you to close local and global files left open by REXX exec errors.
- AF/OPERATOR allocates SYSOUT datasets with the FREE=CLOSE parameter to dynamic allocation. The dataset is freed as soon as the CLOSE command is issued for the ddname, so you need not issue a FREE command for SYSOUT datasets. Such a command returns an error message indicating that the dataset is no longer allocated.

Example
The following is an example of this command embedded in a REXX exec. This example shows how to use the CLOSE command as part of a routine that copies the contents of one file into another.
indsn = "'VALID.INPUT.DDNAME'"
outdsn = "'VALID.OUTPUT.DDNAME'"
ADDRESS AFHOST
  "ALLOC DDNAME(IFILE) DSN(" || indsn || ") SHR"
  "ALLOC DDNAME(OFILE) DSN(" || outdsn || ") SHR"
  "OPEN IFILE INPUT GLOBAL"
  "OPEN OFILE OUTPUT GLOBAL"
iorecord=''
  status= 'GLBVPUT'('IORECORD')
  "READ IFILE IORECORD"
  status= 'GLBVGET'('IORECORD')
  DO WHILE(iorecord ^= '')
    "WRITE OFILE "||iorecord||""
    "READ IFILE IORECORD"
    status = 'GLBVGET'('IORECORD')
  END
  "CLOSE IFILE"
  "CLOSE OFILE"
  "FREE DDNAME(IFILE)"
  "FREE DDNAME(OFILE)"
ADDRESS
RETURN
**Description**

COMM manages AF/OPERATOR communications servers.

This section gives a general description of the COMM command. Its subtypes are described separately in subsequent sections.

**Syntax**

```
COMM
```

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td>Initiates an AF/OPERATOR communication server.</td>
</tr>
<tr>
<td>STOP</td>
<td>Terminates an AF/OPERATOR communication server and any client sessions.</td>
</tr>
<tr>
<td>SUSPEND</td>
<td>Temporarily disallows the specified AF/OPERATOR server or group of servers from accepting new communication sessions.</td>
</tr>
<tr>
<td>RELEASE</td>
<td>Releases temporarily suspended AF/OPERATOR servers.</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>Displays communications server information.</td>
</tr>
</tbody>
</table>
COMM DISPLAY

Description
Displays communication services information.

Syntax

```plaintext
>> COMM DISPLAY
    TYPE(protocol)
>>
```

```plaintext
>>
TPNAME(tpname)
    GROUP(xcf-group-name)
    MEMBER(xcf-member-name)
    PORTNUMBER(portnumber)
<
```

```plaintext
> <
DETAIL(level)
```

Parameters

**TYPE(protocol)**
Identifies the server to display by specifying the communication protocol type.

**TPNAME(tpname)**
Identifies the server to display by specifying its APPC transaction program name. The maximum length of `tpname` is 64 characters.

**GROUP(xcf-group-name)**
Specifies the XCF group name. Identifies which particular server to display and must be used in conjunction with MEMBER. The maximum length of `xcf-group-name` is 8 characters. An XCF group name can consist of alpha-numeric and national characters ($, #, and @). Group names starting with the letters A through I or the character string SYS or name UNDESIG are reserved.

**MEMBER(xcf-member-name)**
Specifies the XCF member name. Identifies which particular server to display and must be used in conjunction with GROUP. The maximum length of `xcf-member-name` is 8 characters. An XCF member name can consist of alpha-numeric and national characters ($, #, and @).

*Note*: Avoid using member names starting with the # symbol. AF/OPERATOR reserves a special member name of #INTERNL for its “Internal Server”. Avoid using member names that conflict with the names generated by XCF. (These start with M and have a numeric suffix; for example M1234.)

**PORTNUMBER(portnumber)**
COMM DISPLAY

Identifies which particular TCP/IP server to display. The port number can be any integer greater than or equal to 1025 and less than or equal to 32767.

DETAIL(level)

Identifies the level of information to be displayed and permits you to reduce the number of output lines produced.

BRIEF Displays minimal server information. No connector information is displayed.

SERVER Displays server information only. No connector information is displayed.

LOGONS Displays server information plus connector information and a total receive count.

COUNTS The default, displays server information plus connector information and non-zero receive counts per datatype.

FULL Displays server information plus connector information and all receive counts per datatype.

Comments

- You can display information for all servers by issuing a COMM DISPLAY command without specifying any further qualification.
- You can also use the command COMM DISPLAY DETAIL(SERVER).
- Specifying only the TYPE parameter displays information for all servers of specified protocol.
- When TYPE is not specified, the default will be determined by which other protocol specific keywords are present.
- The keywords for a given communications protocol are mutually exclusive with the keywords of any other communications protocol.
Example
The following is an example of how to display information for all communications servers:

/* REXX */
"COMM DISPLAY"

The following is an example of how to display information for a particular TCP/IP server:

/* REXX */
"COMM DISPLAY TYPE(TCPIP) PORTNUMBER(1111)"

The following is an example of how to display information for all XCF communication servers:

/* REXX */
"COMM DISPLAY TYPE(XCF)"

A typical response to the COMM DISPLAY TYPE(APPC) command follows:

COMM DISPLAY TYPE(APPC)
!AOP3765 SERVER TPNAME(TEST1) LLUNAME(HIGHWAY1) ACTIVE
!AOP3768 ---- CONNECT_EXEC - **NONE**
!AOP3768 ---- TOTAL CONNECTIONS - 1
!AOP3768 ****************************************

/* REXX */
"COMM DISPLAY TYPE(TCPIP)"

COMM DISPLAY TYPE(TCPIP) PORTNUMBER(1111)

/* REXX */
"COMM DISPLAY TYPE(XCF)"

COMM DISPLAY TYPE(XCF)
A typical response to the COMM DISPLAY TYPE(XCF) command follows:

```
COMM DISPLAY TYPE(XCF)
!AOP3765 SERVER GROUP(KOGXCF01) MEMBER(SYSBSRVR) TYPE(XCF) ACTIVE
!AOP3768 ---- CONNECT_EXEC - **NONE**
!AOP3768 ---- TOTAL CONNECTIONS - 1
!AOP3768 ******************************************************
!AOP3743 ---- SERVER MEMBER NAME - SYSBSRVR
!AOP3743 ---- PARTNER MEMBER NAMES - M1197/M1196
!AOP3743 ---- SERVER - SYSB (LNKB )
!AOP3743 ---- INTERNAL LINK ID - #0000002
!AOP3743 ---- CONNECTION ID - #A000002
!AOP3743 ---- INTERNAL SEND ONLY ID - #0000001
!AOP3743 ---- TOTAL RECEIVE COUNT - 61
!AOP3743 ---- ID COUNT - 1
!AOP3743 ---- EXEC COUNT - 60
!AOP3743 *******************************************
```

A typical response to the COMM DISPLAY TYPE(TCPIP) command follows:

```
COMM DISPLAY TYPE(TCPIP)
!AOP3765 SERVER HOSTNAME(1.2.3.4) PORTNUMBER(7890) TYPE(TCPIP) ACTIVE
!AOP3768 ---- CONNECT_EXEC - **NONE**
!AOP3768 ---- TOTAL CONNECTIONS - 1
!AOP3768 ******************************************************
!AOP3743 ---- CLIENT HOSTNAME - 111.112.113.114
!AOP3743 ---- SERVER - SYSB (LNKT )
!AOP3743 ---- CLIENT - SYSG
!AOP3743 ---- INTERNAL LINK ID - #0000002
!AOP3743 ---- CONNECTION ID - #A000002
!AOP3743 ---- TOTAL RECEIVE COUNT - 33
!AOP3743 ---- ID COUNT - 1
!AOP3743 ---- WTO COUNT - 22
!AOP3743 ---- EXEC COUNT - 7
!AOP3743 ---- MLWTO COUNT - 1
!AOP3743 ---- CMDRSP QUEUED COUNT - 2
!AOP3743 ******************************************************
```
Following are examples at other levels of detail.

To display all defined servers:

```rexx
/* REXX */
"COMM DISPLAY DETAIL(BRIEF)"
```

This command yields the results:

```
!AOP3765 SERVER TPNAME(TEST1) LLUNAME(OGAPITG1) ACTIVE
!AOP3765 SERVER GROUP(KOGXCF01) MEMBER(#INTERNL) TYPE(XCF) ACTIVE
!AOP3765 SERVER GROUP(KOGXCF01) MEMBER(SYSBSRVR) TYPE(XCF) ACTIVE
!AOP3765 SERVER HOSTNAME(10.11.12.13) PORTNUMBER(7890) TYPE(TCPIP) ACTIVE
!AOP3765 SERVER HOSTNAME(10.11.12.13) PORTNUMBER(9999) TYPE(TCPIP) ACTIVE

Alternatively, for additional information relating to a server:

```rexx
/* REXX */
"COMM DISPLAY DETAIL(SERVER)"
```

This command yields the results:

```
!AOP3765 SERVER TPNAME(TEST1) LLUNAME(OGAPITG1) ACTIVE
!AOP3768 ---- CONNECT_EXEC - **NONE**
!AOP3768 ---- TOTAL CONNECTIONS - 2
!AOP3768 ***********************************************
!AOP3766 SERVER GROUP(KOGXCF01) MEMBER(#INTERNL) TYPE(XCF) ACTIVE
!AOP3768 ---- CONNECT_EXEC - **NONE**
!AOP3768 ---- TOTAL CONNECTIONS - 1
!AOP3768 ***********************************************
!AOP3766 SERVER GROUP(KOGXCF01) MEMBER(SYSBSRVR) TYPE(XCF) ACTIVE
!AOP3768 ---- CONNECT_EXEC - **NONE**
!AOP3768 ---- TOTAL CONNECTIONS - 15
!AOP3768 ***********************************************
!AOP3765 SERVER HOSTNAME(10.11.12.13) PORTNUMBER(7890) TYPE(TCPIP) ACTIVE
!AOP3768 ---- CONNECT_EXEC - **NONE**
!AOP3768 ---- TOTAL CONNECTIONS - 11
!AOP3768 ***********************************************
```
To display connector information for a server:

```/* REXX */
"COMM DISPLAY DETAIL(LOGONS)"
```

This command yields the results:

```
!AOP3765 SERVER HOSTNAME(10.11.12.13) PORTNUMBER(7890) TYPE(TCPIP)
ACTIVE
!AOP3768 ---- CONNECT_EXEC - **NONE**
!AOP3768 ---- TOTAL CONNECTIONS - 2
!AOP3768 ***********************************************

!AOP3743 ---- CLIENT HOSTNAME - 10.11.12.14
!AOP3743 ---- SERVER - SYSBOGBB(LINKSYSB)
!AOP3743 ---- CLIENT - SYSAOGAA
!AOP3743 ---- INTERNAL LINK ID - #0000008
!AOP3743 ---- CONNECTION ID - #A000010
!AOP3743 ---- TOTAL RECEIVE COUNT - 1

!AOP3743 ***********************************************

!AOP3743 ---- CLIENT HOSTNAME - 10.11.12.15
!AOP3743 ---- SERVER - SYSBOGBB(TEST)
!AOP3743 ---- CLIENT - SYSGOGGG
!AOP3743 ---- INTERNAL LINK ID - #0000006
!AOP3743 ---- CONNECTION ID - #A000007
!AOP3743 ---- TOTAL RECEIVE COUNT - 358
!AOP3743 ***********************************************
```
To display all receive counts for all connections:

```rexx
/* REXX */
"COMM DISPLAY DETAIL(FULL)"
```

This command yields the results:

```
!AOP3765 SERVER GROUP(KOGXCF01) MEMBER(SYSBSRVR) TYPE(XCF) ACTIVE
!AOP3768 ---- CONNECT_EXEC - **NONE**
!AOP3768 ---- TOTAL CONNECTIONS - 1
!AOP3768 *******************************************************************
!AOP3743 ---- SERVER MEMBER NAME - SYSBSRVR
!AOP3743 ---- PARTNER MEMBER NAMES - M2790/M2787
!AOP3743 ---- SERVER - SYSBOGBB(OGBB )
!AOP3743 ---- CLIENT - SYSZOGZZ
!AOP3743 ---- INTERNAL LINK ID - #0000002
!AOP3743 ---- CONNECTION ID - #A000002
!AOP3743 ---- TOTAL RECEIVE COUNT - 25
!AOP3743 ---- ID COUNT - 1
!AOP3743 ---- WTO COUNT - 3
!AOP3743 ---- CMD COUNT - 0
!AOP3743 ---- EXEC COUNT - 8
!AOP3743 ---- WTOR COUNT - 1
!AOP3743 ---- WTORT COUNT - 1
!AOP3743 ---- CMDRT COUNT - 2
!AOP3743 ---- MLWTO COUNT - 3
!AOP3743 ---- CMDRSP COUNT - 5
!AOP3743 ---- WTORDEL COUNT - 0
!AOP3743 ---- CMDRSPRT COUNT - 0
!AOP3743 ---- CMDRSPRT COUNT - 0
!AOP3743 ---- DATARPLY COUNT - 0
!AOP3743 ---- DATARPER COUNT - 0
!AOP3743 ---- SLF DATA COUNT - 0
!AOP3743 ---- UNKNOWN COUNT - 0
!AOP3743 ---- WTO QUEUED COUNT - 0
!AOP3743 ---- CMD QUEUED COUNT - 0
!AOP3743 ---- EXEC QUEUED COUNT - 0
!AOP3743 ---- WTOR QUEUED COUNT - 0
!AOP3743 ---- WTORT QUEUED COUNT - 0
!AOP3743 ---- CMDRT QUEUED COUNT - 0
!AOP3743 ---- MLWTO QUEUED COUNT - 0
!AOP3743 ---- CMDRSP QUEUED COUNT - 0
!AOP3743 ---- WTORDEL QUEUED COUNT - 0
!AOP3743 ---- CMDRSPRT QUEUED COUNT - 0
!AOP3743 ---- DATARPLY QUEUED COUNT - 0
!AOP3743 ---- DATARPER QUEUED COUNT - 0
```

The `COMM DISPLAY` command displays both completed and queued counts for the various communication tasks managed by AF/OPERATOR.
A high queued count, anything over 1000, indicates a communication problem. A warning message is issued if this occurs.

COMM DISPLAY output is routed to the SYSLOG and to the AF/OPERATOR message log.

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Communication display successful.</td>
</tr>
<tr>
<td>2701</td>
<td>Communication display unsuccessful.</td>
</tr>
<tr>
<td>2716</td>
<td>Display processing abended.</td>
</tr>
</tbody>
</table>
**COMM RELEASE**

**Description**
Releases temporarily suspended AF/OPERATOR servers.

**Syntax**

```
COMM RELEASE

>> TYPE(protocol)

TPNAME(tpname)

GROUP(xcf-group-name)
MEMBER(xcf-member-name)
PORTNUMBER(listening-port-number)
```

**Parameters**

**TYPE(protocol)**
Identifies the servers to release by specifying the communication protocol used.

**TPNAME(tpname)**
Identifies the server to release by specifying its APPC transaction program name. The maximum length of `tpname` is 64 characters.

**GROUP(xcf-group-name)**
Specifies the XCF group name. Identifies which particular server to release. It must be used in conjunction with MEMBER. The maximum length of `xcf-group-name` is 8 characters. An XCF group name can consist of alpha-numeric and national characters ($, #, and @). Group names starting with the letters A through I or the character string SYS or name UNDESIG are reserved.

**MEMBER(xcf-member-name)**
Specifies the XCF member name. Identifies which particular server to release. It must be used in conjunction with GROUP. The maximum length of `xcf-member-name` is 8 characters. An XCF member name can consist of alpha-numeric and national characters ($, #, and @).

**Note:** Avoid using member names starting with the # symbol. AF/OPERATOR reserves a special member name of #INTERNL for its “Internal Server”. Avoid using member names that conflict with the names generated by XCF. (These start with M and have a numeric suffix; for example M1234.)
PORTNUMBER(listening-port-number)

Identifies which particular TCP/IP server to release. The port number can be any integer greater than or equal to 1025 and less than or equal to 32767.

Comments

- You can release all temporarily suspended servers by issuing a COMM RELEASE command without specifying any further qualifications.
- When TYPE is not specified, the default will be determined by which other protocol specific keywords are present.
- Specifying only the TYPE parameter releases all suspended servers of the specified protocol.
- The keywords for a given communications protocol are mutually exclusive with the keywords of any other communications protocol.

Example

The following example releases the server identified by APPC transaction program named PEERTST:

```rexx
/* REXX */
ADDRESS AFHOST
"COMM RELEASE TPNAME(PEERTST)"
IF rc <> 0 THEN SAY "RELEASE FAILED FOR SERVER"
ELSE DO
   "COMM DISPLAY"
   END
RETURN rc
```

The following example releases all servers:

```rexx
/* REXX */
ADDRESS AFHOST
"COMM RELEASE"
IF rc <> 0 THEN SAY "RELEASE FAILED FOR SERVER"
ELSE DO
   "COMM DISPLAY"
   END
RETURN rc
```

The following example releases a specific XCF communications server:

```rexx
/* REXX */
"COMM RELEASE TYPE(XCF) GROUP(KOGXCF01) MEMBER(SYSASRVR)"
```

In the above example, we attempt to release server member SYSASRVR in the XCF Group called KOGXCF01.
The following is an example of how to release a specific TCP/IP communications server:

/* REXX */
"COMM RELEASE TYPE(TCPIP) PORTNUMBER(1111)"

In the above example, we attempt to release a TCP/IP server listening on Port 1111.

**Return Codes**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Communication released successfully.</td>
</tr>
<tr>
<td>2601</td>
<td>No server active for one or more selected protocols.</td>
</tr>
<tr>
<td>2602</td>
<td>Operation not permitted against ’Internal Server’.</td>
</tr>
<tr>
<td>2616</td>
<td>Release processing abended.</td>
</tr>
</tbody>
</table>
Description
Initiates a communication server controlled by AF/OPERATOR.

Syntax

```
>> COMM_START TYPE(protocol) |

>> SYMDEST(symdest) |
    TPNAME(tpname) |
    LLUNAME(lluname) |
    GROUP(xcf-group-name) |
    MEMBER(xcf-member-name) |
    PORTNUMBER(listening-port-number) |
    AFFACKET(ON/OFF) |

>> CONNECT_EXEC(execname) <<
```

Parameters

**TYPE(protocol)**
Specifies the type of communication protocol to use for this server’s sessions. The supported protocols are TYPE(APPC), TYPE(XCF), and TYPE(TCPIP). If you do not supply a specified TYPE, the default protocol will be determined by which other protocol-specific keywords are present.

**SYMDEST(symdest)**
Specifies the symbolic destination name which identifies the server. The maximum length of symdest is 8 characters.

**TPNAME(tpname)**
Specifies the transaction program name which identifies this server. The maximum length of tpname is 64 characters.

**LLUNAME(lluname)**
Specifies the logical unit name which identifies this server. This is the NOSCHED type LU 6.2 defined to APPC/MVS and VTAM. The maximum length of lluname is 8 characters.

**GROUP(xcf-group-name)**
Specifies the XCF group name. It must be using in conjunction with MEMBER to identify the server. The maximum length of xcf-group-name is 8 characters. An XCF group name can consist of alpha-numeric and national characters ($, #, and &). Group names starting with the letters A through I or the character string SYS or UNDESIG are reserved.

MEMBER(xcf-member-name)

Specifies the XCF member name. It must be used in conjunction with GROUP to identify the server. The maximum length of xcf-member-name is 8 characters. An XCF member name can consist of alpha-numeric and national characters ($, #, and @).

Note: Avoid using member names starting with the # symbol. AF/OPERATOR reserves a special member name of #INTERNL for its “Internal Server”. Avoid using member names that conflict with the names generated by XCF. (These start with M and have a numeric suffix; for example M1234.)

PORTNUMBER(listening-port-number)

Specifies the port number that this TCP/IP communications server will be listening on for new client connections. The port number can be any integer greater than or equal to 1025 and less than or equal to 32767.

AFPACKET(ON/OFF)

Enables or disables the AF packet header on connections made to this TCP/IP port. The default is ON.

Notes:
- For additional information, see “Open TCP/IP Communications” in the AF/OPERATOR User’s Guide section called “Advanced TCP/IP Communications.”
- If AFPACKET(OFF) is specified, the CONNECT_EXEC match is the only match able to issue the COMSDRCV REXX function SEND and RECEIVE requests over an inbound connection, until that exec relinquishes ownership by means of the COMADMIN REXX function.
- If AFPACKET(OFF) is specified, and if the CONNECT_EXEC match does not relinquish ownership of a connection, the connection is terminated when the CONNECT_EXEC match ends.

CONNECT_EXEC(execname)

Specifies that the named exec is to be run as a COM type match and global variables containing connection information are available.

Optional for all protocol types except when using the AFPACKET(OFF) option on a TCP/IP server. In this case, specification of CONNECT_EXEC(execname) is mandatory.

Comments
- If you specify TPNAME or LLUNAME parameters in addition to a SYMDEST parameter, the TPNAME or LLUNAME specification overrides the value obtained from the side information dataset. Information on defining side information datasets can be found in the IBM document, MVS/ESA V4 Planning: APPC Management. This allows you to dynamically change server definitions for testing purposes.
When type is not specified, the default will be determined by which other protocol specific keywords are present.

The keywords for a given communications protocol are mutually exclusive with the keywords of any other communications protocol.

For details about communications sessions, please refer to Session Types in the AF/OPERATOR User’s Guide.

Example

The following example starts the server identified by APPC transaction program name PEERTEST and local LU name HIGHWAY1:

```rexx
/* REXX */
ADDRESS AFHOST
"COMM START TPNAM(PEERTEST) LLUNAME(HIGHWAY1)"
IF rc <> 0 THEN SAY "SERVER START FAILED"
ELSE DO
   "COMM DISPLAY"
END
RETURN rc
```

The following example starts a server identified by the symbolic destination name APPCTST:

```rexx
/* REXX */
ADDRESS AFHOST
"COMM START SYMDEST(APPCTST)"
IF rc <> 0 THEN SAY "SERVER START FAILED"
ELSE DO
   "COMM DISPLAY"
END
RETURN rc
```

The following command starts an XCF communications server:

```rexx
/* REXX */
"COMM START TYPE(XCF) GROUP(KOGXCF01) MEMBER(SYSASRVR)"
```

In the above example, we attempt to start server member SYSASRVR in the XCF Group called KOGXCF01.

The following example attempts to start a TCP/IP communications server. It starts a listening socket on port 1111:

```rexx
/* REXX */
"COMM START TYPE(TCPIP) PORTNUMBER(1111)"
```
## Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Server started successfully.</td>
</tr>
<tr>
<td>2301</td>
<td>Missing required subparameter(s).</td>
</tr>
<tr>
<td>2302</td>
<td>Server is already active.</td>
</tr>
<tr>
<td>2303</td>
<td>If TYPE is APPC, either the IBM APPC load library SYS1.CSSLIB is not specified or the minimum requirement of MVS/ESA Version 4.3 or higher has not been met. If TYPE is TCPIP, the minimum requirement of TCP/IP Version 3 Release 2 for MVS has not been met.</td>
</tr>
<tr>
<td>2304</td>
<td>Attach failure for APPC server component. Contact Candle Customer Support.</td>
</tr>
<tr>
<td>2305</td>
<td>Server component failure. Check SYSLOG for accompanying messages. If TYPE is APPC, ensure that the LUs are defined correctly to APPC and are active under APPC.</td>
</tr>
<tr>
<td>2316</td>
<td>Start processing abended.</td>
</tr>
</tbody>
</table>
COMM STOP

Description
Terminates an AF/OPERATOR-controlled communication server and any client sessions.

Syntax

```plaintext
>> COMM STOP
  TYPE(protocol)   -->

>> TPNAME(tpname)
  GROUP(xcf-group-name) MEMBER(xcf-member-name) PORTNUMBER(portnumber)

<< CID(connectionID)
```

Parameters

**TYPE(protocol)**
Identifies the servers to stop by specifying the communication protocol type.

**TPNAME(tpname)**
Identifies the server to stop by specifying its APPC transaction program name. The maximum length of `tpname` is 64 characters.

**GROUP(xcf-group-name)**
Specifies the XCF group name. Identifies which particular server to stop and must be used in conjunction with MEMBER.

**MEMBER(xcf-member-name)**
Specifies the XCF member name. Identifies which particular server to stop and must be used in conjunction with GROUP. The maximum length of `xcf-member-name` is 8 characters. An XCF member name can consist of alpha-numeric and national characters ($, #, and @).

**PORTNUMBER(listening-port-number)**
Identifies which particular TCP/IP server to stop. The port number can be any integer greater than or equal to 1025 and less than or equal to 32767.

**Note:** Avoid using member names starting with the `#` symbol. AF/OPERATOR reserves a special member name of `#INTERNL` for its “Internal Server”. Avoid using member names that conflict with the names generated by XCF. (These start with `M` and have a numeric suffix; for example `M1234`.)
**CID(connection-ID)**

Identifies which particular connection to a server is to be stopped. This identifier can be used with other existing qualifiers.

**Comments**

- You can stop all active servers by issuing a COMM STOP command without specifying any further qualifications.
- Specifying only the TYPE parameter stops all servers of the specified protocol.
- When TYPE is not specified, the default will be determined by which other protocol specific keywords are present.
- The TPNAME can be taken directly from the TPNAME specified on the COMM START command or, if a SYMDEST was specified on the COMM START command, you can issue a COMM DISPLAY command, which displays the TPNAME on the first line.
- When an AF/OPERATOR server is terminated, the server’s client AF/OPERATOR sessions are terminated immediately if heartbeat checking is in effect. If heartbeat checking is not in effect, client sessions can not detect that the session is down until the next request is sent from client to server.
- The keywords for a given communications protocol are mutually exclusive with the keywords of any other communications protocol.

**Example**

The following example stops the server identified by APPC transaction program named PEERTEST:

```rexx
/* REXX */
ADDRESS AFHOST
"COMM STOP TPNAME(PEERTEST)"
IF rc <> 0 THEN SAY "STOP FAILED FOR SERVER"
ELSE DO
   "COMM DISPLAY"
END
RETURN rc
```

The following example stops all servers:

```rexx
ADDRESS AFHOST
"COMM STOP"
IF rc <> 0 THEN SAY "STOP FAILED FOR SERVER"
ELSE DO
   "COMM DISPLAY"
END
RETURN rc
```

The following example stops an XCF communications server:
COMM STOP

/* REXX */
"COMM STOP TYPE(XCF) GROUP(KOGXCF01) MEMBER(SYSASRVR)"

In the above example, we attempt to stop server member SYSASRVR in the XCF Group called KOGXCF01.

The following is an example of how to stop a TCP/IP communications server:

/* REXX */
"COMM STOP TYPE(TCPIP) PORTNUMBER(1111)"

The TCP/IP server, which has been listening on port number 1111, is terminated.

Following are some examples of the use of CID with COMM STOP:

COMM STOP CID(xyz)

or

COMM STOP CID(xyz) TYPE(protocol)

or

COMM STOP CID(xyz) TYPE(TCPIP) PORT(nnnnn)

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Communication stopped successfully.</td>
</tr>
<tr>
<td>2401</td>
<td>No server active for one or more specified protocols.</td>
</tr>
<tr>
<td>2402</td>
<td>Operation not permitted against ‘Internal Server’.</td>
</tr>
<tr>
<td>2416</td>
<td>Stop processing abended.</td>
</tr>
<tr>
<td>2904</td>
<td>Invalid protocol requested.</td>
</tr>
<tr>
<td>2908</td>
<td>No sessions were active for this server. This is an informational message only.</td>
</tr>
<tr>
<td>2912</td>
<td>Not permitted against ‘Internal Server’ sessions.</td>
</tr>
</tbody>
</table>
COMM SUSPEND

Description
Temporarily disallows the specified AF/OPERATOR server, or group of servers, from accepting new communication sessions.

Syntax

```
>> COMM SUSPEND
   TYPE(protocol)
>><
   TPNAME(tpname)
   GROUP(xcf-group-name)
   MEMBER(xcf-member-name)
   PORTNUMBER(listening-port-number)
```

Parameters

**TYPE(protocol)**
Identifies the servers to suspend by specifying the communication protocol used.

**TPNAME(tpname)**
Identifies the server to suspend by specifying its APPC transaction program name. The maximum length of `tpname` is 64 characters.

**GROUP(xcf-group-name)**
Specifies the XCF group name. Identifies which particular server to suspend and must be used in conjunction with MEMBER. The maximum length of `xcf-group-name` is 8 characters. An XCF group name can consist of alpha-numeric and national characters ($, #, and @).

*Note*: Avoid using member names starting with the `#` symbol. AF/OPERATOR reserves a special member name of `#INTERNL` for its “Internal Server”. Avoid using member names that conflict with the names generated by XCF. (These start with `M` and have a numeric suffix; for example `M1234`.)

**MEMBER(xcf-member-name)**
Specifies the XCF member name. Identifies which particular server to suspend. It must be used in conjunction with GROUP. The maximum length of `xcf-member-name` is 8 characters. An XCF member name can consist of alpha-numeric and national characters ($, #, and @).

*Note*: Avoid using member names starting with the `#` symbol. AF/OPERATOR reserves a special member name of `#INTERNL` for its “Internal Server”. Avoid using member names that conflict with the names generated by XCF. (These start with `M` and have a numeric suffix; for example `M1234`.)
COMM SUSPEND

PORTNUMBER(listening-port-number)
Identifies which particular TCP/IP server to suspend. The port number can be any integer greater than or equal to 1025 and less than or equal to 32767.

Comments
- You can suspend all active servers by issuing a COMM SUSPEND command without specifying any further qualification.
- When TYPE is not specified the default will be determined by which other protocol specific keywords are present.
- Specifying only the TYPE parameter suspends all servers of the specified protocol.
- The keywords for a given communications protocol are mutually exclusive with the keywords of any other communications protocol.

Example
The following example suspends the server identified by APPC transaction program name PEERTEST:

```/* REXX */
ADDRESS AFHOST
"COMM SUSPEND TPNAME(PEERTEST)"
IF rc <> 0 THEN SAY "SUSPEND FAILED FOR SERVER"
ELSE DO
  "COMM DISPLAY"
END
RETURN rc
```

The following example suspends all servers:

```/* REXX */
ADDRESS AFHOST
"COMM SUSPEND"
IF rc <> 0 THEN SAY "SUSPEND FAILED FOR SERVER"
ELSE DO
  "COMM DISPLAY"
END
RETURN rc
```

The following suspends all XCF communication servers:

```/* REXX */
"COMM SUSPEND TYPE(XCF)"
```

The following is an example of how to suspend all TCP/IP communication servers:
/* REXX */
"COMM SUSPEND TYPE(TCPIP)"

## Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Communication suspended successfully.</td>
</tr>
<tr>
<td>2501</td>
<td>No server active for one or more selected protocols.</td>
</tr>
<tr>
<td>2502</td>
<td>Operation not permitted against ‘Internal Server’.</td>
</tr>
<tr>
<td>2516</td>
<td>Suspend processing abended.</td>
</tr>
</tbody>
</table>
**Description**

CP issues a VM operator command to the VM host from a guest MVS system console. (Can be issued from the console.)

**Syntax**

```plaintext
CP <command>
```

“**command**” Specifies the VM command. The command is issued to VM as if it were issued from your VM console.

**Return Codes**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>VM operator command is issued to VM host.</td>
</tr>
<tr>
<td>4</td>
<td>Not running under VM.</td>
</tr>
<tr>
<td>8</td>
<td>Not APF-authorized.</td>
</tr>
<tr>
<td>16</td>
<td>Invalid syntax.</td>
</tr>
</tbody>
</table>

**Comments**

- The specified VM command executes as if it were issued from the virtual machine console under which the current operating system is running. Responses to the command are displayed on the MVS console from which this CP command was issued.
- Responses returned from VM are put into the global variable AOCPL<nnn>. The characters <nnn> are sequence numbers beginning with 001.
  
  One variable is created for each line returned from VM. The end of the return string is flagged by the first null variable.
  
  The variable AOCPL# stores the number of lines returned.
- Because CP executes at a level above MVS, when MVS is executing in the virtual machine environment, CP is not restricted by the MVS statement of integrity. For example, if you use the CP command to store data into virtual machine storage you may alter the contents of storage and inadvertently cause MVS to crash.
  
  Use of CP commands can be controlled through standard VM command-limiting facilities.

**Example**

The following are examples of this command embedded in a REXX exec.

```
"CP 'QUERY NAMES"
"CP 'VARY ONLINE 14F"
```
DELETE (DEL)

Description
DELETE deletes system variables that do not begin with AO.

Syntax

```
subsys          DELETE         SYSVAR(pattern,s)
```

- **subsys**: This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The `subsys` name is assigned by the SUBSYS startup parameter.
- **SYSVAR(pattern,s)**: Deletes any variable(s) with name(s) matched by one of the patterns. You can use up to 32 patterns.

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Delete SYSVAR is not allowed from a REXX program.</td>
</tr>
<tr>
<td>0</td>
<td>Delete is successful.</td>
</tr>
<tr>
<td>12</td>
<td>System variable not found.</td>
</tr>
<tr>
<td>16</td>
<td>Invalid syntax.</td>
</tr>
</tbody>
</table>

Comments

- DEL is a command synonym for DELETE.
- You cannot delete or reassign system variables that begin with AO.
- When the DELETE command is issued from REXX, the SYSVAR operand is not valid.
DOM

Description
DOM enables deletion of a non-scrollable WTO message.

Syntax

```
DOM number
```

`number` The number of the message to be deleted. You must specify this number.

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Non-scrollable WTOs are deleted.</td>
</tr>
<tr>
<td>16</td>
<td>Invalid syntax.</td>
</tr>
<tr>
<td>Other</td>
<td>Issued by IBM DOM macro (entered through SVC).</td>
</tr>
</tbody>
</table>

Comments

- MVS assigns a unique number to each console message. The WTO NO keyword of the WTO command contains this unique number. Use this number in conjunction with the DOM command to change a non-scrollable message to a scrollable message.
- AF/OPERATOR sets the match variable AOWTONO to the WTO number of any message it traps. You can use this match variable as the DOM command operand to specify the number of the trapped message to be deleted.

For example, if a WTO trap executes a REXX exec when triggered, the command

```
DOM WTONO
```

in that REXX exec deletes the trapped message.
Example

The following is an example of this command embedded in a REXX exec. This example shows the use of the DOM command as part of a routine that requests the response to a WTO.

```
ADDRESS 'AFHOST'

"WTO '||replyno||'--Should CICS1 be shutdown? Reply Y or N'",
"WTONO(WTONO) DESC(2) NOMSGID"

"WAIT CMD('R' ||replyno||'"YN',")
    "R " ||replyno||'"YN',")
    "REPLY " ||replyno||'"YN',")
    "\" ||replyno||'"YN')")
    "SUPPRESS SECONDS(30)"

status='GLBVGET'('WTONO')
"DOM" WTONO
ADDRESS
```
EXEC (EX)

Description
EXEC executes a REXX exec. (Can be issued from the console.)

Syntax

```
subsys EXEC filename 'parm string,s'
```

**subsys**  This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The **subsys** name is assigned by the SUBSYS startup parameter.

**filename**  The name of the REXX exec to be executed.

**“parm string,s”**  A parameter to be passed to the REXX exec.

If you want to use LIST or TEST as a parameter, you can try either of the following methods:

Provide a second parameter, for example,

```
“EXEC exec name ’’TEST DUMMY’’ “
```

Add blanks or other characters to the keyword, for example,

```
“EXEC exec name ’’TEST’’ “
```

The following process occurs if you have used the REXXARGS(AF) startup option:

REXX execs receive parameters as separate argument strings.

The following process occurs if you have used the REXXARGS(TSO) startup option:

REXX execs receive parameters delimited by blanks as a single parameter. Therefore, the string of words within the quotes will be passed as one parameter whether or not there are spaces between words.

**Note:** If you want to separate parameters, use the PARSE ARG statement in the REXX exec as follows:

```
PARSE ARG argstring  
PARSE ARG argstring arg1 arg2 arg3 arg4 arg5 . . .
```

For more information on this startup option, see “Differences in Passing Parameters” on page 38.
Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Procedure exists.</td>
</tr>
<tr>
<td>16</td>
<td>No member name is specified; invalid member name is specified; member is not found in command library; member has not records; or BLDL or I/O error.</td>
</tr>
</tbody>
</table>

Comments

- EX is a command synonym for EXEC.
- If `filename` is a REXX exec, then the first line of the exec must be a comment line, which contains the word `REXX` in upper or lower case.

Example

To call one REXX exec from another EXEC, use the following syntax:

```
xy=cxx('A B C D')
```

where `cxx` is the name of a REXX exec.

You can also use the following syntax, but the previous example is preferable.

```
"EX cxx 'A B C D'"
```

or

```
CALL cxx 'A B C D'
```

This example illustrates how individual arguments can be extracted from the argument string. The REXX exec called RHYME is passed several arguments while REXXARGS(TSO) is in effect.

```
EX RHYME 'MARY HAD A LITTLE LAMB'

/* REXX */
PARSE ARG argstring
argno = 'WORDS'(argstring) /* get # of arguments */
DO i = 1 TO argno /* display all the arguments */
   SAY 'argument #i 'is: 'WORD(argstring,i)
END
```

RHYME receives the following single argument: "MARY HAD A LITTLE LAMB", which it then has to break into individual arguments.
**EXECDROP**

**Description**
Causes specified REXX execs, or all REXX execs, to be removed from storage. Any exec which is in use is marked for removal when it ceases to be used. Any exec which is not in use is removed immediately. (Can be issued from the console.)

**Syntax**

```
EXECDROP execname LIST >>
```

or

```
EXECDROP *
```

**subsys**
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

**execname**
The name of a REXX exec.
You can use the pattern characters question mark (?) or asterisk (*) to specify a list of names. The pound sign (#) cannot be used as a pattern character, but can be used as a symbol within a name. The question mark is used to select any character. The asterisk is used to select any string of characters. If you imbed question marks or an asterisk within a text string, the character that follows the pattern character or characters must be a period. This example specifies a text string with two selection characters: **ABC??..XYZ**

* All REXX execs. To remove all execs from storage, specify an asterisk (*) instead of an exec name.

**LIST**
Indicates that a message is to be issued for each EXEC that is successfully dropped. The default is that a message is not issued as each EXEC is dropped.

The EXECDROP command is the only way to remove an exec from storage which was marked with EXECLOAD. EXECDROP removes the mark that indicates EXECLOAD was used, and decrements the use count.

**Note:** *EXECDROP does not decrement the use count for an exec which is not marked by EXECLOAD.*
### Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal completion</td>
</tr>
<tr>
<td>4</td>
<td>Specified member not known</td>
</tr>
<tr>
<td>8</td>
<td>One or more EXECs not dropped due to service error</td>
</tr>
<tr>
<td>12</td>
<td>Missing or invalid member name</td>
</tr>
<tr>
<td>16</td>
<td>Internal logic error</td>
</tr>
</tbody>
</table>

### Example

The command: **EXECDROP MEM1**
removes the REXX exec called MEM1 from storage.

The command: **EXECDROP *  
removes all loaded REXX execs from storage.**
EXECLOAD

Description
Causes a specified REXX exec to be loaded into storage. If it is already loaded in storage, it is just marked. If it is not already loaded in storage, it is explicitly loaded and marked. (Can be issued from the console.)

Syntax

```
   EXECLOAD execname
   subsys
```

subsys
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

execname
The name of a REXX exec

Comments
EXECLOAD increments the use count for a REXX exec. It prevents the exec from being removed from storage even when the exec is not in use. IF EXECLOAD is issued for an exec which has already been marked by EXECLOAD, the command is ignored and the use count is not further incremented.

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal completion</td>
</tr>
<tr>
<td>12</td>
<td>Missing or invalid member name</td>
</tr>
<tr>
<td>16</td>
<td>Internal logic error</td>
</tr>
</tbody>
</table>

Example
The command: EXECLOAD MEM1
loads a REXX exec called MEM1 into storage.
EXECMAP

Description
Causes the status of either a specified REXX exec or all REXX execs known by AF/OPERATOR to be listed. (Can be issued from the console.)

Syntax
EXECMAP execname

subsys
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

execname
The name of a REXX exec. You can use the pattern characters question mark (?) or asterisk (*) to specify a list of names. The pound sign (#) cannot be used as a pattern character, but can be used as a symbol within a name. The question mark is used to select any character. The asterisk is used to select any string of characters. If you imbed question marks or an asterisk within a text string, the character that follows the pattern character or characters must be a period. This example specifies a text string with two selection characters: ABC??..XYZ

* All REXX execs. An asterisk (*) is used to request the status of all known execs.

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal completion</td>
</tr>
<tr>
<td>4</td>
<td>Specified member not known</td>
</tr>
<tr>
<td>12</td>
<td>Missing or invalid member name</td>
</tr>
<tr>
<td>16</td>
<td>Internal logic error</td>
</tr>
</tbody>
</table>

Example
The command: EXECMAP MEM1
displays the status of the loaded REXX exec called MEM1.

The command: EXECMAP *
displays the status of all loaded REXX execs.
FREE (FR)

Description
FREE closes and reopens a SYSOUT dataset or frees a ddname or dataset that was previously allocated with the ALLOC command.

Syntax

```
FREE AFDDNAME(afddname)
FR DDNAME(ddname)
DDNAME(ddname)
DSN(dsname)
DSN(dsname)
UNALC
```

**AFDDNAME(ddname)**
Closes and reopens Automation Facility reserved SYSOUT datasets. Currently, the only applicable ddname is RKOGLOGM, the message log.

**DDNAME(ddname)**
The ddname of the file to be freed.

**DSN(dsname)**
The dsname of the dataset to be freed.

**UNALC** FREES a permanently allocated dataset.

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>File is freed successfully.</td>
</tr>
<tr>
<td>12</td>
<td>File is not allocated.</td>
</tr>
<tr>
<td>16</td>
<td>Invalid syntax.</td>
</tr>
</tbody>
</table>

Comments
- FR is a command synonym for FREE.
- The dsname must always be fully qualified; that is, patterns or default prefixes are not supported. However, you can use symbolic variables.
- AF/OPERATOR allocates SYSOUT datasets with the FREE=CLOSE parameter to dynamic allocation. The dataset is freed as soon as the CLOSE command is issued for the ddname, so you need not issue a FREE command for SYSOUT datasets. Such a command returns an error message indicating that the dataset is no longer allocated.
- Specify the UNALC keyword to FREE permanently allocated datasets such as those datasets allocated via JCL.
Example
The following is an example of this command embedded in a REXX exec:

```
"FREE DDNAME(OUTFILE)"
```
frees the file with the ddname OUTFILE.

```
"FREE DSN('SYS1.PROCLIB')"
```
frees the dataset SYS1.PROCLIB regardless of what ddname was used to allocate it.
HELP (HE)

Description
HELP displays language information about AF/OPERATOR commands. (Can be issued from the console.)

Syntax

```
HELP <subsys><command><FUNCTION><OPERAND(keyword,s)><SYNTAX><ALL>
```

subsys This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

command An AF/OPERATOR command such as AF, CP, EXEC, HELP, SHOW, and TRAP.

COMMANDS Displays a list of valid commands.
FUNCTION Displays information about the function of a command.

OPERAND(keyword,s) If you specify any keywords in the subfield, only information about those keywords is displayed. If you omit the subfield, information about all operands is displayed.

SYNTAX Displays information about the syntax of a command.

ALL Displays all help information.

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Help information is displayed.</td>
</tr>
<tr>
<td>4</td>
<td>Invalid action code.</td>
</tr>
<tr>
<td>12</td>
<td>Help is not available.</td>
</tr>
<tr>
<td>16</td>
<td>Invalid syntax.</td>
</tr>
</tbody>
</table>
Comments
- HE is a command synonym for HELP.
- All operands are optional. Entering HELP without operands has the same effect as entering: HELP COMMANDS ALL.
- HELP members must reside in one or more partitioned datasets that have been allocated to a DDNAME of RKANHENU.

Example
The following is an example of this command issued from the console.

HELP EXEC
HELP HELP
HELP TRAP OPERANDS(WTO)
IMSOPER

Description
IMSOPER issues a command from AF/OPERATOR to an IMS region, using the IMS WTOR reply ID. (Can be issued from the console.)

Syntax

```
subsys
IMSOPER  CTLRGN(regionname)  CMD('command')
```

subsys This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

CTLRGN(regionname)
The IMS region to which the command is directed.

CMD("/command")
An IMS command. Prefix the command text with a slash (/). This string can be from 1 to 126 characters.

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command issued successfully to an IMS region.</td>
</tr>
<tr>
<td>6</td>
<td>No outstanding reply ID found.</td>
</tr>
<tr>
<td>16</td>
<td>IMS control region not active; argument error found in replyID parmlist; non-zero return code received from IMS region; or invalid syntax.</td>
</tr>
</tbody>
</table>

Comments
If the WTOR is not available at the time the command is issued, AF/OPERATOR waits up to 5 minutes before issuing an error message. AF/OPERATOR uses its own internal enqueue to serialize its access to the WTOR in the event multiple IMSOPER commands are triggered at the same time.

Example
The following is an example of this command embedded in a REXX exec.

```
"IMSOPER CTLRGN(IMSTEST) CMD('/CHE SNAPQ')"
```
JES3OPER (J3OPER)

Description
JES3OPER issues a command to JES3. (Can be issued from the console.)

Syntax

```
subsys JES3OPER 'command'
```

```
subsys This operand is only required when issuing the command from the console. This is
the name of the AF/OPERATOR address space where you want the command to
execute. The subsys name is assigned by the SUBSYS startup parameter.

"command" A JES3 command.
```

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command issued successfully to JES3.</td>
</tr>
<tr>
<td>16</td>
<td>Invalid syntax.</td>
</tr>
</tbody>
</table>

Comments
If you are using JES3 Version 2.2.1 or above, you do not need to use JES3OPER. Use OPER to send commands to JES3.

If you are using a version of JES3 before 2.2, you must install the JES3 interface and use JES3OPER to send commands to JES3. See the AF/OPERATOR Configuration and Customization Guide for information on the JES3 interface.

Example
The following is an example of this command embedded in a REXX exec.

```
"JES3OPER ""I R"""
```
**Description**

LINK manages AF/OPERATOR communications client connections to remote servers.

This section gives a general description of the LINK command. Its subtypes are described separately in subsequent sections.

**Syntax**

```
LINK
  DEFINE parameters
  START parameters
  STOP parameters
  DELETE parameters
  DISPLAY parameters
```

**DEFINE**  Defines a remote communication link or client session.

**START**  Initiates a remote communication link or client session.

**STOP**  Terminates a remote communication link or client session.

**DELETE**  Deletes a remote communications link or client definition. A LINK STOP command is implied before the definition is deleted.

**DISPLAY**  Displays communications link or client definition information and session information.
LINK DEFINE

Description
Defines a remote communication link.

Syntax

```
>> LINK DEFINE LINKID(linkid) >>
      TYPE(protocol) >>
  SYMDEST(symdest) >>
      TPNAME(tpname)    PLUNAME(pluname) >>
                        MODENAME(modename)
  GROUP(xcf-group-name)    MEMBER(xcf-member-name)
  HOSTNAME(remote-hostname) PORTNUMBER(remote-port-number)

  NOHRTBT >>
  NOID >>
  SENDRCV >>
  RECOVERY >>

  ASCII >>
  APPACKET(ON/OFF) >>
  ACTIVATE <<
```

Parameters

**TYPE(protocol)**

Specifies the type of communication protocol to use for this session.

**LINKID(linkid)**

Identifies the target application for this communication session and uniquely identifies the session. The maximum length of linkid is 8 characters.

*Note: The linkid you specify must not start with the # symbol. AF/OPERATOR reserves linkids beginning with a # in the range #0000001 to #9999999 for internal use only.*

**SYMDEST(symdest)**

Identifies the application you want to establish a session with by its APPC symbolic destination name. The maximum length of symdest is 8 characters.

**TPNAME(tpname)**

Identifies the application you want to establish a session with by its APPC transaction program name. The maximum length of tpname is 64 characters.

**PLUNAME(pluname)**

Identifies the application you want to establish a session with by its APPC partner LU (logical unit) name. The maximum length of pluname is 17 characters.
**MODENAME**(*modename*)

Optional. Identifies the VTAM log mode table entry definition used for VTAM LU6.2 sessions at your site. It sets default session parameters for this session. The maximum length of *modename* is 8 characters. If MODENAME is not specified, it defaults to the mode in effect for the partner LU. Refer to *MVS/ESA V4 Planning: APPC Management* topic “logon mode, specifying for a conversation”.

**NOHRTBT**

Optional.

Turns off AF/OPERATOR to AF/OPERATOR heartbeat checking. Specify NOHRTBT if you are defining a session to a distributed application, such as REMO/OS2.

**NOID**

Optional. Turns off AF/OPERATOR internal ID request processing. Specify NOID if you are defining a session to a distributed application, such as REMO/OS2. This prevents an internal AF/OPERATOR record from flowing as part of the data stream of outbound data.

**SENDRCV**

Optional. Send and receive data in a two-way alternate session on the same link (a half-duplex session). Distributed applications, such as REMO/OS2, do not support SENDRCV. This option is provided for internal use by AF/OPERATOR applications.

**GROUP**(*xcf-group-name*)

Identifies the XCF group you want to establish a session with. It must be used in conjunction with MEMBER. The maximum length of *xcf-group-name* is 8 characters.

**MEMBER**(*xcf-member-name*)

Specifies the XCF member name. Identifies which particular server to display. It must be used in conjunction with GROUP. The maximum length of *xcf-member-name* is 8 characters.

**HOSTNAME**(*remote-hostname*)

Identifies the IP address of the remote host you wish to establish a session with. HOSTNAME must be used in conjunction with PORTNUMBER. The maximum length of *remote-hostname* is 24 characters. It can be either a dotted decimal IP address or a hostname defined to your network’s domain name server. If *remote-hostname* is a dotted decimal IP address, none of the four numeric levels of the address can exceed 255.

**PORTNUMBER**(*remote-port-number*)

Identifies the listening port of the remote host you wish to establish a session with. PORTNUMBER must be used in conjunction with HOSTNAME. The port number can be any integer greater than or equal to 1025 and less than or equal to 32767.

**RECOVERY**

Optional. When specified, indicates that an INACTIVE link with a ‘desired’ state of ACTIVE is to be automatically started when the specified server becomes available.
ASCII
When a TCP/IP connection is established between AF/OPERATOR and any other partner application, a negotiation occurs regarding whether EBCDIC or ASCII text is to be transferred. AF/OPERATOR will initiate the negotiation by sending an ID record. By default, this record will be EBCDIC. Specifying ASCII on the LINK DEFINE causes this initial contact to be made using ASCII.

Using this option can mean that the distributed system never has to handle EBCDIC data, thereby simplifying the process on that system.

**Note:** This keyword affects the first ID record sent from AF/OPERATOR when the LINK START is performed. It does not necessarily determine the outcome of the negotiation. The distributed system can still negotiate that EBCDIC be used.

AFPACKET(ON/OFF)
Enable or disable the AF packet header over this TCP/IP LINK. The default is ON.

**Notes:**
- For additional information, see “Open TCP/IP Communications” in the AF/OPERATOR User’s Guide section called “Advanced TCP/IP Communications.”
- AF/PACKET(OFF) can only be specified for a TCP/IP connection.
- Data transfer over a LINK with the AF Packet header turned off can only be performed using the COMSDRCV REXX function SEND and RECEIVE requests. Conversely, SEND and RECEIVE are not permitted on a LINK that uses the AF Packet header.

ACTIVATE
Assuming successful link definition, indicates a LINK START will be automatically attempted for the particular linkid.

**Note:** When used within the LINKDEFS member of RKANPAR, the automatic LINK START is deferred until all records in LINKDEFS are processed.

Comments
- Each LINK DEFINE session must have a unique LINKID.
- When TYPE is not specified, the default will be determined by which other protocol specific keywords are present.
- If you specify a SYMDEST parameter, there is no need to specify a TPNAME, PLUNAME, or MODE parameter.
- If you specify TPNAME, PLUNAME, or MODE parameters in addition to a SYMDEST parameter, the TPNAME, PLUNAME, or MODE specification overrides the value obtained from the side information dataset.
- When defining a session to an AF/OPERATOR system, the linkid you specify can be the value assigned to the AF/OPERATOR LINKID startup parameter on the other system.
- Heartbeat checking between AF/OPERATOR systems (automatic internal checking to verify the link is active) is performed by default for APPC and XCF. AF/OPERATOR sessions are started as confirm sessions. When defining an APPC connection to a distributed application, such as AF/REMOTE for OS2 specify NOID and NOHRTBT.
The keywords for a given communications protocol are mutually exclusive with the keywords of any other communications protocol.

**Internal Sessions and Servers**

For information about communications sessions, refer to “Session Types” in the chapter “Understanding the Communications Environment” in the *AF/OPERATOR User’s Guide.*

**The XCF RECOVERY option**

When the RECOVERY keyword is specified on an XCF LINK DEFINE, this indicates that an INACTIVE link that has a ‘desired’ state of ACTIVE is to be automatically started when the specified server becomes available.

A link’s ‘desired’ state is set to ACTIVE when the last state changing command issued against the link was an attempt to start the link (LINK START), otherwise it’s ‘desired’ state is INACTIVE.

An INACTIVE link that has a ‘desired’ state of ACTIVE can occur after attempting a LINK START to a server that is not currently active, or if the server is stopped following a successful LINK START.

**Example**

The following exec defines an APPC communication session whose LINKID is RMT:

```rexx
/* REXX */
ADDRESS AFHOST
"LINK DEFINE LINKID(RMT) TYPE(APPC) TPNAME(PEERTEST) MODE(LU62) PLU(LOA4769I)"
IF rc <> 0 THEN SAY "LINK DEFINE FAILED"
RETURN rc
```

The following defines an XCF communication link:

```rexx
/* REXX */
"LINK DEFINE LINKID(RMT) TYPE(XCF) GROUP(KOGXCF01) MEMBER(SYSASRVR)"
```

In the above example, we attempt to define a link to member SYSASRVR of the XCF group called KOGXCF01.

The following is an example of how to define a TCP/IP communications link. It targets a specific remote IP address and listening port

```rexx
/* REXX */
"LINK DEFINE LINKID(RMT) TYPE(TCPIP) HOSTNAME(198.210.44.10) PORTNUMBER(1234)"
```
The following is an example of the use of the ASCII parameter:

/* REXX */
“LINK DEFINE LINK(XYZ) HOST(SYSA) PORT(2134) ASCII”

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Link successfully defined.</td>
</tr>
<tr>
<td>1001</td>
<td>Link define syntax error.</td>
</tr>
<tr>
<td>1002</td>
<td>If TYPE is APPC, either the IBM APPC load library SYS1.CSSLIB is not specified or the minimum requirement of MVS/ESA Version 4.3 or higher has not been met. If TYPE is TCPIP, the minimum requirement of TCP/IP Version 3 Release 2 for MVS has not been met.</td>
</tr>
<tr>
<td>1003</td>
<td>Unknown protocol.</td>
</tr>
<tr>
<td>1004</td>
<td>Duplicate entry. The linkid is already defined.</td>
</tr>
<tr>
<td>2001</td>
<td>Parameters missing on link definition.</td>
</tr>
<tr>
<td>2002</td>
<td>Send save area getmain error.</td>
</tr>
<tr>
<td>2016</td>
<td>Link define processing abended.</td>
</tr>
</tbody>
</table>
LINK DELETE

Terminates and deletes a specified communication link, all links, or all links of a specified protocol type, and their definitions.

Syntax

```
> LINK DELETE
    TYPE(protocol)   LINKID(linkid)
<
```

Parameters

**TYPE(protocol)**

Specifies the type of communication session to delete.

**LINKID(linkid)**

Uniquely identifies a predefined session to delete. The maximum length of linkid is 8 characters.

*Note:* The linkid you specify must not start with the # symbol. **AF/OPERATOR** reserves linkids beginning with the # symbol in the range #0000001 to #9999999 for internal use only.

**DRAIN**

Allow queued requests for this link to complete before terminating and deleting the link.

**FORCE**

Allow an active LINKID to be deleted regardless of the link’s status.

*Note:* When a link is deleted with the FORCE option, the active match holding the enqueue is also cancelled. When this occurs, you may receive 13E abends in the SYSLOG indicating that the match had active MVS subtasks that were detached.

*Caution:* A forced LINK DELETE bypasses an orderly shutdown of the link. Frequent use of this option may result in virtual memory shortage or other unpredictable behavior that could force you to recycle AF/OPERATOR.
Comments

- You can terminate and delete the link entry block definitions of all sessions by issuing a LINK DELETE command without specifying a LINKID or TYPE parameter.
- Specifying only the TYPE parameter terminates and deletes the link entry block definitions of all sessions of the specified protocol.
- If you specify a LINKID parameter, the TYPE specification from the session’s LINK DEFINE is assumed.
- LINK DELETE command deletes the link entry block. Since linkid names must be unique, you must first issue a LINK DELETE command for a particular linkid name before reusing the name in another LINK DEFINE command.
- The LINK DELETE command includes a LINK STOP command. You do not need to issue a LINK STOP before issuing a LINK DELETE command.
- If you specify DRAIN, all queued data will be transferred before the link is terminated and deleted. If DRAIN is not specified, the specified link and its definition, or all links and their definitions, are immediately terminated and deleted.

Example

The following example deletes the connection to the session whose LINKID is RMT:

```rexx
/* REXX */
ADDRESS AFHOST
"LINK DELETE LINKID(RMT)"
IF rc <> 0 THEN SAY "LINK DELETE FAILED"
RETURN rc
```

The following example deletes a specific XCF communication session whose LINKID is LNKXCF1:

```rexx
/* REXX */
"LINK DELETE LINKID(LNKXCF1) TYPE(XCF)"
```

The following example deletes a specific TCP/IP communication session whose LINKID is LNKTCP1:

```rexx
/* REXX */
"LINK DELETE LINKID(LNKTCP1) TYPE(TCPIP)"
```
## Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Link deleted successfully.</td>
</tr>
<tr>
<td>1001</td>
<td>Syntax error.</td>
</tr>
<tr>
<td>1005</td>
<td>No linkid defined.</td>
</tr>
<tr>
<td>1201</td>
<td>Link not active.</td>
</tr>
<tr>
<td>1202</td>
<td>Invalid protocol TYPE.</td>
</tr>
<tr>
<td>1216</td>
<td>Link delete processing abended.</td>
</tr>
</tbody>
</table>
LINK DISPLAY

Description
Displays the current link status for a specified link, for all links, or for all links of a specified protocol type.

Syntax

```
>> LINK DISPLAY
    LINKID(linkid)  TYPE(protocol)  DETAIL(FULL/BRIEF) <<
```

Parameters

**TYPE(protocol)**
Specifies the type of communication session to display.

**LINKID(linkid)**
Identifies the target application for this communication session and uniquely identifies the session. The maximum length of *linkid* is 8 characters.

*Note:* the linkid you specify must not start with the # symbol. AF/OPERATOR reserves linkids beginning with the # symbol in the range #0000001 to #9999999 for internal use only.

**DETAIL(FULL/BRIEF)**
Identifies the level of information to be displayed and permits you to reduce the number of output lines produced.

  - **BRIEF**  Displays minimal connection information.
  - **FULL**  Displays detailed connection information, including options from LINK DEFINE and send counts, etc.

Comments

- You can display information for all sessions by issuing a LINK DISPLAY command without specifying a LINKID or TYPE parameter.
- Specifying only the TYPE parameter displays information for all sessions of the specified protocol.
- If you specify a LINKID parameter, the TYPE specification from the session’s LINK DEFINE command is assumed.


**Example**

The LINK DISPLAY command in the following exec displays the status of the APPC session whose LINKID is RMT.

```rexx
/* REXX */
ADDRESS AFHOST
"LINK DEFINE LINKID(RMT) TYPE(APPC) TPNAME(PEERTEST) MODENAME(LU62)
PLUNAME(LOA4769I)"
IF rc <> 0 THEN SAY "LINK DEFINE FAILED"
ELSE DO
   "LINK START LINKID(RMT)"
   IF rc <> 0 THEN SAY "LINK START FAILED"
   ELSE
      "LINK DISPLAY LINKID(RMT)"
   END
END
RETURN rc
```

The following is an example of how to display all XCF communication sessions.

```rexx
/* REXX */
"LINK DISPLAY TYPE(XCF)"
```

A typical response to the LINK DISPLAY TYPE(APPC) command follows:

```
LINK DISPLAY TYPE(APPC)
!AOP3776 SEND ONLY SESSION SYSG1 ACTIVE
!AOP3776 ---- SYMDEST - ********
!AOP3776 ---- TPNAME - PEERTEST
!AOP3776 ---- MODNAME - LU62
!AOP3776 ---- PLUNAME - HIGHWAY1
!AOP3776 ---- TYPE - APPC
!AOP3776 ---- SEND COUNT (SEND ONLY) - 1
!AOP3776 ---- QUEUED COUNT (SEND ONLY) - 0
!AOP3776 ---- SENDRCV SESSION INACTIVE
!AOP3776 ---- NO HEARTBEAT CHECKING
!AOP3776 **********************************************
```

A typical response to the LINK DISPLAY TYPE(XCF) command follows:

```
LINK DISPLAY TYPE(XCF)
!AOP3776 SEND ONLY SESSION LNKXCF1 ACTIVE
!AOP3776 ---- GROUP - KOGXCF1
!AOP3776 ---- MEMBER - SYSASRVR
!AOP3776 ---- TYPE - XCF
!AOP3776 ---- SEND COUNT (SEND ONLY) - 10686
!AOP3776 ---- QUEUED COUNT (SEND ONLY) - 0
!AOP3776 ---- SENDRCV SESSION INACTIVE
!AOP3776 ---- RECOVERY ON - DESIRED ACTIVE
```
A typical response to the LINK DISPLAY TYPE(TCPIP) command follows:

```
LINK DISPLAY TYPE(TCPIP)
!AOP3776 DUPLEX SESSION LNKTCP1 ACTIVE
!AOP3776 ---- HOSTNAME   - 111.112.113.114
!AOP3776 ---- PORTNUMBER - 1234
!AOP3776 ---- TYPE       - TCPIP
!AOP3776 ---- SEND COUNT (SEND ONLY) - 10
!AOP3776 ---- QUEUED COUNT (SEND ONLY) - 0
!AOP3776 ---- RECEIVE COUNT (DUPLEX) - 2
!AOP3776 ---- NO HEARTBEAT CHECKING
!AOP3776 *****************************************
```

The display output is routed to the SYSLOG and to the AF/OPERATOR message log.

A typical response to the LINK DISPLAY DETAIL(BRIEF) command follows:

```
LINK DISPLAY DETAIL(BRIEF)
!AOP3776 DUPLEX SESSION LINK1 ACTIVE
!AOP3776 SEND ONLY SESSION LINK2 INACTIVE
!AOP3776 SEND ONLY SESSION LINK3 ACTIVE
!AOP3776 DUPLEX SESSION LINK4 INACTIVE
```

**Return Codes**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Link display successful.</td>
</tr>
<tr>
<td>1316</td>
<td>Link display processing abended.</td>
</tr>
</tbody>
</table>
# LINK START

**Description**

Initiates a particular communication session, all communication sessions, or all sessions of a specified type.

**Syntax**

```
> LINK START TYPE(protocol) LINKID(linkid) <
```

**Parameters**

**TYPE(protocol)**

Specifies the type of communication session to initiate.

**LINKID(linkid)**

Uniquely identifies the target application for this communication session and uniquely identifies the session. The maximum length of `linkid` is 8 characters.

**SCOPE(ALL/INACTIVE/ACTIVATE)**

Links are selected for LINK START processing according to their activation state.

- **ALL** -- Links in any activation state are selected. This is the default.
- **INACTIVE** -- Only inactive links are selected.
- **ACTIVATE** -- Links that are “desired active” are selected. See the RECOVERY option on “LINK DEFINE” on page 275 for additional information.

**Note:** SCOPE(ACTIVATE) is also used to honor the ACTIVATE option used on a LINK DEFINE contained within the LINKDEFS member of RKANPAR, where the immediate activation is deferred until all records in LINKDEFS are processed.

**Comments**

- You can start all defined sessions that are not already started by issuing a LINK START command with no further specifications.
- Specifying only the TYPE parameter starts all defined sessions of the specified protocol that are not already started.
- If you specify a LINKID parameter, the TYPE specification from the session’s LINK DEFINE command is assumed.
- Before a successful LINK START command can be issued from one AF/OPERATOR to another:
If the communications protocol is APPC, the AF/OPERATOR communications component must be active on both AF/OPERATOR systems which are to be linked. (You can check the status of the communication component by issuing the AF/OPERATOR COMM DISPLAY command.)

- The communication session from this AF/OPERATOR system to the partner AF/OPERATOR system be defined using the LINK DEFINE command.

Example
The following is a sample REXX exec that first defines, then starts, an APPC communication session whose LINKID is RMT:

```rexx
/* REXX */
ADDRESS AFHOST
"LINK DEFINE LINKID(RMT) TYPE(APPC) TPNAME(PEERTEST) MODE(LU62)
PLU(LOA4769I)"
IF rc <> 0 THEN SAY "LINK DEFINE FAILED"
ELSE DO
  "LINK START LINKID(RMT)"
  IF rc <> 0 THEN SAY "LINK START FAILED"
  ELSE
    "LINK DISPLAY LINKID(RMT)"
  END
END
RETURN rc
```

The following sample REXX exec defines and starts an XCF communication session whose LINKID is LNKXCF1:

```rexx
/* REXX */
ADDRESS AFHOST
"LINK DEFINE LINKID(LNKXCF1) TYPE(XCF) GROUP(KOGXCF01) MEMBER(SYSASRVR)"
IF rc <> 0 THEN SAY "LINK DEFINE FAILED"
ELSE DO
  "LINK START LINKID(LNKXCF1)"
  ELSE
    "LINK DISPLAY LINKID(LNKXCF1)"
  END
END
RETURN rc
```

The following sample REXX exec defines and starts a TCP/IP communication session whose LINKID is LNKTCP1:

```rexx
/* REXX */
ADDRESS AFHOST
"LINK DEFINE LINKID(LNKTCP1) TYPE(TCP)"
IF rc <> 0 THEN SAY "LINK DEFINE FAILED"
ELSE DO
  "LINK START LINKID(LNKTCP1)"
  ELSE
    "LINK DISPLAY LINKID(LNKTCP1)"
  END
END
RETURN rc
```
LINK START

/* REXX */
ADDRESS AFHOST
"LINK DEFINE LINKID(LNKTCP1) TYPE(TCPIP) HOSTNAME(SYSA) PORTNUMBER(1234)"
IF rc <> 0 THEN SAY "LINK DEFINE FAILED"
ELSE DO
  "LINK START LINKID(LNKTCP1)"
  IF rc <> 0 THEN SAY "LINK START FAILED"
  ELSE
    "LINK DISPLAY LINKID(LNKTCP1)"
  END
RETURN rc

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Link started successfully.</td>
</tr>
<tr>
<td>1001</td>
<td>Syntax error.</td>
</tr>
<tr>
<td>1003</td>
<td>Unknown protocol.</td>
</tr>
<tr>
<td>1005</td>
<td>No linkid is defined.</td>
</tr>
<tr>
<td>1101</td>
<td>Internal error.</td>
</tr>
<tr>
<td>1102</td>
<td>Link is already active.</td>
</tr>
<tr>
<td>1103</td>
<td>Not a valid type to start.</td>
</tr>
<tr>
<td>1104</td>
<td>Unknown protocol.</td>
</tr>
<tr>
<td>1105</td>
<td>Send manager attach error.</td>
</tr>
<tr>
<td>1106</td>
<td>Internal error.</td>
</tr>
<tr>
<td></td>
<td>For AF/OPERATOR to AF/OPERATOR sessions make sure that the AF/OPERATOR address space and AF/OPERATOR server are active.</td>
</tr>
<tr>
<td></td>
<td>For AF/OPERATOR to AF/REMOTE sessions, make sure that the APPC or TCP/IP interface is active.</td>
</tr>
<tr>
<td>1107</td>
<td>No server active; no COMM START has been issued.</td>
</tr>
<tr>
<td>1108</td>
<td>Internal link entry can not be started.</td>
</tr>
<tr>
<td>1109</td>
<td>Link not started—link termination in progress.</td>
</tr>
<tr>
<td>1116</td>
<td>Link start processing abended.</td>
</tr>
<tr>
<td>2201</td>
<td>Session allocation error.</td>
</tr>
<tr>
<td>2202</td>
<td>If TYPE is TCPIP, the minimum requirement of TCP/IP Version 3 Release 2 for MVS has not been met.</td>
</tr>
<tr>
<td>2216</td>
<td>LINK START abended.</td>
</tr>
</tbody>
</table>
LINK STOP

Description
Terminates a specified communication link, terminates all links, or terminates all links of a specified protocol type.

Syntax

```plaintext
>> LINK STOP  
   TYPE(protocol)  
   LINKID(linkid)  
   DRAIN  
<<
```

Parameters

**TYPE(protocol)**
Specified the type of communication session to stop.

**LINKID(linkid)**
Uniquely identifies a predefined session to stop. The maximum length of the `linkid` is 8 characters.

*Note: The linkid you specify must not start with the # symbol. AF/OPERATOR reserves linkids beginning with the # symbol in the range #0000001 to #9999999 for internal use only.*

**DRAIN**
Allow queued requests for this link to complete before terminating the link.

Comments

- You can terminate all active sessions by issuing a LINK STOP command without specifying either a LINKID or TYPE parameter.
- Specifying only the TYPE parameter stops all active sessions of the specified protocol.
- If you provide a LINKID parameter, the TYPE specification from that session’s LINK DEFINE command is assumed.
- If you specify DRAIN, all queued data will be transferred before the link is terminated. If DRAIN is not specified, the specified link, or all links, immediately terminate.
Example
The following example stops a specific APPC communication session whose LINKID is RMT:

```rexx
/* REXX */
ADDRESS AFHOST
"LINK STOP LINKID(RMT)"
IF rc <> 0 THEN SAY "LINK STOP FAILED"
RETURN rc
```

The following example demonstrates the behavior of the DRAIN option:

```rexx
/* REXX */
DATA = 'SHOW MAT'
DO 100
   STATUS = COMSDRV('SYSG1,DATA,EXEC)
   SAY 'STATUS = ' STATUS
END
"LINK STOP LINKID(SYSG1) DRAIN"
RETURN
```

All 100 SHOW MAT requests will be sent to SYG1 before the link terminates.

The following example stops a specific XCF communication session whose LINKID is LNKXCF1:

```rexx
/* REXX */
"LINK STOP LINKID(LNKXCF1) TYPE(XCF)"
```

The following example stops all XCF communication sessions:

```rexx
/* REXX */
"LINK STOP TYPE(XCF)"
```

The following example stops all TCP/IP communication sessions:

```rexx
/* REXX */
"LINK STOP TYPE(TCPIP)"
```
Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Link stopped successfully.</td>
</tr>
<tr>
<td>1001</td>
<td>Syntax error.</td>
</tr>
<tr>
<td>1005</td>
<td>No linkid defined.</td>
</tr>
<tr>
<td>1201</td>
<td>Link not active.</td>
</tr>
<tr>
<td>1202</td>
<td>Invalid protocol TYPE.</td>
</tr>
<tr>
<td>1216</td>
<td>Link stop processing abended.</td>
</tr>
</tbody>
</table>
LOGOFF

Description
LOGOFF terminates a VTAM application session that was previously initiated with a LOGON command.

Syntax
LOGOFF applname

applname  The name assigned to the application session by the NAME operand in the LOGON command that initiated the session. This is the same name used in the AFADDR command.

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>VTAM session is terminated.</td>
</tr>
<tr>
<td>16</td>
<td>Environment error or invalid syntax.</td>
</tr>
</tbody>
</table>

Example
The following is an example of this command embedded in a REXX exec.

An OMEGAMON session was initiated with the following LOGON command:

   "LOGON OMMVS APPLID(OMVTAM) NAME(OMMVS1) USERID(SYS001/SECRET)"

You can terminate this session with the following command:

   "LOGOFF OMMVS1"
LOGON

Description
LOGON initiates a session with a VTAM application, specifies operational parameters for the link, and gives the session a name referenced by AFADDR commands.

Syntax

LOGON applname APPLID(applid) NAME(applname) <option(s)>

Options:
- DATA(text)
- INTERVAL(time)
- LTERM(luid)
- PASSWORD(password)
- RRESPTIME(number of seconds)
- SSIZE(n)
- TRACE
- USERID(userid/password)

applname The type of VTAM application. Supported options include:

- AFGATE Programmerless Open VTAM Interface
- OMCICS OMEGAMON for CICS
- OMDB2 OMEGAMON II for DB2
- OMIMS OMEGAMON for IMS
- OMMVS OMEGAMON for MVS
- OMVIEW OMEGAVIEW
**APPLID**<br> The APPLID of the VTAM application. This is the ID specified by the user data center. The default APPLID for the Programmerless Open VTAM Interface is OVIAO.

For OMEGAMON II CUA™ products, use the APPLID you used for the AF/OPERATOR logon, not the CUA APPLID. This is user modifiable during installation. For more information about this interface, see the AF/OPERATOR Configuration and Customization Guide and the AF/OPERATOR User’s Guide.

**NAME**<br> A unique user-defined name to be used in subsequent AF/OPERATOR commands and/or functions (for example, the AFADDR command) to refer to the VTAM session. AF/OPERATOR identifies the VTAM session by this name. AF/OPERATOR supplies the value match variable AOXANSID stores.

For OMEGAVIEW (when appltype is OMVIEW), this value is passed as the assigned managed system name (MSN) for the session.

**Note:** If the value is OGATEWAY, then the MSN value becomes OGMVS. This is provided for backward compatibility.

The following operands are used only when logging onto an OMEGAMON session:

**DATA**<br> Logon information required by the application. For example, when logging onto OMEGAMON for CICS, this operand is required in the form:

```
DATA('CICS=jobname')
```

where `jobname` specifies the job running the OMEGAMON session.

You can also use this parameter to pass other types of information to OMEGAMON, such as the name of a user profile.

**INTERVAL**<br> Specifies how frequently AF/OPERATOR polls OMEGAMON for exceptions. The polling interval is written in the form `hh:mm:ss`.

This operand can be used only when logging onto OMEGAMON sessions; that is, if the appltype is OMCICS, OMDB2, OMIMS, or OMMVS.

If an interval is not specified, xtype traps will not function.

**LTERM**<br> A parameter that can be used to force the same virtual terminal ID to be used each time.

```
LOGON OMMVS APPLID(OMVTAM) NAME(OMMVS1) +
USERID(SYS001/password) LTERM(AOVTM101)
```

The luid does not have to be in any of the $VTAPPL statements of the AOVTPool (so it won’t be chosen by any other LOGON), but it needs to be defined to VTAM just as though it were in the pool.

**PASSWORD**<br> A password required by OMEGAMON’s internal security for issuing authorized commands.
SSIZE(2/3/4/5)

A single-digit numeric argument used to select the virtual screen size.

Note: When you are logging onto OMEGAVIEW, you cannot specify a value other than the default (2).

Optional screen requests are:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Terminal Type</th>
<th>Screen Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSIZE(2)</td>
<td>3270 model 2</td>
<td>24 by 80 (default)</td>
</tr>
<tr>
<td>SSIZE(3)</td>
<td>3270 model 3</td>
<td>32 by 80</td>
</tr>
<tr>
<td>SSIZE(4)</td>
<td>3270 model 4</td>
<td>43 by 80</td>
</tr>
<tr>
<td>SSIZE(5)</td>
<td>3270 model 5</td>
<td>27 by 132</td>
</tr>
</tbody>
</table>

TRACE

Causes the active environment to be traced and reports specific error information.

USERID(userid/password)

A userID and password required by OMEGAMON’s external security for logging on. The password appears as asterisks in the system log.

The following operand is used only when logging onto an OMEGAVIEW session:

RESPTIME(number of seconds)

Specifies in seconds the maximum time that a session is expected to take to respond to a logon request. If this time is exceeded, the session will be logged off and the appropriate return code set. If 0 is specified, then requests will wait for a response and never time out. The maximum allowed is 9999 seconds. The default RESPTIME is specified by means of the LOGON_RESPTIME startup parameter.

Note: Although this option may be specified for all LOGON commands, it affects only OMEGAVIEW sessions.

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>VTAM session is initiated.</td>
</tr>
<tr>
<td>16</td>
<td>Link is in use, or invalid syntax.</td>
</tr>
</tbody>
</table>
Comments

- The LOGON command is often issued by a startup REXX exec.
- You can log onto several applications at one time. Applications with different applnames can share the same appltype; for example, one OMEGAMON for MVS session and two OMEGAMON for CICS sessions.
- AF/OPERATOR sets the match variable AOXANSID to the value of the NAME parameter of the LOGON command. You can use the AOXANSID variable in xtype traps to determine which application generated an exception, when you are logged on to more than one application of the same type.
- The maximum logon time default is 20 seconds, unless overridden by the LOGONTLM startup option. The range available is 1-99 seconds. See the AF/OPERATOR Configuration and Customization Guide for more information.
- The overhead associated with exception processing increases as the value of INTERVAL decreases.
- OMEGAMON for MVS version 700 or higher, OMEGAMON for CICS Version 451, OMEGAMON for DB2, and OMEGAMON for IMS include a data compression option. If you are currently running either of these versions with the data compression option activated (DC=Y), it is not necessary to restart OMEGAMON. You can have AF/OPERATOR logon with the data compression turned off by specifying DATA(‘DC=N’) in the LOGON statement.

Example

The following are examples of this command embedded in a REXX exec.

- **Examples of OMEGAMON Logon**

  The following LOGON command logs onto a session of OMEGAMON for MVS through the VTAM APPL OMVTAM. AF/OPERATOR will address the session as OMMVS1 and will poll for OMEGAMON exceptions every minute. AF/OPERATOR supplies OMEGAMON’s external security system with the user ID ABCD99 and the user password AFPASS. In order to issue OMEGAMON authorized commands, AF/OPERATOR supplies the internal password AFAUTH.

  ```
  "LOGON OMMVS APPLID(OMVTAM) NAME(OMMVS1) INTERVAL(00:01:00)
  USERID(ABCD99/AFPASS) PASSWORD(AFAUTH)"
  ```

  The following LOGON command logs onto a session of OMEGAMON for CICS through the VTAM link OCVTAM. AF/OPERATOR will address the session as OCPROD and will poll for OMEGAMON exceptions every two minutes. The DATA operand requests logon to the CICS session run by the job OCPROD.

  ```
  "LOGON OMCICS APPLID(OCVTAM) NAME(OCPROD) INTERVAL(00:02:00)
  DATA(CICS=OCPROD) USERID(ABCD99/AFPASS) PASSWORD(AFAUTH)"
  ```

  The following LOGON command logs onto a session of OMEGAMON for CICS through the same VTAM link, OCVTAM. AF/OPERATOR will address the session as OCTEST and will poll for OMEGAMON exceptions every two minutes. The DATA operand requests logon to the CICS session run by the job OCTEST.
“LOGON OMICS APPLID(OIVTAM) NAME(OCTEST) INTERVAL(00:02:00),
   "DATA(CICS=OCTEST) USERID(ABCD99/AFPASS) PASSWORD(AFAUTH)"

The following LOGON command logs onto a session of OMEGAMON for IMS through the VTAM link OIVTAM. AF/OPERATOR will address the session as OIPROD and will poll for OMEGAMON exceptions every two minutes.

“LOGON OMIMS APPLID(OIVTAM) NAME(OIPROD) INTERVAL(00:02:00),
   "USERID(ABCD99/AFPASS) PASSWORD(AFAUTH)"

The following LOGON command logs onto a session of OMEGAMON for IMS through the VTAM link OIVTAMD1. AF/OPERATOR will address the session as OIDEV1 and will poll for OMEGAMON exceptions every two minutes. (IMS differs from CICS in that different OMEGAMON for IMS sessions must use different VTAM applids.)

“LOGON OMIMS APPLID(OIVTAMD1) NAME(OIDEV1) INTERVAL(00:02:00),
   "USERID(ABCD99/AFPASS) PASSWORD(AFAUTH)"

- **Example of Programmerless Open VTAM Interface Logon**

  The following example logs onto a session of the Programmerless Open VTAM Interface, called POVI01:

  “LOGON AFGATE APPLID(OVIAO) NAME(POVI01)"

- **Example of OMEGAVIEW Logon**

  The following example logs onto an OMEGAVIEW session specifying an MSN of SYSG:

  “LOGON OMVIEW APPLID(MV130) NAME(SYSG)"
**MLWTO**

**Description**
MLWTO issues a multiple-line write-to-operator message to one or more consoles.

**Syntax**

```
MLWTO variable_prefix

AFMLWTO variable_prefix CONSOLE(id)

name

ROUT(code,s,n:n,n-n)

DESC(code,s,n:n,n-n)

MSGID(id)

WTNO(varname)
```

**variable_prefix**
A 1- to 7-byte variable name prefix.

**CONSOLE(id)**
A 1- to 4-digit identifier of the console which is the message destination.

**CONSOLE(name)**
A 1- to 8-character console name which is the message destination. By default, the message is routed back to the console that issued the command.

**Note**: *If the specified console is inactive then the message will be sent to the master console.*

**ROUT(code,s,n:n,n-n)**
Route codes for the message. The value of the code is an integer from 1-128. If not specified, the default from the MSGDFLT startup option will be used. However, if CONSOLE has been specified then ROUT will be ignored.

**DESC(code,s,n:n,n-n)**
Descriptor codes for the message. The value of the code is an integer from 1-16. If not specified, the default from the MSGDFLT startup option will be used. However, if CONSOLE has been specified then DESC will be ignored.

**MSGID(id)**
A 1- to 7-character message prefix. The MLWTO command will use this value as the message prefix in messages it issues. If you don't specify MSGID, the MLWTO will be issued without a prefix.
**WTONO(varname)**

The name of a character variable in which the message number for this message will be stored. This variable name can be used as input to a subsequent DOM command in order to delete the MLWTO from the operator console. If a variable of this name does not already exist, the system creates it in the global pool. If a variable of this name exists, in either the global or local variable pools, then the system will reuse it. If both global and local variables of this name exist, then the system will reuse the local variable, leaving the global variable unchanged. If the variable name is omitted, the system will create a variable based on the variable_prefix with a # (pound sign) concatenated at the end. For example, if the variable_prefix was specified as MLW then a variable named MLW# is created to contain the message number.

**Comments**

Before calling MLWTO, the user should create a set of global variables based on the variable_prefix as follows:

- **variable_prefix**
  
  Label line text. Optional. If supplied it will be inserted as a label in the resulting multiple-line WTO.

- **variable_prefix0**
  
  Count of the number of data lines.

- **variable_prefix1**
  
  The first line of data.

- **variable_prefixn**
  
  The last line of data. Where \( n \) is the number of data lines. The number \( n \) should be the same as the contents of variable_prefix0 above.
Example

The following is an example of the MLWTO command embedded in a REXX program.

```rexx
/* REXX */
mlw0=4
mlw1='First line of text.'
mlw2='Second line of text.'
mlw3='Third line of text.'
mlw4='Fourth line of text.'
rc.af='GLBVPUT("mlw0","mlw1","mlw2","mlw3","mlw4")
"MLWTO mlw MSGID(!AFB000)"
rc.af=rc
```

On execution, the contents of variables `mlw1-mlw4` will be written to the operator console as consecutive lines of a multiple-line WTO message as follows:

```
!AFB000 hh.mm.ss OG/MVS
First line of text.
Second line of text.
Third line of text.
Fourth line of text.
```

Where `hh.mm.ss` is the time.

```rexx
rc.af='GLBVGET("mlw#")
.
.
.
rc.af='GLBVDEL("mlw0","mlw1","mlw2","mlw3","mlw4","mlw#")
.
.
```
NVALERT

Description
NVALERT causes generation of a NetView® alert. The alert is sent via VTAM and the CNM routing table unless the keyword RECEIVER is specified. In that case, the alert is sent using NetView's Program-to-Program Interface (PPI). (Can be issued from the console.)

Syntax

```
subsys NVALERT TYPE(altype) ALID('alid') DESC('code') PC('code')
```

```
HIER('hierpair',s)
```

```
USER('ua-pair')
```

```
FAIL('fa-pair')
```

```
INST('ia-pair')
```

```
TEXT(text)
```

```
RECEIVER(id)
```

subsys This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

TYPE(altype) Specifies the type of NetView alert to be generated. The possible choices are:

- **IMPD** An impending loss of availability that has not yet occurred.
- **PERF** Performance below an acceptable level.
- **PERM** Permanent loss of availability that cannot be recovered without external intervention.
- **TEMP** Temporary loss of availability which may be noticed by end users, but which is recovered without external intervention.
- **UKWN** An alert condition of unassessable severity.

ALID('alid') Specifies the ID number of the generated alert, expressed as 1-8 hexadecimal digits enclosed in single quotes. If fewer than 8 digits are specified, the value is left justified with zeros.

DESC('code') Specifies the alert description code point, expressed as 4 hexadecimal digits enclosed in single quotes.
Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NetView alert initiated.</td>
</tr>
<tr>
<td>16</td>
<td>Invalid syntax.</td>
</tr>
</tbody>
</table>

Comments

Although the AF/OPERATOR command processor checks for correct syntax, it is not possible to check for meaningful values. Before executing the NVALERT command, it is very important that all supplied values have been defined in the Hardware Monitor Code Point.
For more information about NetView alerts, see the IBM publications *SNA Formats*, *SNA Management Services Alert Implementation Guide*, and *NetView Customization Guide*.

**Example**

The following is an example of this command embedded in a REXX exec.

```
"TRAP ADD(CICSABND) WTO('DFH0612 * ABEND *') ENABLE",
 "ACTION('NVALERT TYPE(PERM) ALID('EEE98')'),
 "DESC('2000') PC('1001'),
 "USER('0111,1000'),
 "HIER('&&AOJNAME,PROD'),
 "TEXT('&&AOTEXT'))"
```

This example shows the creation of a trap for CICS abends which, when matched, will cause a NetView alert to be generated. Note that the ACTS option is not needed because an action code point is specified in the USER option. Also note the use of system match variables AOJNAME and AOTEXT to pass to NetView the jobname of the issuing address space and the full text of the WTO that caused the match. The double ampersands that precede the variable names indicate that substitution of these variables should be deferred until the WTO trap is triggered (matched).
NVISHOW

Description
NVISHOW displays the status of the AF/OPERATOR to NetView communication. (Can be issued from the console.)

Syntax
```
subsys NVISHOW
```

- **subsys**: This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The `subsys` name is assigned by the SUBSYS startup parameter.

Comments
NVISHOW indicates:
- whether NetView is active
- whether the NetView Interface receiver ID is not defined, inactive, or active
- the number of command requests which have been received
- the number of currently active NetView Interface matches

The format of the display, with an explanation of the text, appears below.

```
!AOP3207 NETVIEW INTERFACE(status)
!AOP3207 RECEIVER ID(receiver ID) STATUS(status) SUBTASK(status)
!AOP3207 COUNT(nn,mm) ACTIVE MATCHES(nn)
```

The following describes the message text:

- **NETVIEW INTERFACE(status):**
  - **ACTIVE**: NetView is available to service requests. The AF/OPERATOR receiver ID must also show active in order for AF/OPERATOR to receive commands from NetView.
  - **INACTIVE**: NetView is not currently active. No requests may be sent/received over the NetView Interface.
  - **UNDEFINED**: Either this release of NetView does not support the NetView Interface, or some internal error has made it impossible to use (for example, an error LOADING the CNMNETV module).

- **RECEIVER ID(receiver ID):**
receiver ID: The name of the AF/OPERATOR receiver ID. Currently will be subsys.

STATUS(status):

ACTIVE: The AF/OPERATOR receiver is active and may receive commands from NetView.

INACTIVE: The AF/OPERATOR receiver had been started; however, it has now been marked inactive. Either an NVISTOP command has been received, or some internal error was detected, causing a shutdown.

NOT DEFINED
AF/OPERATOR has never defined itself as a receiver and thus cannot accept commands from NetView, or the receiver has become undefined. See the NVISTART command for more information on the receiver ID and communication between NetView and AF/OPERATOR.

SUBTASK(status):

ACTIVE: The AF/OPERATOR NetView Interface subtask is active. The receiver should also be active and receiving requests.

INACTIVE: The AF/OPERATOR NetView Interface subtask is inactive. Either the interface was never started via NVISTART, or has been terminated due to NVISTOP or some internal error. The receiver should also be inactive (or undefined).

TERMINATING
The AF/OPERATOR NetView Interface subtask has terminated due to NVISTOP or some internal error; however, it has not yet been detached by the AF/OPERATOR main task.

Note: It is possible for the subtask to be active but the receiver to be undefined. This could occur if the subtask is waiting for NetView to become active.

COUNT(nn,mm):

nn,mm: The number of commands received over the NetView Interface since the interface was last started/restarted (nn), and the total number of commands since AF/OPERATOR was started (mm).

ACTIVE MATCHES(nn):

nn: The number of currently active NVI matches.
NVISTART

Description
NVISTART starts the AF/OPERATOR to NetView communication. (Can be issued from the console or by a triggered trap.)

Syntax

```
NVISTART LOG|NOLOG <subsys>
```

**subsys**  This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The **subsys** name is assigned by the SUBSYS startup parameter.

**LOG**  Turns on command logging.

**NOLOG**  Turns off command logging.

Comments
- If neither LOG nor NOLOG is specified, the trap default value (based upon the TRAPDFLT startup option set during AF/OPERATOR installation) will be used.
- The following message appears when the NetView Interface is available:

  `!AOP3214 NETVIEW INTERFACE SUCCESSFULLY STARTED`

- If NetView is not up, AF/OPERATOR will try to initiate communication every 30 seconds until it becomes available.
- The NVISTART command establishes a receiver ID with the NetView PPI. Either the first NetView defined (the primary PPI server) must be active before NVISTART is issued, or no NetViews must be active. AF/OPERATOR does not automatically detect a change of PPI control from one NetView system to another. If this change occurs, the receiver ID becomes undefined. If your site switches control of the NetView PPI from a primary to a secondary NetView, include automation on the NetView side that issues the NVISTOP and NVISTART commands to reactivate the receiver ID.
Description
NVISTOP stops the AF/OPERATOR to NetView communication. (Can be issued from the console.) NVISTOP’s effect on currently running REXX execs initiated by AOCMD() is unpredictable; however, the AOCMD function returns a non-zero function code indicating an error occurred.

Syntax

```
subsys NVISTOP subsys
```

**subsys**
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

Comments
If your site switches control of the NetView PPI from a primary to a secondary NetView, include automation on the NetView side that issues the NVISTOP and NVISTART commands to reactivate the receiver ID.
OPEN (OP)

Description
OPEN opens a previously allocated file for input-output. (Can be issued from the console.)

Syntax

```
subsys
OPEN filename
    INPUT
    OUTPUT
    DDNAME(ddname)
    APPEND
    LOCAL
    GLOBAL
    SYSTEM
```

subsys
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

filename
A valid filename. The filename (1-8 characters) must be unique within the scope (local, global, or system) specified by this command, since subsequent READ and WRITE commands will reference it. If the filename is different from the ddname in the ALLOC statement, the ddname (defined below) must reflect the one in the ALLOC statement.

INPUT
Only existing files can be opened. Output must have been previously defined. Makes it possible for the file to be read by subsequent READ commands. Files allocated with the SYSOUT option cannot be opened for input processing.

OUTPUT
Makes it possible for data to be written to the file being opened with subsequent WRITE commands. Any existing data is overwritten.

APPEND
Specifies that new data (from subsequent WRITE commands) will be appended to any existing data in the file. Use only if a sequential disk dataset is being opened. (Equivalent to JCL DISP=MOD parameter.)

DDNAME(ddname)
Specifies a ddname for the file. The ddname defaults to the filename if this option is not specified. The same ddname can be opened using different filenames.

LOCAL
Gives the file a local scope. That is, it can be referenced only from the REXX exec that opened it.

GLOBAL
Gives the file a global scope. That is, it can be referenced only from the REXX exec that opened it or from another command procedure executed by the same match. This also applies to closing the file; that is, a file opened with a scope of GLOBAL must be closed from that REXX exec or from another command procedure executed by the same match.

SYSTEM
Gives the file a system scope. That is, it can be referenced from any REXX exec.
Comments
AF/OPERATOR always transfers blocks of logical records during an I/O operation, but it passes them back to READ variables one logical record at a time.

Example
The following is an example of this command embedded in a REXX exec. This example shows how to use the OPEN command as part of a routine that opens a file, reads the records within it, and closes the file.

```plaintext
indsn = "'VALID.INPUT.DSN'"
outdsn= "'VALID.OUTPUT.DSN'"
ADDRESS APHOST
 "ALLOC DDNAME(IFILE) DSN("||indsn||") SHR"
 "ALLOC DDNAME(OFILE) DSN("||outdsn||") SHR"
 "OPEN IFILE INPUT GLOBAL"
 "OPEN OFILE OUTPUT GLOBAL"
iorecord=``
 status= 'GLBVPUT'("IORECORD")
 "READ IFILE IORECORD"
status= 'GLBVGET'("IORECORD")
DO WHILE(iorecord `==``)
 "WRITE OFILE `||iorecord||`'
 "READ IFILE IORECORD"
 status= 'GLBVGET'("IORECORD")
END
 "CLOSE IFILE"
 "CLOSE OFILE"
 "FREE DDNAME(IFILE)"
 "FREE DDNAME(OFILE)"
ADDRESS
RETURN
```
OPER issues an MVS or subsystem command. (Can be issued from the console.)

**Important**

If a match issues an OPER command containing a SYSID option, the match will be transmitted to the system indicated by the linkid before execution.

**Syntax**

```
subsys 'command'
```

- `subsys` This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The `subsys` name is assigned by the SUBSYS startup parameter.

- `'command'` The text of the command. The string can be from 1-126 characters long.

- **CONSOLE**(name)

  When `RESP` is specified, gives the name of the EMCS console from which the command seems to be issued. If this operand is omitted, a default name is used. This operand is ignored if the command is to be issued on a remote system.

- **SYSID**(linkid)

  Identifies the name of a target system where the operator commands will be transmitted and executed when system linkages have been defined. SYSID is a 1-to 8-character name that matches a LINKID defined previously with the LINK DEFINE command. If the console ID is not specified, the response to the OPER command and the authority of the MVS command are sent to the master console by default. If omitted, the command will be routed to the local system where the command is issued.

  The SYSID used for an OPER command is:
  - If the SYSID parameter is specified, then that value is used and the command is sent to the appropriate remote AF/OPERATOR system with a matching LINKID for execution.
  - If the SYSID parameter is not specified, then no SYSID value is presumed and the command is executed on the local AF/OPERATOR system.
WAIT(sec) Specifies that if all allocated consoles are in use by this AF/OPERATOR address space, the console request waits for the indicated maximum number of seconds for a console to become available. WAIT(0) specifies that the request waits as long as is necessary for a console to become available.

If this operand is omitted and all consoles are busy, a return code of 24 is issued. This operand is valid only if the command is to be issued on the local system.

TIMEOUT(pp,ss)

pp specifies in seconds (1--3600) how long to wait for the first or only response message line before assuming that no response has been issued. Default is 5 seconds or the value specified in the OPERRESP(MAXTVAL(pp,ss)) startup parameter.

ss specifies in seconds (1--3600) how long to wait for each subsequent response message line from a MLWTO before assuming that the response is complete. Default is the first non-zero value found in the following order:
- the pp value as defined above,
- the ss value specified in the startup parameter OPERRESP(MAXTVAL(pp,ss)), or
- 5 seconds

See the AF/OPERATOR Configuration and Customization Guide for more information on the OPERRESP(MAXTVAL) startup option.

While the values of TIMEOUT(0,0) will be accepted, this specification has no meaning and the defaults will be used for any value specified as zero. As these values are required to be numeric, the value (5,.) or (.5) for example are invalid and will not be accepted. You may, however, enter values such as (5,0) or (0,5) where the zero value will cause the default to be used.

MAXLINES(nnnn)

Specifies that the response is to be deemed complete when the specified number of lines has been reached. Range for nnnn is 1--9999. The default is no limit.

While the value MAXLINES (0) will be accepted, it has no meaning and the default of no limit will be used.

ENDMSG(prefix)

Specifies that the response is to be deemed complete when a line with the specified prefix is encountered in one of the response lines. The maximum length of the prefix is 20 characters. Wild-card characters are not supported. If ENDMSG(prefix) is not specified, by default, the message text will not be tested.

RESP Specifies that a set of line variables will receive the response text resulting from the command. For the OPER RESP command to work properly, the subsystem consoles must be defined with AUTH(ALL) in SYS1.PARMLIB. The TRAPRESP startup option chosen will affect whether or not responses from the OPER RESP command can be trapped by other AF/OPERATOR traps. For more information about line variables, see “Variables in the AF/OPERATOR Environment” on page 41.

NORESP Specifies that line variables will not be returned.
## Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>See page 313 for a means to determine the status of the response lines when the return code is 0.</td>
</tr>
<tr>
<td>4</td>
<td>AOCMDRSP not authorized.</td>
</tr>
<tr>
<td>8</td>
<td>Invalid MVS command length (must be 1-126).</td>
</tr>
<tr>
<td>12</td>
<td>OPER with RESP is unavailable under OG*TSO.</td>
</tr>
</tbody>
</table>
| 16          | One of the following:  
|             | - console inactive or undefined  
|             | - invalid name or ID  
|             | - no command supplied  
|             | - invalid syntax |
| 20          | AF/OPERATOR is unavailable. |
| 24          | All consoles are busy. |
| 28          | Insufficient storage for Response Block. |
| 32          | No consoles are allocated. |
| 36          | AOCMDRSP ESTAE failed. |
| 40          | Insufficient storage for private response buffer. |
| 44          | Incomplete response. |
| 48          | The named console is busy. |
| 64          | Specified SYSID is not an active link. |
| 1mmnn       | Activate failed (return code mm, reason code nn). |
| 2mmnn       | Deactivate failed (return code mm, reason code nn). |
Determining the status of the response lines for zero return codes

The REXX exec can determine which condition ended the wait for response lines by testing the AOCASE variable. AOCASE is assigned values with the following meaning:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal end of response, single line message, or ‘END’ detected for MLWTO.</td>
</tr>
<tr>
<td>1</td>
<td>TIMEOUT1(pp) value was reached.</td>
</tr>
<tr>
<td>2</td>
<td>TIMEOUT2(ss) value was reached.</td>
</tr>
<tr>
<td>3</td>
<td>MAXLINES(nnnnn) was reached.</td>
</tr>
<tr>
<td>4</td>
<td>ENDMMSG(prefix) was found.</td>
</tr>
</tbody>
</table>

Comments

- Normally, the OPER with RESP command terminates when the first end marker is received or when the default timeout specified in the OPERRESP(MAXTVAL(nn)) startup parameter expires, whichever comes first. Coding a TIMEOUT value for an OPER with RESP means the command will not be terminated until the specified number of seconds has expired. This gives you the ability to receive multiple MLWTO responses generated from a single command.

- If an OPER command is issued from within a REXX exec invoked by a TRAP CMD match, the default console ID will be that of the original console that issued the trapped command.

- OPER with RESP uses subsystem consoles. If all subsystem consoles are in use at the time a trap fires, the automation procedure must account for the failed return code (24) from the OPER command and process accordingly.

- If both RESP and CONSOLE are specified, CONSOLE is ignored.

- If both NORESP and TIMEOUT are specified, TIMEOUT is ignored.

- If SYSID is specified, OPER command output is routed for execution through the MSMF interface to the targeted site, provided the supporting network environment is defined and operational. If the RESP parameter is specified, the resulting data from the target system is sent back to the originator of the command in line variables.

- The OPER command issues any command it is directed to, through SVC 34. It does not check for the validity of a command before passing it to MVS for processing.

Most MVS DISPLAY and CONTROL commands work when issued from AF/OPERATOR. However, some may produce unpredictable results due to changes in IBM code or MVS architectural constraints. Some known MVS commands which will not work are listed below:

- K C,D ...
- K D
- K E ...
- K N,PFK ...

Example

The following is an example of this command embedded in a REXX exec.

/* REXX */

"OPER 'D A,L' RESP WAIT(5)"
VGetRC = GLBVGET("Line#")

DO Index = 1 TO Line#
   CurrentLine = "Line"Index
   VGetRC = GLBVGET(CurrentLine)
   SAY VALUE(CurrentLine)
END

RETURN

In this next example, the text of the OPER command contains a string of characters normally enclosed by single quotes ('). In this case you must use 2 single quotes when you enter the text.

"OPER 'SEND "MESSAGE TEXT",USER=(TDXXX),LOGON'"

The following is a REXX exec that issues a VARY INACT command using OPER with the RESP option.

/* REXX */
"OPER 'V NET,INACT,ID=L60AS29' RESP"
X='GLBVGET'('LINE#')
DO J = 1 TO LINE# BY 1
   CALL 'GLBVGET'('LINE"J')
   INTERPRET SAY LINE||J
END
IF LINE# >= 2 THEN
   IF LINE2 = 'L60AS29 NODE NOW INACTIVE' THEN DO
      "WTO 'VARY INACT FAILED'"
   END
   ELSE
      "WTO 'VARY INACT COMPLETED'"
EXIT
The following MSMF OPER command starts CICS on SYSB.

```
“OPER ‘S CICS’ SYSID(SYSB)”
```

If your site has enabled console communication with NetView, you may use the OPER command to issue NetView commands and CLISTs in the following format:

```
OPER ‘% nvclist/nvcmd’ CONSOLE(id)
```

The text enclosed in quotes is a NetView-identifying character followed by a NetView CLIST or command name, and any other parameters required by NetView. The console ID must have been defined during customization in the NetView parameter library.

The following is an example of an OPER with RESP command specifying a TIMEOUT value.

```
OPER ‘S CICS’ SYSID(SYSB) RESP TIMEOUT(6)
```

The 6-second TIMEOUT value insures that the subsystem consoles are freed after 6 seconds. This enables them to execute other queued OPER RESP commands.
PEER

PEER activates or terminates the Multi-System Management Facility (MSMF) interface. (Can be issued from the console.)

Two forms of the command and their parameters are described separately on the following pages.

Syntax

```
PEER subsys START parameters <
STOP FORCE IMMEDIATE
TIMEOUT
```

subsys This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

START Activates the MSMF interface.
STOP Deactivates the MSMF interface.
FORCE Deactivates the MSMF interface and any AF/OPERATOR operations using the interface.
IMMEDIATE Terminates MSMF after all active messages are sent and before new active messages arrive.
TIMEOUT(sec) The TIMEOUT option lets you specify that if your PEER STOP command does not terminate within a certain time frame, to terminate the communication. After the timeout period has expired, communication will be cancelled regardless of work in progress.

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Peer-to-peer interface is activated.</td>
</tr>
<tr>
<td>16</td>
<td>Peer-to-peer is not supported under TSO, anchor build failed, or invalid syntax.</td>
</tr>
</tbody>
</table>
PEER START

Description
PEER START activates the Multi-System Management Facility (MSMF) interface. (Can be issued from the console.)

Syntax

```
subsys  PEER START APPLID(luname) MODE(modename)
```

subsys  This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

APPLID(luname)

The application ID that defines MSMF to VTAM.

MODE(modename)

The VTAM log mode table entry definition that is to be used for VTAM LU6.2 sessions.

Comments
PEER START must be the first command executed in the PEER-PEERLINK command series.

Example
The following is an example of this command embedded in a REXX exec.

```
"PEER START APPLID(AFSYSA) MODE(AFLU62)"
```

In this example, APPLID(AFSYSA) identifies the local AF/OPERATOR system to VTAM, and MODE(AFLU62) specifies the VTAM logon mode table entry containing the default session parameters to be used by this system.

Messages like the following are displayed:

```
!AOP3001 PEER INTERFACE INITIALIZATION STARTED
!AOP3003 PEER LOCAL ACB AFSYSA SUCCESSFULLY OPENED
!AOP3005 PEER LOCAL TRANSACTION PROGRAMS DEFINED
!AOP3006 PEER INTERFACE INITIALIZATION SUCCESSFULLY COMPLETED
```
PEER STOP

Description
PEER STOP deactivates the Multi-System Management Facility (MSMF) interface. When issued, if any received WTORs have not been replied to, the matches executing those WTORs will be cancelled. In addition, a PEER STOP will be automatically issued anytime the communication terminates due to network problems. The AF STOP command always issues a PEER STOP. (Can be issued from the console.)

Syntax

```
PEER STOP [subsys] [FORCE] [TIMEOUT]<
```

**subsys**
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The `subsys` name is assigned by the SUBSYS startup parameter.

**FORCE**
Deactivates the MSMF interface and any AF/OPERATOR operations using the interface.

Without the FORCE option, PEER STOP deactivates the MSMF interface but does not deactivate any AF/OPERATOR operations using the interface. Use when the shutdown of AF/OPERATOR is prevented while PEERLINK activities are in process.

**IMMEDIATE**
Terminates MSMF after all active messages are sent and before new active messages arrive.

**TIMEOUT(sec)**
The TIMEOUT option specifies the maximum time that the interface stays active after the PEER STOP command is issued. After the timeout period has expired, outstanding WTO and command traffic will be cancelled regardless of work in progress.

The PEER STOP TIMEOUT(sec) command overrides the default timeout value of 60 seconds. The time may be increased to allow time to drain peer-to-peer messages.
Example
The following is an example of this command embedded in a REXX exec.
If the MSMF interface has been activated with an APPLID of AFSYSA, then

"PEER STOP"

does the following:
1. Deactivates the MSMF interface. The deactivation occurs after all units of work being
   processed by the system are completed.
2. Closes the VTAM access control block AFSYSA of the local system.
Messages like the following are displayed:

```
!AOP3007 PEER TERMINATION REQUEST ACCEPTED, TERMINATING
!AOP3108 PEER INTERFACE NOT QUIESED, REENTER STOP COMMAND
```
or

```
!AOP3008 PEER SUBTASK AND LOCAL LU TERMINATED
```
Description

PEERLINK defines, activates, terminates, and deletes communication links among AF/OPERATOR systems in a Multi-System Management Facility (MSMF) network. (Can be issued from the console.) Various forms of the command are described separately in subsequent sections.

Syntax

```
subsys PEERLINK DEFINE parameters
         DELETE parameters
         DISPLAY parameters
         START parameters
         STOP parameters
```

subsys This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The `subsys` name is assigned by the SUBSYS startup parameter.

DEFINE Defines a remote MSMF communications link.
DELETE Erases a link’s PEER definition from the PEER definition table, which causes the link (and all internal support programs) to be removed from the network.
DISPLAY Displays the current session status of the specified link(s).
START Initiates a session between two previously defined remote MSMF sites (activates the communication link).
STOP Terminates the session in progress between two MSMF sites (terminates the communication link).

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Peer-to-peer network communications are established.</td>
</tr>
<tr>
<td>16</td>
<td>Peer link entry is not found, link is already active, or COM1 is not initialized.</td>
</tr>
</tbody>
</table>
Comments

The PEERLINK commands must be executed in the following order:

- PEERLINK DEFINE
- PEERLINK START
- PEERLINK STOP
- PEERLINK DELETE

PEERLINK DISPLAY can be executed at any time.
PEERLINK DEFINE

Description
PEERLINK DEFINE defines a communication link between AF/OPERATOR systems through the MSMF interface. (Can be issued from the console.)

AF/OPERATOR loads the necessary link support and establishes queues in preparation for an active communication link.

Syntax
```
subsys PEERLINK DEFINE LINKID(linkid)

subsyst
```

```
subsyst APPLID(luname) MODE(modename)
```

```
subsyst PEERLINK DEFINE LINKID(linkid)

subsyst > APPLID(luname) >>

subsyst MODE(modename) <=
```

subsyst This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsyst name is assigned by the SUBSYS startup parameter.

LINKID(linkid)
The LINKID you specified in the LINKID startup parameter.

APPLID(luname)
The name of the remote communication link (luname).

MODE(modename)
The name of the VTAM log mode table that contains the session parameters and characteristics. If no modename is specified, the modename used in the PEER START command is used.

Comments
This must be the first PEERLINK command executed.

Example
The following is an example of this command embedded in a REXX exec.
```
"PEERLINK DEFINE LINKID(SYSB) APPL(AFSYSB) MODE(AFLU62)"
```
In this example, the DEFINE statement specifies SYSB as the remote system’s LINKID. AFSYSB is the VTAM APPLID name, and the mode table entry name is AFLU62. PEERLINK START activates link SYSB.

Messages like the following are displayed.
```
AOP3105 PEER LINK DEFINE COMMAND FOR SYSB SUCCESSFUL
```
PEERLINK DELETE

Description
PEERLINK DELETE erases a definition created by the PEERLINK command from the PEER definition table. (Can be issued from the console.)
This removes the defined link, and all internal support programs, from the network.

Syntax

```
subsys PEERLINK DELETE LINKID(linkid)..<subsys>
```

- **subsys**: This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The `subsys` name is assigned by the SUBSYS startup parameter.

- **LINKID(linkid)**: The LINKID you specified in the LINKID startup parameter.

Comments
This command is rejected if the selected link is active. A PEERLINK STOP command must be executed before a PEERLINK DELETE.

Example
The following is an example of this command embedded in a REXX exec.

```
"PEERLINK DELETE LINKID(SYSB)"
```

A message like the following is displayed:

```
!AOP3103 PEER LINK REQUESTED LINK ENTRY SYSB DELETED SUCCESSFULLY
```
Description

PEERLINK DISPLAY shows the current session status of the requested MSMF link. (Can be issued from the console.)

Syntax

```
PEERLINK DISPLAY LINKID (linkid) subsys
```

- **subsys**: This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The `subsys` name is assigned by the SUBSYS startup parameter.

- **LINKID(linkid)**: The LINKID you specified in the LINKID startup parameter.

- **LINKID(ALL)**: Displays all defined (active and inactive) links. (This is equivalent to supplying a pattern of `*`.) This is the default.

- **ACTIVE**: Displays only active links.

- **INACTIVE**: Displays only inactive links.

Example

The following is an example of this command embedded in a REXX exec.

```
"PEERLINK DISPLAY LINKID(ALL) ACTIVE"
```

Messages like the following are displayed:

```
!AOP3102 PEER LINK SYSB SEND COUNT(678), RECEIVE COUNT(542), STATUS(ACTIVE)
!AOP3102 PEER LINK SYSC SEND COUNT(2), RECEIVE COUNT(67), STATUS(ACTIVE)
2 ITEMS SHOWN
```

**COUNT** represents the number of transactions processed during the current session.
PEERLINK START

Description
PEERLINK START activates a MSMF communications link by initiating the link startup procedure contained in a definition created by a PEERLINK DEFINE command. (Can be issued from the console.)

Syntax

```
PEERLINK START LINKID(linkid) <subsys>
```

subsys
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

LINKID(linkid)
The LINKID you specified in the LINKID startup parameter.

Comments
The communication session continues as long as the link is active (until a PEERLINK STOP command is received).

Example
The following is an example of this command embedded in a REXX exec.

```
"PEERLINK START LINKID(SYSB)"
```

A message like the following appears:

```
!AOP3101 PEER LINK SYSB STARTED SUCCESSFULLY
```
Description
PEERLINK STOP terminates the communication session in progress on the specified link. (Can be issued from the console.)

Syntax
```
subsys PEERLINK STOP LINKID(linkid) < FORCE
```

subsys  This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

LINKID(linkid)

The LINKID you specified in the LINKID startup parameter.

FORCE  Terminates all active units of work currently using the specified link.

Comments
PEERLINK STOP must be specified before a PEERLINK DELETE. Termination occurs when the next unit of work processed by the MSMF is completed.

Example
The following is an example of this command embedded in a REXX exec.
```
"PEERLINK STOP LINKID(SYSB)"
```

The following messages are displayed if execution is successful:
```
!AOP3029 PEERLINK STOP COMMAND ACCEPTED FOR SYSB
!AOP3032 PEERLINK STOP COMMAND COMPLETED SUCCESSFULLY, SYSB NOW INACTIVE
```
You can use the Programmerless Open VTAM Interface (POVI) commands to accelerate scripts. These commands are listed in “Programmerless Open VTAM Interface Commands” on page 423 and “Programmerless Open VTAM Interface Operator Commands” on page 435.
READ

Description
READ retrieves the next line or block from a file that has been previously opened for input and stores it in a variable.

Syntax

```
READ filename varname [ NOTRUNC ]
```

filename The name of the file. The file must already be open (see the OPEN command) and cannot be a SYSOUT file.

varname A variable name. This variable will contain the retrieved data. If the variable does not exist, AF/OPERATOR creates a local variable with the specified name of up to 255 bytes.

NOTRUNC Prevents the stripping of leading and trailing blanks from the input record. If NOTRUNC is omitted then the blanks are removed.

Comments
- If you read past the end-of-file, a null string is stored in the variable. A null record usually means end-of-file.
- Your input file must not contain any blank records, otherwise AF/OPERATOR detects a premature end of file.
- The READ command transfers a record from a file into a local variable only. If you use the READ command in a REXX exec, you must first give READ a global scope as REXX execs can access only system or global variables. To make READ global, insert the following lines into your exec:
  
  ```
  VARNAME = ''
  R = GLBVPUT("VARNAME")
  "READ SYSIN VARNAME"
  R=GLBVGET("VARNAME")
  ```
Example

The following is an example of this command embedded in a REXX exec.

```rexx
/* REXX */
indsn = "'VALID.INPUT.DSN'"
outdsn= "'VALID.OUTPUT.DSN'"
ADDRESS APHOST
   "ALLOC DDNAME(IFILE) DSN(''||indsn||') SHR"
   "ALLOC DDNAME(OFILE) DSN(''||outdsn||') SHR"
   "OPEN IFILE INPUT GLOBAL"
   "OPEN OFILE OUTPUT GLOBAL"
   /* Note: Input file must not contain any blank records, */
   /* otherwise a premature end of file is detected. */
   iorecord=''
status= 'GLBVPUT'('IORECORD')
"READ IFILE IORECORD"
status= 'GLBVGET'('IORECORD')
DO WHILE(iorecord ^= ' ')
   "WRITE OFILE ' '||iorecord||' '"
   "READ IFILE IORECORD"
   status= 'GLBVGET'('IORECORD')
END
"CLOSE IFILE"
"CLOSE OFILE"
"FREE DDNAME(IFILE)"
"FREE DDNAME(OFILE)"
ADDRESS
RETURN
```
SHOW (SH)

Description
SHOW displays information about traps, variables, active matches, files, and other aspects of AF/OPERATOR. (Can be issued from the console.)

Because of its complexity, various forms of the command are described separately in subsequent sections.

Syntax

```
SHOW subsystem
         | ENQUEUE parameters
         | FILES parameters
         | HOSTNAME parameters
         | LOGONS parameters
         | MAINT parameters
         | MATCHES parameters
         | OG parameters
         | RSVS parameters
         | SCONS parameters
         | SSCTS parameters
         | STATS parameters
         | SVCS parameters
         | TRAPS parameters
         | TRAPSTITLEFORMAT parameters
         | VARS parameters
```

**subsys**
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsystem name is assigned by the SUBSYS startup parameter.

**FILES**
Displays information about AF/OPERATOR files.

**HOSTNAME**
Displays the AF/OPERATOR trusted hostnames list.

**LOGONS**
Displays VTAM links initiated by previous LOGON commands and indicates their status.

**MAINT**
Displays the maintenance levels of specified programs and the current cumulative maintenance tape level.
SHOW (SH)

MATCHES            Displays all active matches; that is, those matches which AF/OPERATOR is currently processing.
OG                  Displays the name, status, and relative order of all AF/OPERATOR multiple address space support subsystems.
RSVS                Displays shared variables from the RSV pool to the console and AF/OPERATOR log.
SSCONS              Displays subsystem console definitions and current status.
SSCTS               Displays the subsystems in a data center that use a subsystem communications table (SSCT) and the type of subsystem calls that each processes.
STATS               Displays various AF/OPERATOR statistics.
SVCS                Displays SVC entries.
TRAPS               Displays trap definitions.
VARS                Displays variable names and their current values.
TRAPS TITLE         Customizes a SHOW TRAPS command.

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Data found to show.</td>
</tr>
<tr>
<td>4</td>
<td>No data found to show.</td>
</tr>
<tr>
<td>12</td>
<td>Required module not linked.</td>
</tr>
<tr>
<td>16</td>
<td>Invalid syntax.</td>
</tr>
</tbody>
</table>

Comments
The responses to the SHOW command are displayed on the console or TSO terminal from which the command was issued.
SHOW ENQUEUE

**Description**

The SHOW ENQUEUE command shows the resources currently enqueued with the enqueueing match number, and lists any matches waiting for the resource.

**Syntax**

```
SHOW ENQUEUE MAJOR(qname) MINOR(rname)
```

- **subsys**
  The name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter. This operand is only required when you issue the command from the console.

- **MAJOR(qname)**
  A 1-to-8 character identifier, used in conjunction with rname, to identify which enqueued resources are to be displayed.

- **MINOR(rname)**
  A 1-to-255 character identifier, used in conjunction with qname, to identify which enqueued resources are to be displayed.

- **SCOPE()**
  Limits the display to enqueues of the specified scope.

- **SPECIFIC()**
  Indicates the type of match, as follows:

  - **YES**
    Indicates that an exact match of resource name is required. This is the default.

  - **NO**
    Indicates a generic match or resource name. A match occurs against any resource whose name starts with the specified resource name.

- **LIMIT()**
  Limits the number of resources displayed by a single SHOW ENQUEUE command, as follows:

  - **nnn**
    Can be a number from 1 to 999

  - **200**
    The display will include up to 200 resources.
Example
The following is an example of this command in a REXX exec:

```/*- REXX -*/
 "SHOW ENQUEUE SCOPE(ALL) MAJOR(GHALL) MINOR(TEST)"
```

A typical response to this command follows:

```
!AOP0210 --- LIST OF ENQUEUES, SCOPE=ALL ---
!AOP0201 RESOURCE MAJOR(GHALL)
!AOP0201 -------- MINOR(TEST)
!AOP0201 -------- SCOPE(STEP) CONTROL(SHR) STATUS(OWN)
!AOP0201 -------- SYSTEM(SYSG) JOB(GHALLOG) ASID(0038) TCB(006947A8) MATCH(4)
!AOP0201 RESOURCE MAJOR(GHALL)
!AOP0201 -------- MINOR(TEST)
!AOP0201 -------- SCOPE(STEP) CONTROL(SHR) STATUS(OWN)
!AOP0201 -------- SYSTEM(SYSG) JOB(GHALLOG) ASID(0038) TCB(00694368) MATCH(5)
!AOP0208 3 ITEMS SHOWN
```
SHOW FILES

Description
SHOW FILES displays information about AF/OPERATOR files. (Can be issued from the console.)

Syntax

```
subsys SHOW FILES (pattern,s) OPEN ALLOCATED
```

**subsys**
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The *subsys* name is assigned by the SUBSYS startup parameter.

**(pattern,s)**
A name pattern intended to match one or more file names. You can use up to 32 patterns.

**OPEN**
Displays only those files that have been allocated with the ALLOC command.

**ALLOCATED**
Displays only files allocated to AF/OPERATOR. The files can be in the AF/OPERATOR startup JCL or dynamically allocated with the ALLOC command.

Example
The following is an example of this command embedded in a REXX exec.

```
"SHOW FILES(SYSUT1)"
```

A typical response to this command follows:

```
--- LIST OF OPEN FILES ----
SYSUT1 MODE(INPUT) DDNAME(SYSUT1) SCOPE(SYSTEM)
--------- BUFSIZE(23440) MATNUM(33) EOF(NO)
--------- DSN(USER05) COUNT(75)
1 ITEM SHOWN
```
SHOW HOSTNAME

Description
SHOW HOSTNAME displays the trusted hostnames list. The trusted hostnames list is the list of authorized host names or IP addresses that can connect to a particular AF/OPERATOR. These entries are stored in the HOSTNAME member of the RKANPAR DD. The list is created by reading this member at AF/OPERATOR startup.

Syntax

Example
A typical response to the SHOW HOSTNAME command follows:

```
!AOP0210 ---LIST OF TCPIP TRUSTED HOSTNAMES ---
!AOP0201 TRUSTED HOSTNAME :ADDRESS=192.168.113.110
!AOP0201 TRUSTED HOSTNAME :ADDRESS=192.168.113.111
!AOP0201 TRUSTED HOSTNAME :ADDRESS=SYSC(192.168.113.113) STATIC
!AOP0201 TRUSTED HOSTNAME :ADDRESS=SYSD(192.168.113.114) DYNAMIC
!AOP0201 TRUSTED HOSTNAME :ADDRESS=192.168.113.112
!AOP0201 TRUSTED HOSTNAME :ADDRESS=SYSF.ZYX.COM(192.168.113.118) STATIC
!AOP0201 TRUSTED HOSTNAME :ADDRESS=USER1.ZYX.COM(192.168.10.10) DYNAMIC
!AOP0201 TRUSTED HOSTNAME :ADDRESS=USER2.ZYX.COM DYNAMIC
!AOP0201 TRUSTED HOSTNAME :ADDRESS=USER3.ZYX.COM DYNAMIC
!AOP0201 TRUSTED HOSTNAME :ADDRESS=192.168.113.128
!AOP0201 TRUSTED HOSTNAME :ADDRESS=SYSG STATIC
!AOP0201 TRUSTED HOSTNAME :ADDRESS=192.168.113.120
!AOP0208 12 ITEMS
```

Entries in the list that were defined to AF/OPERATOR as a hostname rather than a dotted decimal indicate whether they are a dynamic or static definition. Any hostname entry in the list that has been resolved also displays the resolved IP address.

For example, in the list above, SYSC has been resolved to IP address 192.168.113.113 and SYSD has most recently been resolved to IP address 192.168.113.114. It has not yet been necessary to resolve SYSG.
SHOW LOGONS

Description
SHOW LOGONS displays VTAM links initiated by previous LOGON commands and indicates their status. (Can be issued from the console.)

Syntax

```
SHOW LOGONS <subsys>
```

$subsys$ This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The $subsys$ name is assigned by the SUBSYS startup parameter.

Example
The following is an example of this command embedded in a REXX exec.

```
"SHOW LOGONS"
```

A typical response to this command follows:

```
Environment(OCCI) Applid(OCVTAMB) Type(OMCICS)
Userid(AFOPER) Userdata(CICS=$CICSCIM) Interval(00:05:00)
Environment(SYSD) Applid(OMVTAMD) Type(OMMVS)
Userid(AFOPER) Userdata(*NONE*) Interval(00:02:00)
Environment(SYSB) Applid(OMVTAMB) Type(OMMVS)
Userid(AFPERF) Userdata(*NONE*) Interval(00:03:00)
Environment(SYSA) Applid(OMVTAMA) Type(OMMVS)
Userid(AFPERF) Userdata(*NONE*) Interval(00:05:00)
4 Environments logged on
```
SHOW MAINT

Description
SHOW MAINT displays the maintenance levels of specified programs and the current cumulative maintenance tape level. (Can be issued from the console.)

Syntax

```
SHOW MAINT(pattern) <subsyst>
```

**subsyst**
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The `subsyst` name is assigned by the SUBSYS startup parameter.

**(pattern)**
A 1- to 8-character module name. You can use AF/OPERATOR pattern matching characters to display groups of modules or an asterisk (*) to display all modules.

Comments
- SHOW MAINT displays only the information for executable programs of the base product.
- Candle Customer Support may ask you to issue this command to verify your current maintenance level.

Example
The following is an example of this command issued from the console.

```
SHOW MAINT(KAB*)
```

Messages such as the following are displayed:

```
SHOW MAINT(KOGRX*)
!AOP3230                : VER  MM/DD/YY HH.MM RMID
!AOP3230 MODULE KOGRXLNE: V999 01/16/01 00.12 AKOG999
!AOP3230 MODULE KOGRXVAR: V999 01/16/01 00.13 AKOG999
!AOP3230 MODULE KOGRXRED: V999 01/16/01 00.12 AKOG999
!AOP3230 MODULE KOGRXVTX: V999 01/16/01 00.13 AKOG999
!AOP3231 4 MODULE(S) SHOWN
```
SHOW MATCHES

Description
SHOW MATCHES displays all active matches; that is, matches that AF/OPERATOR is currently processing. (Can be issued from the console.)

Syntax

subsys
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

(number,s)
The number of a match. Using one or more numbers specifies that only definitions of traps with those numbers will be displayed. 

Note: Leave no blanks between MATCHES and the parenthesis before number.

CMD
Displays only matches of CMD traps.

COM
Displays only communication matches. A communication match is an AF/OPERATOR match that has been scheduled from another system using the COMSDRCV REXX function with a datatype specification of EXEC.

NVI
Displays only NetView Interface matches.

DOM
Displays only matches of DOM traps.

TOD
Displays only matches of TOD traps.

WTO
Displays only matches of WTO traps.

xtype
Displays only matches of a type of OMEGAMON exception traps. The supported OMEGAMONs are:

XO2 OMEGAMON II for DB2
XOC OMEGAMON for CICS
SHOW MATCHES

XOI OMEGAMON for IMS
XOM OMEGAMON for MVS

INT Displays all internal matches and is used for diagnostic purposes. This is the default.

ACTIVE Displays only matches that have been triggered.

ALL Displays both queued and active matches.

QUEUED Displays only active matches for which processing has not yet started.

TITLE('title/s')

Used with 1- to 8-title parameters in its subfield, specifies up to 8 title lines that will be displayed before any detail information lines are written. A title line can contain literal text and symbolic variables.

FORMAT('line/s')

Used with 1- to 8-line parameters in its subfield, specifies up to 8 lines that will be used to format the detail information. A format line can contain literal text and symbolic variables.

Comments

- Each match is assigned a unique match number that can be used in the SHOW MATCHES command. To discover this number, use SHOW MATCHES to display all matches or all matches of the appropriate kind (omit the number operand). You can also examine the AF/OPERATOR message log. In the log, each message is prefixed with the number of the match that issued the message.

- Because matches are processed with great speed, queued matches are rarely seen. If the same match appears in successive SHOW MATCHES commands, it is likely that a hung match condition exists.

Example

The following are examples of this command embedded in REXX execs.

“SHOW MATCHES” Displays all active matches.
“SHOW MATCHES(2)” Displays active match number 2.
“SHOW MATCHES COM” Displays all communication matches.
“SHOW MATCHES WTO” Displays only active matches triggered by a WTO trap.
“SHOW MATCHES ALL” Displays all queued and active matches.
“SHOW MATCHES CMD TITLE('EXAMPLE')” Displays all CMD matches using EXAMPLE as a title.

“SHOW MATCHES CMD TOD” Displays all CMD and TOD matches.

A typical response to this command follows:
A typical response to the command SHOW MATCHES COM follows:

SHOW MATCHES Title Variables
The following variable names can be used in title strings for SHOW MATCHES commands:

-&SHOWSTS  ACTIVE, QUEUED, WAITING, or ALL
-&SHOWTYPE  CMD, TOD, WTO, XO2, XOC, XOI, XOM, or combination

SHOW MATCHES Format Variables
The following variable names can be used in FORMAT strings for SHOW MATCHES commands:

-&AOTTYPE  Type of trap that caused the match (3 characters)
-&ASID  Address space ID (1 to 5 digits)
-&CMD  Currently executing command (0 to 255 characters)
-&CMDFILE  Name of the currently executing command file
-&CONS  Console ID (0 to 3 digits)
-&DESCCD  List of descriptor codes separated by commas
-&EXEFILE  Name of the currently executing REXX file
-&JOBCLASS  Job class (1 character)
-&JOBNAME  Job name (1 to 8 characters)
-&JOBNUM  JES job number (1 to 5 digits)
-&JOBTYPE  IMS, JOB, STC, or TSU for CMD and WTO, and O2, OC, OI, or OM for xtypes
-&MATNUM  Match number (1 to 8 digits)
-&OPTS  WAIT or CONTINUE may be followed by ERROR
-&ROUTCD  List of route codes separated by commas
-&STATUS  QUEUED, ACTIVE, or WAITING
SHOW MATCHES Format Variables for WTO Traps

The following variable names can be used in FORMAT strings for SHOW MATCHES commands. They apply only to WTO trap matches and return a null string if used for other types of matches:

- **&STMNUM**: Statement number currently executing
- **&SYSID**: System ID (1 to 8 characters)
- **&TEXT**: Message or command text (1 to 255 characters)
- **&TRAPNAME**: Name of trap (0 to 8 characters)
- **&TYPE**: CMD, TOD, WTO, XO2, XOC, XO1, or XOM
- **&USERFLD**: Value of AOMATUSR field (1 to 8 characters)
- **&AREAID**: Console area ID (1 character)
- **&CTLFLG**: Control flags (4 hex digits)
- **&DSPFLG**: Disposition flag (2 hex digits)
- **&ECBFLG**: ECB flags (2 hex digits)
- **&HCPYID**: Hard copy ID (4 digits)
- **&LINTYP**: Line type flags (2 hex digits)
- **&MCSFLG**: MCS flags (4 hex digits)
- **&MLWFLG**: MLWTO flags (2 hex digits)
- **&MSCFLG**: Miscellaneous flags (2 hex digits)
- **&RPLYID**: Reply ID for WTORs only (2 digits)
- **&STSFLG**: Status flags (2 hex digits)
- **&VERSN**: WTO version (2 hex digits)
- **&WTONUM**: WTO number (1 to 8 digits)
SHOW OG

Description
Displays the name, status, and relative order of all AF/OPERATOR multiple address space support subsystems. (Can be issued from the console.)

Syntax

```
SHOW OG [subsys] [*] [local system name]
```

**subsys**  This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

**[*]**  Shows all AF/OPERATORs. This is the default.

**local system name**

_  Shows the AF/OPERATOR specified in subsys.

Example
The following is an example of this command embedded in a REXX exec.

**“SHOW OG”**

A typical response to this command follows:

```
!AOP0202 SSID LINKID ORD PRODUCT JOBNAME STATE DATE TIME
!AOP0203 0999 SYSA 000 OMC GATEWAY OGMVS00 UP 01.043 22:19
!AOP0203 0991 SYSA0991 000 OMC GATEWAY OGMVS01 UP 01.044 01:35
!AOP0203 0992 SYSA0992 000 OMC GATEWAY DOWN 01.044 01:37
!AOP0203 0993 SYSA0993 000 OMC GATEWAY TESTOG5 UP 01.044 09:45
!AOP0208 4 ITEMS SHOWN
```
SHOW RSVS

Description
SHOW RSVS displays shared variables from the RSV pool. It displays to the console and to the AF/OPERATOR log. (Can be issued from the console.)

Syntax

```
subsys
SHOW RSVS ('text')
NAME
subsys

LIMIT(200)
POOL (SYSTEM) (SYSPLEX)

NODATA
LIMIT(nnn)

COUNT
PATTERN
``` 

subsys
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsystem name is assigned by the SUBSYS startup parameter.

'text'
The variable name you want to display. This can be a variable name, a compound variable group, or a stem.

NAME
The default if neither a name nor a stem is specified. Indicates that 'text' is a unique variable name or a CVG root.

NODATA
Displays only the name of the matching variables. If not specified, 256 bytes of the data from the matching variable is displayed.

LIMIT
Limits the number of variables displayed. LIMIT(0) means display all matching variables. If not specified, the default is LIMIT(200).

POOL
The name of the shared pool to use. Values for POOL may be either SYSTEM or SYSPLEX. The default is POOL(SYSTEM).
SHOW RSVS

COUNT
Indicates that the detail lines of the display are to be suppressed. The final line showing the count of RSVs that matched the name or pattern is still displayed.

The use of COUNT causes the default limit of 200 to be ignored, so that the count of RSVs shown is the true total count of RSVs matching the name or pattern. However, if the LIMIT keyword is explicitly coded, then the value coded for the LIMIT keyword is used.

PATTERN
Indicates that the 'text' value is to be interpreted as a pattern rather than as a specific shared variable name. This allows the SHOW RSVS command to display all shared variables that match the name or pattern.

There are some limitations on the shared variable names that can be displayed using the PATTERN option. Because REXX shared variable names may contain characters that AF/OPERATOR uses as part of its normal scanning operation, certain names can conflict with the wild card characters normally used in AF/OPERATOR pattern matching. Therefore, it is strongly recommended that you use patterns that comprise only the characters A through Z and 0 through 9 followed by an asterisk. Other combinations may work, but leave open the possibility that some variables will not be found.

The REXX shared variable name scan invoked using the PATTERN option is the same as that used by AF/OPERATOR trap pattern matching. Thus, it is possible to use the escape character ~ (tilde) to match certain characters that might otherwise be unmatchable. You should exercise caution when creating matching patterns using the PATTERN option to ensure that you are retrieving exactly those variables you are interested in.

Comments
The SHOW RSVS command does not accept patterns unless the PATTERN option is selected. In that case, certain limitations apply. Because REXX variable names can contain pattern characters such as *, $, and #, there is no way to determine whether the special characters are part of a pattern or part of a variable name.

However, you can use a stem prefix instead to match multiple names.

Return Codes
The SHOW RSVS command returns one of the following codes upon completion.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>4</td>
<td>No variables found.</td>
</tr>
<tr>
<td>8</td>
<td>No variables found.</td>
</tr>
<tr>
<td>16</td>
<td>A parameter is missing or misspelled.</td>
</tr>
<tr>
<td>64</td>
<td>RSV pools have not been created.</td>
</tr>
</tbody>
</table>
Example

A typical use of this command follows:

```
SHOW RSVS LIMIT(0)
```

A typical response to this command follows:

```
NAME=RSV.GLOBAL_VARIABLE_1
VALUE=This is the value of RSV.GLOBAL_VARIABLE_1

NAME=RSV.GLOBAL_VARIABLE_LAST
VALUE=VALUE OF THE LAST RSV VARIABLE
```

**Note:** There is one system pool for each AF/OPERATOR address space on a system.

The next example shows the use of SHOW RSVS with the PATTERN option. The first part of the example shows a portion of a REXX EXEC that adds two variables to the REXX shared variable pool.

```
/*   REXX EXEC   */
PAGECT = 1
PAGE1 = 123456781
RC3 = SHARVPUT('SYSTEM','PAGE1','CKPT')
RC4 = SHARVPUT('SYSTEM','PAGECT','CKPT')
```

This portion of the example illustrates the SHOW RSVS command using the PATTERN option to display the shared variables returned by the EXEC.

**SHOW RSVS (PAGE*) PATTERN**

The output of this command is:

```
NAME=PAGE1
VALUE=123456781
NAME=PAGECT
NAME=PAGE1
VALUE=1
2 VARIABLES SHOWN
```
SHOW SSCONS

Description
SHOW SSCONS displays subsystem console definitions and their current status. (Can be issued from the console.) The SHOW SSCONS command has two (2) aliases:

- SHOW EMCS
- SHOW CONSOLES

Syntax

```
SHOW SSCONS <subsys>
```

`subsys` This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The `subsys` name is assigned by the SUBSYS startup parameter.

Example
The following is an example of this command embedded in a REXX exec.

```
"SHOW SSCONS"
```

A typical response to this command follows:

```
4 SUBSYSTEM CONSOLES DEFINED TO OG/MVS
SUBSYSTEM CONSOLE 10 IS NOT IN USE
SUBSYSTEM CONSOLE 11 IS NOT IN USE
SUBSYSTEM CONSOLE 12 IS NOT IN USE
SUBSYSTEM CONSOLE 13 IS NOT IN USE
```
SHOW SSCTS

Description
SHOW SSCTS displays the subsystems using a subsystem communications table (SSCT), and the type of subsystem calls that each processes. (Can be issued from the console.)

Syntax

```
subsys SHOW SSCTS (pattern,s) ACT option/s DETAIL
```

where the options are:

- **AG**  Allocation group subsystem requests
- **AL**  SYSIN/SYSOUT allocation
- **CA**  Common allocation
- **CF**  Failing start command
- **CI**  Converter subsystem exit
- **CM**  Command
- **CS**  Cancel a job
- **CU**  Common unallocation
- **DA**  Open/close/ckpt/restart a subsystem dataset
- **DD**  Change ddname
- **DM**  Delete operator message
- **DR**  DDR activity
- **DY**  Dummy option
- **EN**  End of list
- **ET**  Exception handling
- **JS**  Job state
- **JT**  Job termination
- **MO**  Message output
- **MS**  Message status
- **NQ**  Non-queueing
- **RQ**  Request queue
- **RR**  Real time
- **SI**  System interface
- **SO**  System output
- **US**  User subsystem
- **VS**  Virtual storage
- **WT**  Waiting event

**subsys**
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

**pattern,s**
A name pattern intended to match one or more SSCT names. You can use up to 32 patterns.

**ACT**
Displays only active SSCTs (those with non-zero SSVT pointers).

**INACT**
Displays only inactive SSCTs (those with zero SSVT pointers).

**option/s**
Restricts the display to subsystems that support the following functions:
SHOW SSCTS

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DY</td>
<td>Dynamic allocation</td>
</tr>
<tr>
<td>EN</td>
<td>End of address space</td>
</tr>
<tr>
<td>ET</td>
<td>End-of-task</td>
</tr>
<tr>
<td>JS</td>
<td>Job selection</td>
</tr>
<tr>
<td>JT</td>
<td>Job termination</td>
</tr>
<tr>
<td>MO</td>
<td>SS volume control</td>
</tr>
<tr>
<td>MS</td>
<td>MSS message task exit</td>
</tr>
<tr>
<td>NQ</td>
<td>Change enqueue use attribute</td>
</tr>
<tr>
<td>RQ</td>
<td>Re-enqueue a job</td>
</tr>
<tr>
<td>RR</td>
<td>Request/return a job ID</td>
</tr>
<tr>
<td>SI</td>
<td>Step initiation</td>
</tr>
<tr>
<td>SO</td>
<td>Process SYSOUT</td>
</tr>
<tr>
<td>US</td>
<td>Validate remote destination user ID</td>
</tr>
<tr>
<td>VS</td>
<td>Verify subsystem name</td>
</tr>
<tr>
<td>WT</td>
<td>WTO message</td>
</tr>
</tbody>
</table>

**DETAIL**
Includes a list of functions supported by the subsystem.

**Example**
The following is an example of this command embedded in a REXX exec.

"SHOW SSCTS ACT EN ET"

In response to SHOW SSCTS, messages like the following appear:

```
--SSCTS--  NAME       -STATUS-          DESCRIPTION
00C86700  JES2       ACTIVE           JOB ENTRY SUBSYSTEM
00C860C8  MSTR      ACTIVE           MASTER SUBSYSTEM
00C866D8  ACF2      ACTIVE           ACF2 - ACCESS CONTROL FACILITY
00C86670  CICS      ACTIVE           CUSTOMER INFORMATION CONTROL SYSTEM
00C86648  DBR3      ACTIVE           
00C86440  IRLM      ACTIVE           
00C6D138  NETV      ACTIVE           
7 ITEMS SHOWN
```

You may customize the subsystem name descriptions using the AOSSNTBL utility. See the AF/OPERATOR Configuration and Customization Guide for more information.
SHOW STATS

Description
SHOW STATS displays various AF/OPERATOR statistics. (Can be issued from the console.)

Once you have applied the Statistics PTF and specified the appropriate startup parameters, you can use the SHOW STATS command to determine if the system is collecting the SMF events that you have specified.

Syntax

```
SHOW STATS <subsys>
```

In response to SHOW STATS, messages like the following appear:

```
SHOW STATS
!AOP2021 OM Gateway   ACTIVITY SUMMARY REPORT
!AOP2021 product      --- WTO/WTOR COMMAND DOM
!AOP0280 # MONITORED : 3614    78   113
!AOP0280 # SELECTED : 0      11    0
!AOP0280 % SELECTED : 0      14    0
!AOP0280 # SUPPRESSED : 0      11    0
!AOP0280 % SUPPRESSED : 0      14    0
!AOP0280 # ALTERED:   0       0    0
!AOP0280 # Time-of-Day Events Triggered: 2
!AOP0280 SMF STATS(COLLECT(NONE),SMFREC(200))
!AOP0280 SMF STATS CURRENTLY: INACTIVE
```

These messages have the following meanings:

**subsys**
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The `subsys` name is assigned by the SUBSYS startup parameter.

**# MONITORED**
All WTOs and commands issued through the subsystem interface are totalled, including those of AF/OPERATOR.

**# SELECTED**
The number of WTO/WTORs and commands that matched enabled traps.

**% SELECTED**
The percentage of WTO/WTORs and commands on the system that matched enabled traps.

**# SUPPRESSED**
The number of WTO/WTORs and commands suppressed.
SHOW STATS

% SUPPRESSED
The percentage of WTO/WTORs and commands on the system suppressed.

# ALTERED
The number of WTO/WTORs altered, for example, messages with descriptor codes changed.

# Time-of-Day
The number of Time-of-Day events that have been triggered.
SHOW SVCS

Description
SHOW SVCS displays the SVC entries on the SVC table in use. (Can be issued from the console.)

Syntax

```
subsys SHOW SVCS (number,s) TYPE(typenum,s)
```

subsys
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

(number,s)
An SVC number. You can specify from 1 to 32 SVC numbers, each in the range 0-255. If no numbers are specified, all 255 SVC numbers are displayed.

TYPE(typenum,s)
Displays only SVC numbers of one or more of the specified types. Type 5 represents unused SVC entries.
Example

The following is an example of this command embedded in a REXX exec.

“SHOW SVCS(6,10) TYPE(1,2)”

Portions of a typical response to this command follow:

<table>
<thead>
<tr>
<th>ENTRY</th>
<th>WORD2</th>
<th>SVC</th>
<th>DESCRIPTION</th>
<th>TYP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0FFBA10</td>
<td>00008000</td>
<td>0</td>
<td>EXCP/XDAP</td>
<td>1</td>
</tr>
<tr>
<td>80FF98CE</td>
<td>00008000</td>
<td>1</td>
<td>WAIT/WAITR/PRTOV</td>
<td>1</td>
</tr>
<tr>
<td>80FF6628</td>
<td>00008000</td>
<td>2</td>
<td>POST</td>
<td>1</td>
</tr>
<tr>
<td>81179FD0</td>
<td>00808000</td>
<td>3</td>
<td>EXIT</td>
<td>1</td>
</tr>
<tr>
<td>811C486A</td>
<td>00008000</td>
<td>4</td>
<td>GETMAIN</td>
<td>1</td>
</tr>
<tr>
<td>811C486A</td>
<td>00008000</td>
<td>5</td>
<td>FREEMAIN</td>
<td>1</td>
</tr>
<tr>
<td>8116D000</td>
<td>80008000</td>
<td>6</td>
<td>LINK</td>
<td>2</td>
</tr>
<tr>
<td>811715C8</td>
<td>80008000</td>
<td>7</td>
<td>XCTL</td>
<td>2</td>
</tr>
<tr>
<td>8116D1E0</td>
<td>80008000</td>
<td>8</td>
<td>LOAD</td>
<td>2</td>
</tr>
<tr>
<td>8116CED8</td>
<td>80008000</td>
<td>9</td>
<td>DELETE</td>
<td>2</td>
</tr>
<tr>
<td>811C5288</td>
<td>00008000</td>
<td>10</td>
<td>GETMAIN/FREEMAIN R</td>
<td>1</td>
</tr>
<tr>
<td>80FED356</td>
<td>80000000</td>
<td>251</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00D1E000</td>
<td>C0000000</td>
<td>252</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>80FED356</td>
<td>80000000</td>
<td>253</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>00FD3650</td>
<td>80000000</td>
<td>254</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>00D20050</td>
<td>C0000000</td>
<td>255</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>256 ITEMS SHOWN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: You may customize SVC names and descriptions using the AOSVCTBL utility. See the Configuration and Customization Guide for more information.
SHOW TRAPS

Description
SHOW TRAPS displays trap definitions. (Can be issued from the console.)

Syntax

```
subsys
(pattern,s)
CMD
DOM
TOD
WTO
xtype
```

`subsys` This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The `subsys` name is assigned by the SUBSYS startup parameter.

`(pattern,s)` A name pattern intended to match one or more trap names. You may use up to 32 patterns.

`CMD` Displays only definitions of CMD traps.

`DOM` Displays only definitions of DOM traps.

`TOD` Displays only definitions of TOD traps.

`WTO` Displays only definitions of WTO traps.

`xtype` Displays only definitions of OMEGAMON exception traps. The supported types are XO2, XOC, XOI, and XOM.

`DISABLE` Displays only definitions of disabled traps.

`ENABLE` Displays only definitions of enabled traps.

`PENDING` Displays only definitions of pending traps. These traps are waiting for AF TODSTART to be issued.
SHOW TRAPS

Comments
Any SHOW TRAPS display that exceeds the default of 1000 output messages will abend with a U722 abend code. You can specify the default at startup with the MSGLIM parameter. Also, you can reset the AF/OPERATOR system variable AOMSGLIM to an overriding value for just that match.

PENDING applies to traps that have been checkpointed and subsequently reloaded. For details of checkpointing, see the startup parameters CKPT, RELOAD, TODISYNC, and TRAPDFLT, in the AF/OPERATOR Configuration and Customization Guide, and the Checkpoint Dataset and AF TODSTART command sections of this manual.

Example
Following are some examples of this command as they would appear in a REXX exec.

"SHOW TRAPS" Displays all traps.
"SHOW TRAPS WTO" Displays only WTO traps.
"SHOW TRAPS ENABLE" Displays only enabled traps.
"SHOW TRAPS TOD ENABLE" Displays only enabled TOD traps.
"SHOW TRAPS(VTAM*)" Lists information about all traps with names that begin with VTAM.
"SHOW TRAPS XOI CMD" Displays only XOI traps or CMD traps.

A typical response to this command follows:

--- LIST OF ALL WTO CMD TRAPS ---
ENABLED WTOEXAMPLE WTO('TEST WTO*') COUNT(19)
-------- JOBTYPE(*) JOBNAMESYS(SYSG) JOBCLASS('**') SYSID(*) LINKID(*)
-------- DESC(*) ROUT(*) CON(*) ASID(*) USR(*) CONNAME() NEWSYS(SYSG)
-------- CONTINUE ALLOW LOG NOTEST RETAIN NOAOTRAP JOURNAL
-------- NEWDESC() NEWROUT()
-------- ADDDESC() ADDROUT()
-------- ACTION('EX TESTEXEC')
-------- MATCHLIM(0) ALTACT('')
-------- USERID(*) DAYS()
-------- DELDESC() DELROUT()
-------- ATTRIBUTES(COLOR(),HIGHLIGHT(),INTENSITY())
-------- JOURATTR(COLOR(GREEN),HIGHLIGHT(RVIDEO),INTENSITY(NORMAL))
-------- DOM ACTION('NOACT')
ENABLED CMDEXAMPLE CMD('MYCMD **') COUNT(5)
-------- JOBTYPE(IMS) JOBNAMESYS() JOBCLASS('**') SYSID(*) LINKID(*)
-------- CONTINUE ALLOW LOG NOTEST RETAIN NOAOTRAP
-------- ACTION('EX TESTEXEC')
-------- MATCHLIM(0) ALTACT('')
-------- USERID(*) DAYS()
2 ITEMS SHOWN
SHOW TRAPS TITLE FORMAT

Description
SHOW TRAPS TITLE FORMAT customizes a SHOW TRAPS command display. (Can be issued from the console.)

Syntax

```
SHOW TRAPS <subsys> TITLE('title/s') FORMAT('line/s')
```

The TITLE and FORMAT operands are usually placed at the end of a SHOW command.

**subsys**
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

**TITLE('title/s')**
Used with 1- to 8-title parameters in its subfield, specifies up to 8 title lines that will be displayed before any detail information lines are written. A title line can contain literal text and symbolic variables.

**FORMAT('line/s')**
Used with 1- to 8-line parameters in its subfield, specifies up to 8 lines that will be used to format the detail information. A format line can contain literal text and symbolic variables.

The symbolic variables that you can use in TITLE and FORMAT strings depend upon the type of SHOW command. They are listed at the end of this section.

Example
To display the names of all traps, their status, and type (WTO, CMD, or TOD), the command would appear as follows in a REXX exec:

```
"SHOW TRAPS TITLE('--- NAME STATUS AND TYPE ---')",
"FORMAT('TRAP: &&NAME &&STATUS &&TYPE')"
```

A typical response follows:

```
--- NAME STATUS AND TYPE ---
TRAP: CP ENABLED CMD
TRAP: NODDNAME ENABLED WTO
2 ITEMS SHOWN
```
SHOW TRAPS TITLE FORMAT

Another example:

```
“SHOW TRAP TITLE(‘EXAMPLE’)”,
“FORMAT(‘--- &&TYPE TRAP &&NAME ---’),
“‘...TEXT: &&TEXT’”,
“‘...ACTION: &&ACTION’)”
```

A typical response follows:

```
--- CMD TRAP CP ---
...TEXT: CP *
...ACTION: &AOTEXT
--- WTO TRAP NODDNAME ---
...TEXT: IEC130I*
...ACTION:
2 ITEMS SHOWN
```

Another example:

To display a count of traps without detail, specify this command with a null format:

```
“SHOW TRAPS FORMAT(‘’)”
```

A typical response follows:

```
3 ITEMS SHOWN
```

SHOW TRAPS Title Variables

The following variable names can be used in title strings for SHOW TRAPS commands:

- **&SHOWSTS**: ENABLED, DISABLED, or ALL
- **&SHOWTYPE**: CMD, TOD, WTO, XO2, XOC, XOI, XOM, or combination
SHOW TRAPS Format Variables

The following variable names can be used in FORMAT strings for SHOW TRAPS commands:

- **&ACTION**: Text of trap’s ACTION (1 to 255 characters)
- **&ASID**: Address space ID (1 to 3 digits)
- **&CONS**: List of console IDs separated by commas or *
- **&CTR**: Trap counter (1 to 8 digits)
- **&DESC**: Descriptor codes separated by commas or *
- **&JOBCLASS**: String of job classes or *
- **&JOBNAME**: Job name pattern (1 to 8 characters)
- **&JOBTYPE**: IMS, JOB, STC, TSU, combination, or * for CMD and WTO, and O2, OC, OI, OM, combination, or * for xtypes
- **&NAME**: Trap name (1 to 24 characters)
- **&OPTS**: SUPPRESS|ALLOW followed by WAIT|CONTINUE
- **&ROUT**: List of route codes separated by commas or *
- **&STATUS**: ENABLED or DISABLED
- **&SYSID**: System ID pattern (1 to 8 characters)
- **&TEXT**: Message text pattern (1 to 255 characters)
- **&TYPE**: CMD, TOD, WTO, XO2, XOC, XOI, or XOM
- **&USER**: User pattern (1 to 8 characters)
SHOW VARS

Description
SHOW VARS displays variables and their current values. (Can be issued from the console.)

Syntax

```
SHOW VARS
```

```
subsys
(pattern,s)
```

```
GLOBAL
SYSTEM
LOCAL
```

```
>` ASIS `<
```

subsys This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

(pattern,s) A name pattern intended to match one or more variable names. You can use up to 32 patterns.

LOCAL Displays only local variables.

GLOBAL Displays only global variables.

SYSTEM Displays only system variables.

ASIS Displays character variables and shows any leading or trailing blanks.

Comments

- AF/OPERATOR generates and uses many undocumented internal variables for purposes of housekeeping and message display. Some of these may appear in SHOW VARS listings.

- Any SHOW VARS display that exceeds the default of 1000 output messages will abend with a U722 abend code. You can specify the default at startup. Also, you can reset the system variable AOMSGLIM to an overriding value for just that match.

- SHOW VARS does not display REXX variables. Use the REXX SAY statement to display REXX variables.

- The variable type can be deduced from the display:
  - Quotes indicate a CHAR variable
  - No quotes indicate an INT or HEX variable.
Example
The following are examples of this command as it would appear in a REXX exec:

"SHOW VARS" Displays all variables.
"SHOW VARS SYSTEM" Displays only system variables.
"SHOW VARS GLOBAL" Displays only global variables.
"SHOW VARS(VTAM*)" Displays all variables that begin with the characters VTAM.
"SHOW VARS GLOBAL SYSTEM" Displays all global and system variables.

A typical response to this command follows:

```
SCOPE VARIABLE VALUE
GLOBAL AOTEXT = 'SHOW VARS'
GLOBAL AOJNAME= '$SYSM'
GLOBAL AOJTYPE= 'TSU'
GLOBAL AOTRAP = 'SHOW'
SYSTEM AOCPULIM= 5
SYSTEM AOMSLIM= 100
SYSTEM AOTASK = 'AO'
SYSTEM AOWATLIM= 300
8 ITEMS SHOWN
```
TPUT (TP)

Description
TPUT issues a message to a terminal that is currently logged onto TSO.

Syntax

```
TPUT 'message' <-
   ASID(number)  
   TSOID(tsoid) 
   USERID(userid)
```

'message' The text of the message.
ASID(number) Identifies the message destination by ASID number.
TSOID(tsoid) Identifies the message destination by TSO user ID.
USERID(userid) Identifies the message destination by TSO user ID.

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Message is sent to an active TSO session.</td>
</tr>
<tr>
<td>16</td>
<td>Invalid user ID.</td>
</tr>
<tr>
<td>20</td>
<td>User not logged on.</td>
</tr>
</tbody>
</table>

Comments

- Both the TSOID and USERID keywords are supported, although by definition tsoid and userid are equivalent terms.
- Use the MVS SEND command instead of the TPUT command if the user is not logged on when the message is issued.
Example
The following is an example of this command embedded in a REXX exec.

/* REXX */
status = GLBVGET('AOJTYPE')
status = GLBVGET('AOJNAME')
status = GLBVGET('AOSID')

IF aojtype = "TSU" THEN DO
   /* Caller was a TSO user */
   ADDRESS 'AFHOST',
      "TPUT 'This REXX exec was started by",
      "TSOID =" aojname,
      "on SYSID =" aoid " TSOID("||aojname||")"
END
ELSE DO
   /* Caller was not a TSO user */
   ADDRESS 'AFHOST',
      "WTO 'This REXX exec started by",
      "JOB =" aojname,
      "on SYSID =" aoid ""
END
RETURN
**TRAP (TR)**

**Description**
TRAP adds, changes, enables, disables, and deletes traps. (Can be issued from the console.)

This is a general description of the command. Particular forms of the command are described in subsequent sections.

**Note:** The abbreviation for TRAP (TR) is also a short form of the MVS TRACK command. If you are using TR for TRACK, you will need to disable or delete the AF/OPERATOR internal trap for TR.

**Syntax**

```
TEGRAP <subsys> ADD parameters
  CHANGE parameters
  DELETE parameters
  DISABLE parameters
  ENABLE parameters
```

**subsys**
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

**ADD**
Adds (defines) a new trap. You can ADD the following types of traps:
- CMD
- TOD
- WTO
- xtype

**CHANGE**
Changes one or more trap definitions. You can CHANGE the following types of traps:
- CMD
- WTO
- xtype

**ENABLE**
Activates one or more existing traps. You can ENABLE the following types of traps:
- CMD
- TOD
- WTO
- xtype
**DISABLE**

Deactivates one or more existing traps. Disabled traps are not deleted. You can DISABLE the following types of traps:

- CMD
- TOD
- WTO
- xtype

**DELETE**

Deletes one or more existing traps from strings. A deleted trap is no longer defined; a TRAP ADD command is necessary to redefine it. You can DELETE the following types of traps:

- CMD
- TOD
- WTO
- xtype

### Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Trap was successfully defined.</td>
</tr>
<tr>
<td>12</td>
<td>Required module is not linked.</td>
</tr>
<tr>
<td>16</td>
<td>Invalid syntax.</td>
</tr>
</tbody>
</table>

### Comments

There is no TRAP CHANGE command for TOD traps. You must delete the old trap and add a new one.
TRAP ADD CMD

Description
TRAP ADD CMD adds (defines) a trap that is triggered by operator commands. (Can be issued from the console.)

AF/OPERATOR monitors all commands issued by SVC 34, IMS or JES3 interfaces, or other user-defined sources.

Syntax

```
TRAP ADD(trapname) CMD('full-pattern') option/s
```

Options:

- **Triggering Conditions**
  - ASID(number)
  - CONSOLE(id/name,s)
  - DAYS(option/s)
    - Options: MON TUE WED THU FRI SAT SUN WEEKDAYS WEEKENDS
  - JOBCLASS('jobclasses')
  - JOBNAME(pattern)
  - JOBTYPEn(option/s)
    - Options: IMS JOB STC TSU
  - LINKID(value)
  - MATCHLIM(number)
  - SYSID(pattern)
  - USER(pattern)
  - USERID(id)

- **Immediate Action**
  - SUPPRESS|ALLOW

- **Extended Action**
  - ACTION('command')
  - ALTACT('command')

- **Trap Control**
  - AOTRAP|NOAOTRAP
  - ENABLE|DISABLE
  - JOURNAL|NOJOURNAL
  - LOG|NOLOG
  - RETAIN|NORETAIN
  - TEST|NOTEST
  - WAIT|CONTINUE

- **SLF Support**
  - JOURATTR(color,highlight,intensity)
  - JOURHELP(help panel name)

**subsys**
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The **subsys** name is assigned by the SUBSYS startup parameter.
TRAP ADD CMD

(trapname)  The name of the trap to be added.

CMD('full-pattern')

Specifies that the trap will be triggered by an operator command issued through SVC 34, the IMS or JES3 interface, or other user-defined sources, if the command text matches the pattern.

The pattern can contain pattern characters, pattern variables, and continuation signs. It can be up to 128 characters long. The maximum total length of the TRAP ADD command is 255 characters.

Triggering Conditions

ASID(number)

Only commands from jobs in the address space with ID number can trigger the trap. The variable number must be a decimal number.

If this operand is omitted, commands from jobs in any address spaces can trigger the trap. Using an asterisk (*) in place of n has the same effect as omitting the operand.

CONSOLE(id/name,s)

Only commands from consoles with one of the specified IDs and/or console names can trigger the trap. The console ID must be a decimal number or name. You can specify from 1 to 99 console IDs and names. The console name must be alphanumeric; the first character can be alphanumeric or a special character ($ or #). The console name option is only supported for MVS Version 4 and above. If a console name is assigned under a previous version of MVS, it will be ignored.

If this operand is omitted, commands from any console can trigger the trap.

DAY (option/s)

The trap is triggered only on the specified days. The options are:

- MON, TUE, WED, THU, FRI, SAT, SUN
- WEEKDAYS Monday through Friday
- WEEKENDS Saturday and Sunday

JOBCLASS('jobclasses')

Only commands from jobs in the specified job classes can trigger the trap. You can specify from 1 to 38 job classes. For example, JOBCLASS('ABCDE') specifies 5 job classes. Specify a job class by a text character as follows:

- alphanumeric (A to Z, 0 to 9)
- @ for TSO users
- $ for started tasks

If this operand is omitted, commands from any job class can trigger the trap. Using an asterisk (*) as the contents of the subfield has the same effect as omitting the operand.

JOBNAME(pattern)
Only commands from jobs whose job names match the pattern, or system tasks whose names match the pattern, or TSO users whose user IDs match the pattern can trigger the trap.

If this operand is omitted, commands from any jobs, system tasks, or TSO user IDs can trigger the trap. Using an asterisk (*) as the contents of the subfield has the same effect as omitting the operand.

**JOBTYPE(option/s)**

Only commands issued by a task type listed as an option can trigger the trap. The options are:
- JOB for batch jobs
- STC for system-started tasks
- TSU for TSO users
- IMS for IMS MTOs

To trap IMS messages, the SYSID parameter must be set to the name of the IMS control region in which the message originates. Therefore, you should examine your WTO traps and make sure their default SYSID is TRAPDFLT(ALLSYS). Otherwise, the traps will default to LINKID, which will not match the IMS control region name.

If this operand is omitted, commands from any type of task can trigger the trap. Using an asterisk (*) as the contents of the subfield has the same effect as omitting the operand.

**LINKID(value)**

Only commands issued through this LINKID from the corresponding AF/OPERATOR address space can trigger the trap. The default is an asterisk (*), which means that LINKID filtration is not performed.

**MATCHLIM(number)**

Use the MATCHLIM operand with the ALTACT operand. Every nth time the trap is triggered, AF/OPERATOR issues the command in the ALTACT subfield instead of the command in the ACTION subfield.

**SYSID(pattern)**
Only commands from tasks with SMF system IDs or JES3 system names that match a specified pattern can trigger the trap.

If you don’t specify the SYSID, the default is the name set in the TRAPDFT((LINKID) startup option. If neither SYSID nor LINKID is specified, the default is the host SMFID. See the AF/OPERATOR Configuration and Customization Guide for details.

Using an asterisk (*) allows messages from any SMF system to trigger the trap.

The SYSID used for adding a CMD trap is:
1. If the SYSID parameter is specified, then that value is used.
2. If the SYSID parameter is not specified, but the TRAPDFT(ALLSYS) startup parameter is specified, then SYSID(*) is used.
3. If the SYSID parameter is not specified, and the TRAPDFT(ALLSYS) startup parameter is not specified, then the SYSID specified in the startup parms is used. Note that if the SYSID was not specified at startup, then the MVS SYSNAME is used, not SMFID.

USERID(id) Allows a trap to filter matches based on the user who issued the command. This parameter works only with systems that have the RACF™ security product installed.

If you do not specify a value for USERID, all traps will be matched because the USERID default is an asterisk (*).

**Immediate Action**

**SUPPRESS** The trapped command is not issued. The issuing address space continues as if the command had been issued normally, and the command appears in SYSLOG.

*Note: Commands issued by the IMS MTO cannot be suppressed by AF/OPERATOR. IMS does not allow any commands issued from the MTO console to be suppressed.*

*Note: AF/OPERATOR cannot suppress the TSO LOGON command due to inconsistencies in the way MVS processes this command. If you define a command trap for the TSO LOGON command, AF/OPERATOR ignores the SUPPRESS option. Trap control and extended action are not affected. If the SUPPRESS option is the only action specified, then even though this is effectively a “no operation” trap, the trap count increases by one.*

**ALLOW** The trapped command is issued to MVS as normal.

**Extended Action**

**ACTION('command')**
When the trap is triggered, AF/OPERATOR issues the command specified in the subfield.

This is either an MVS operator command or an AF/OPERATOR command. It must be a single command no more than 128 characters in length. It may be an EXEC command that executes a REXX exec.

If the command is an MVS operator command and the UNDFTEXT(LOG) startup option has been specified, the OPER keyword must prefix the quoted command string.

**ALTACT('command')**

If the MATCHLIM parameter is used with the value n, every nth time the trap is triggered, AF/OPERATOR issues the command in the ALTACT subfield.

This is either an MVS operator command or an AF/OPERATOR command. It must be a single command no more than 128 characters in length. It may be an EXEC command that executes a REXX exec.

**Trap Control**

*Note:* Defaults for the ENABLE/DISABLE, LOG/NOLOG and RETAIN/NORETAIN trap control options are set in the TRAPDFLT startup option.

- **AOTRAP** The trap can be triggered by commands generated by AF/OPERATOR.
- **NOAOTRAP** The trap cannot be triggered by commands generated by AF/OPERATOR.
- **ENABLE** The trap is enabled (activated) as soon as it is defined.
- **DISABLE** The trap must be enabled by a TRAP ENABLE command.
- **JOURNAL** The matched data is journaled. Information about each match is recorded for display by the Subsystem Logging Facility (SLF).
- **NOJOURNAL** The matched information is not recorded.
- **LOG** A message is written to the AF/OPERATOR message log every time the trap is triggered.
- **NOLOG** No message is written to the AF/OPERATOR message log when the trap is triggered. Also, the results of the trap action are not logged.
- **RETAIN** Information about each match is recorded for display by AF/OPERATOR’s RETAIN*VIEW facility.
- **NORETAIN** Match information is not recorded for AF/OPERATOR’s RETAIN*VIEW facility.
- **TEST** When the trap is triggered, no direct or extended action is taken. The only commands that execute are the ones that execute in warn mode. However, AF/OPERATOR logs a record of the action that would have been taken.
- **NOTEST** When the trap is triggered, the programmed action is taken.
- **WAIT** The address space from which the command was issued is suspended while AF/OPERATOR is executing the trap action.
- **CONTINUE** The address space from which the command was issued is not suspended. The trap action occurs while the address space continues normal execution.

**Subsystem Logging Facility (SLF) Support**

- **JOURATTR** Specifies optional attributes of SLF data:
  - **color** Valid colors are blue, red, pink, green, turq, yellow and white.
highlight  Valid highlight values are none, blink, rvideo, and underline.
intensity  Valid intensity values are normal and high.

**EXAMPLE:** ATTR(COLOR(RED),HIGH(BLINK),INTEN(HIGH))

**JOURHELP**  Specifies the name of an optional help panel to be displayed when viewing the SLF data.

### Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Trap was successfully added or trap is already defined.</td>
</tr>
<tr>
<td>16</td>
<td>More than 99 console items are specified, no trap entries are free, invalid console name, or invalid syntax.</td>
</tr>
</tbody>
</table>

### Comments

- For information about using quotes and ampersands in the ACTION or ALTACT field of a TRAP command, see the *AF/OPERATOR User’s Guide*.

- When you enter reply commands in the short form, MVS/ESA 4.3.0 and above translates the short form into the long form of the command; that is, the verb **R** is appended at the beginning of the command and a comma is inserted between the reply ID and the reply text. For example, when you reply **yes** to REPL Y 99 by entering the short form **99yes**, MVS translates this into **R 99,yes**.

  The command will also appear in the long form in the SYSLOG. Therefore, you must code traps to match with the long form that appears in the SYSLOG, regardless of the form used when entering the reply at the console.

- The AOTRAP option allows you to trap commands issued by AF/OPERATOR itself. This can be useful. However, make sure AF/OPERATOR does not reissue the same command it traps, since this will cause a nonterminating loop, ending only when the match times out.

- The WAIT option can be used to allow AF/OPERATOR time to set traps for subsequent messages or commands that will be issued by the address space. If the trap is designed to protect the system against some damaging action, WAIT will stop the address space from doing any more of the damage before AF/OPERATOR can correct the problem.

**Note:** If you are using JES3, the trap will not detect the true originating system ID unless the startup parm **LINKID** is set to the name specified in the **MAINPROC** statement during JES3 initialization. For more information, refer to the *AF/OPERATOR Configuration and Customization Guide*.

<table>
<thead>
<tr>
<th>Important</th>
</tr>
</thead>
<tbody>
<tr>
<td>Never use WAIT in traps for commands from COMMTASK or other critical or multi-user address spaces such as ACF2, CICS, IMS, JES, or VTAM.</td>
</tr>
</tbody>
</table>

---

**AF/OPERATOR Commands** 369
Example
The following are examples of this command embedded in REXX execs.

The following trap disables the QUIESCE command from console 2.

```
“TRAP ADD(NOQ) CMD("QUIESCE") CONSOLE(2) SUPPRESS”,
“ACTION(’WTO “QUIESCE NOT ALLOWED FROM THIS CONSOLE’”,
“CONS(2)’)”
```

The trap defined by the following command issues a VTAM command to activate a node in the network.

```
“TRAP ADD(ACT) CMD(‘ACT *NODE’) SUPPRESS”,
“ACTION(’OPER “V NET,ACT,ID=&NODE” CONS(1)’)”
```

Note: The ENABLE keyword is often used in TRAP ADD commands. However, it is generally not shown on TRAP ADD examples in this manual.
TRAP ADD TOD

Description
TRAP ADD TOD adds (defines) a trap that will take action on a specific date at a specific time of day, or at evenly spaced time intervals. (Can be issued from the console.)

Note: There is no TRAP CHANGE command for TOD traps. You must delete the old trap and add a new one.

Syntax

```
TRAP ADD(trapname) TOD(timespec) [option/s]
```

Options:
- Triggering Conditions
  - DAYS(option/s)
    - Options: MON TUE WED THU FRI SAT SUN WEEKDAYS WEEKENDS
  - INTERVAL(interval)
  - MATCHLIM(number)
- Extended Action
  - ACTION('command')
  - ALTACT('command')
- Trap Control
  - CATCHUP(count) | SKIP
  - CKPT | NOCKPT
  - ENABLE | DISABLE
  - LOG | NOLOG
  - TEST | NOTEST

subsys
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

(trapname)
The name of the trap to be added.
TOD(timespec)

The point in time when the trap will be executed. Specify timespec as one of the following:

- A time in the form `hh:mm:ss` or as a number of seconds past midnight, for example:
  
  13:45:00 (1:45 p.m.)
  3600 (1:00 a.m.)

AF/OPERATOR supplies the current system date.

The maximum time allowed is 24:00:00. Note that 5:00 is interpreted as 00:05:00.

- A time followed by a date in the form `mm/dd/yy`, or `mm/dd/yyyy`; for example:
  
  12:34:56 01/23/2002
  12:34:56 01/23/02
  1234 01/23/02

- The current system time when the trap is added, indicated by an asterisk (*). You must specify the time of day or the current system time; there is no default time specification. If you do not specify a date, the system date is used.

Note: The time 00:00:00 is not a valid entry. To specify trap execution at midnight, enter 24:00:00 instead.

Triggering Conditions

DAYS(option/s)

The trap is triggered only on the specified days. The options are:

- MON, TUE, WED, THU, FRI, SAT, SUN
- WEEKDAYS Monday through Friday
- WEEKENDS Saturday and Sunday

INTERVAL(interval)

Specifies that the action will be performed periodically. TOD traps without the INTERVAL option are deleted after execution at the specified time.

The interval parameter is specified either as an integer `n` for `n`-second intervals or in the form `hh:mm:ss`. For example:

- INTERVAL(1) triggers the trap every second.
- INTERVAL(00:01:00) triggers the trap every minute.
- INTERVAL(01:00:00) triggers the trap every hour.

The maximum interval allowed is 24 hours, specified by INTERVAL(24:00:00).

MATCHLIM(n)

Use the MATCHLIM parameter with the ALTACT operand. Every `n`th time the trap is triggered, AF/OPERATOR issues the command in the ALTACT subfield instead of the command in the ACTION subfield.
Extended Action

ACTION('command')

When the trap is triggered, AF/OPERATOR issues the command specified in the subfield.

This is either an MVS operator command or an AF/OPERATOR command. It must be a single command no more than 128 characters in length. It may be an EXEC command that executes a REXX exec.

If the command is an MVS operator command and the UNDFTEXT(LOG) startup option has been specified, the OPER keyword must prefix the quoted command string.

ALTACT('command')

If the MATCHLIM operand is used with the value n, every nth time the trap is triggered, AF/OPERATOR issues the command in the ALTACT subfield.

This is either an MVS operator command or an AF/OPERATOR command. It must be a single command no more than 128 characters in length. It may be an EXEC command that executes a REXX exec.

Trap Control

Note: Defaults for the ENABLE/DISABLE and LOG/NOLOG trap control options are set in the TRAPDFLT startup option.

CATCHUP(count)

Causes AF/OPERATOR to execute the command specified in the ACTION option whenever the current system time and date are later than the next scheduled execution time and date of the trap. It also causes the trap to be checkpointed when added and each time it fires. TOD traps get warm started if the RELOAD parameter is set to TODS.

This process will happen repeatedly until either the TOD trap is scheduled for the future or the catchup limit (count) is reached, if one has been specified. The default (count) is 1.

Note: Catchup processing works only for starts done with RELOAD(TODS). Thus, if a trap is added for an earlier time, it will be rescheduled for a future time but will not go through catchup processing unless AF/OPERATOR is stopped and restarted.

CKPT

Causes this trap to be checkpointed. The default, CKPT or NOCKPT, is determined by what is specified on the TRAPDFLT(CKPT()) startup parameter.

Note: The CKPT startup parameter must be specified to activate the checkpointing facility.

NOCKPT

Causes this trap not to be checkpointed. If CATCHUP has been specified then the trap will always be checkpointed, regardless of specifying NOCKPT.

SKIP

Specifies that if (a) a trap is added after the specified time and (b) an interval is specified, the action is not taken until the next interval is reached.

ENABLE

The trap is enabled (activated) as soon as it is defined.

Note: The ENABLE keyword is often used in TRAP ADD commands. However, it is generally not shown on TRAP ADD examples in this manual.
**Return Codes**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Trap was successfully added or trap is already defined.</td>
</tr>
<tr>
<td>16</td>
<td>More than 99 console items are specified, no trap entries are free, invalid console name, or invalid syntax.</td>
</tr>
</tbody>
</table>

**Comments**

- For information about using quotes and ampersands in the ACTION or ALTACT field of a TRAP command, see the *AF/OPERATOR User's Guide*.

- If you specify **DAYS**, you must specify the interval. For example, if the trap is to fire once a day, you must code a 24-hour interval or, alternatively, if the trap is to fire every Saturday, you must specify a 24-hour interval.

- A TOD trap might require a catchup process if:
  - The system was inoperative during the time the trap was scheduled to be triggered.
  - AF/OPERATOR was inoperative during the time the trap was scheduled to be triggered.
  - The system was unavailable (that is, stopped) during the time the trap was scheduled to be triggered.
  - The CATCHUP option was specified. Then, processing is performed as many times as necessary until the action is scheduled for a future time or until the catchup limit (count) is reached.

- All TOD traps with either the CATCHUP or CATCHUP and INTERVAL options, or the CKPT option are checkpointed to ddname AOVCKPT for warm start processing. These TOD traps can automatically be added and enabled during AF/OPERATOR warm start initialization or they can be delayed. (See startup options in the *AF/OPERATOR Configuration and Customization Guide* and AF TODSTART in this manual.) Once the TOD trap has been scheduled, it will trigger immediately if its scheduled time and date have expired. Any such trap will be triggered immediately after initialization has completed if its scheduled time and date has passed.
TOD traps with short intervals and with the CATCHUP option, should have the COUNT option specified at some reasonable level. If this is not done, AF/OPERATOR might flood the console with messages and appear to have looped when it is simply performing the catchup activity that was specified.

If you specify a time/date combination that has already passed, AF/OPERATOR attempts to reschedule the event. If you do not specify an interval, the trap is deleted without action and AF/OPERATOR issues message AOP0179. This will happen if you specify a time prior to the time the trap is added and allow the date to default to the system date.

Example
The following is an example of this command embedded in a REXX exec.

```
"TRAP ADD(STARTJOB) TOD(22:00:00) DAYS(WEEKDAYS) ",

"INTERVAL(24:00:00) ACTION('EX STRTBKUP')"
```
**Description**

TRAP ADD WTO adds (defines) a trap that is triggered by write-to-operator (WTO) messages and write-to-operator-with-reply (WTOR) messages. (Can be issued from the console.)

AF/OPERATOR monitors its own messages, as well as all messages issued by SVC 35, IMS or JES3 interfaces, or other user-defined sources.

**Notes:**

1. Only the first line of a multi-line WTO is monitored.
2. Starting with MVS Version 4.2, the write-to-operator (WTO), SVC 35, is used to display MVS commands on the operator console. A WTO trap with selection criteria matching an operator command will be triggered by the echo of that command.
Syntax

```
TRAP ADD(trapname) WTO('full-pattern') subsys option/s
```

Options:

**Triggering Conditions**
- **ASID(number)**
- **CONSOLE(id/name,s)**
- **DAYS(option/s)**
  - Options:  MON TUE WED THU FRI  SAT SUN WEEKDAYS WEEKENDS
- **DESC(code,s,n:n,n-n)**
- **JOBCLASS('jobclasses')**
- **JOBNAME(pattern)**
- **JOBTYPE(option/s)**
  - Options:  IMS JOB STC TSU
- **LINKID(value)**
- **MATCHLIM(number)**
- **ROUT(code,s,n:n,n-n)**
- **SYSID(pattern)**
- **USER(pattern)**
- **USERID(id)**

**Immediate Action**
- **ADDDESC(code,s,n:n,n-n)**
- **ADDROUT(code,s,n:n,n-n)**
- **ATTRIBUTE(color,highlight,intensity)**
- **DELDesc(code,s,n:n,n-n)**
- **DELRout(code,s,n:n,n-n)**
- **HCOPY|NOHCOPY**
- **NEWCONS(id/name)**
- **NEWDESC(code,s,n:n,n-n)**
- **NEWROUT(code,s,n:n,n-n)**
- **SUPPRESS|ALLOW**

**Extended Action**
- **ACTION('command')**
- **ALTACT('command')**
- **DOMACT('command')**

**Trap Control**
- **AOTRAP|NOAOTRAP**
- **DOMLOG|NODOMLOG**
- **ENABLE|DISABLE**
- **JOURNAL|NOJOURNAL**
- **LOG|NLOG**
- **RETAIN|NORETAIN**
- **TEST|NOTEST**
- **WAIT|CONTINUE**

**MSMF Support**
- **NEWSYS(linkid)**

**SLF Support**
- **JOURATTR(color,highlight,intensity)**
- **JOURHELP(help panel name)**

**subsys**

This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The `subsys` name is assigned by the SUBSYS startup parameter.
TRAP ADD WTO

(trapname) The name of the trap to be added.

WTO('full-pattern')

Specifies that the trap will be triggered by a WTO/WTOR message, if the message matches the pattern.

The pattern can contain pattern characters, pattern variables, and continuation signs. It can be up to 128 characters long. The maximum total length of the TRAP ADD command is 255 characters.

Triggering Conditions

ASID(number)

Only messages from jobs in the address space with ID \( n \) can trigger the trap. The variable \( n \) must be a decimal number.

If this operand is omitted, messages from jobs in any address space can trigger the trap. Using an asterisk (*) in place of \( n \) has the same effect as omitting the operand.

CONSOLE(id/name,s)

Only WTOs directed to consoles with one of the specified IDs and/or console names can trigger the trap. The console ID must be a decimal number or name. You can specify 1 to 99 console IDs or names. The console name must be alphanumeric; the first character can be alphanumeric or a special character ($ or #). The console name option is only supported for MVS Version 4 and above. If a console name is assigned under a previous version of MVS, it will be ignored. AF/OPERATOR ignores the CONSOLE operand unless both of these conditions are met:

- The ROUTE operand is not used to route the message.
- The message is explicitly solicited by the specified console.

In general, ROUTE should be specified and CONSOLE should not.

DAYS(option/s)

The trap is triggered only on the specified days. The options are:

- MON, TUE, WED, THU, FRI, SAT, SUN
- WEEKDAYS Monday through Friday
- WEEKENDS Saturday and Sunday

DESC(code,s,n:n,n-n)

Only messages with a specified descriptor code can trigger the trap. You can specify 1 to 16 descriptor codes.

If this operand is omitted, messages with any descriptor code can trigger the trap. Using an asterisk (*) as the contents of the subfield has the same effect as omitting the operand.

JOBCLASS('jobclasses')
Only messages from jobs in the specified job classes can trigger the trap. You can specify 1 to 38 job classes. For example, JOBCLASS('ABCDE') specifies 5 job classes. Specify a job class by a text character as follows:

- alphanumeric (A to Z, 0 to 9)
- @ for TSO users
- $ for started tasks

If this operand is omitted, messages from any job class can trigger the trap. Using an asterisk (*) as the contents of the subfield has the same effect as omitting the operand.

**JOBNAME(pattern)**

Only messages from jobs whose job names match the pattern, system tasks whose names match the pattern, or TSO users whose user IDs match the pattern can trigger the trap.

If this operand is omitted, messages from any jobs, system tasks, or TSO user IDs can trigger the trap. Using an asterisk (*) as the contents of the subfield has the same effect as omitting the operand.

**JOBTYPE(option/s)**

Only messages issued by a task type listed as an option can trigger the trap. The options are:

- IMS for IMS MTOs
  To trap IMS messages, the SYSID parameter must be set to the name of the IMS control region in which the message originates. Therefore, you should examine your WTO traps and make sure their default SYSID is TRAPDFLT(ALLSYS). Otherwise, the traps will default to LINKID, which will not match the IMS control region name.
- JOB for batch jobs
- STC for system-started tasks
- TSU for TSO users

If this operand is omitted, messages from any type of task can trigger the trap. Using an asterisk (*) as the contents of the subfield has the same effect as omitting the operand.

**LINKID(value)**

Only WTOs issued through this LINKID from the corresponding AF/OPERATOR address space can trigger the trap. The default is an asterisk (*), which means that LINKID filtration is not performed.

**MATCHLIM(number)**

Use the MATCHLIM operand with the ALTACT operand. Every nth time the trap is triggered, AF/OPERATOR issues the command in the ALTACT subfield instead of the command in the ACTION subfield.

**ROUT(code,s,n:n,n-n)**
Only messages with a specified route code can trigger the trap. You can specify 1 to 128 route codes.

If this operand is omitted, messages with any route code can trigger the trap. Using an asterisk (*) as the contents of the subfield has the same effect as omitting the operand.

SYSID(pattern)

Only messages from tasks with SMF system IDs that match a specified pattern can trigger the trap.

If you do not specify the SYSID, the default is the name set in the TRAPDFLT(LINKID) startup option. If neither SYSID nor LINKID is specified, the default is the host SMFID. See the AF/OPERATOR Configuration and Customization Guide for details.

Using an asterisk (*) allows messages from any SMF system to trigger the trap.

The SYSID used for adding a WTO trap is:
1. If the SYSID parameter is specified, then that value is used.
2. If the SYSID parameter is not specified, but the TRAPDFLT(ALLSYS) startup parameter is specified, then SYSID(*) is used.
3. If the SYSID parameter is not specified, and the TRAPDFLT(ALLSYS) startup parameter is not specified, then the SYSID specified in the startup parms is used. Note that if the SYSID was not specified at startup, then the MVS SYSNAME is used, not SMFID.

USER(pattern)

Allows a trap to filter matches based on characters in the AOUSER field. If USER(pattern) matches the first 4 bytes of the AOUSER field, there is a match.

By default, the AOUSER field is the first 4 bytes of the TSO, STC, or JOB name. The AOUSER field can be overwritten when you generate a command or WTO using AOSIM.

If USER(pattern) is specified as an asterisk (*) or if USER(pattern) is not specified at all, then messages with any USER field trigger the trap.

USERID(id)

Allows a trap to filter matches based on the user who issued the command. This parameter works only with systems that have the RACF security product installed.

If you do not specify a value for USERID, all traps will be matched because the USERID default is an asterisk (*).
**Immediate Action**

**ADDDESC**(code,s,n:n,n-n)

The descriptor code or codes are appended to the existing code or codes. You can specify 1 to 16 descriptor codes.

**ADDRout**(code,s,n:n,n-n)

The specified route code or set of codes is appended to the existing code or set of codes. You can specify 1 to 128 new route codes.

**ATTRIBUTE**(color,highlight,intensity)

Modifies the attributes of the existing WTO. This parameter is optional.

**Color:**

- BLUE
- RED
- GREEN
- PINK
- YELLOW
- TURQ
- WHITE

**Highlight attributes:**

- NONE No highlight for message
- BLINK Blinking message
- UNDERLINE Underlined message
- RVIDEO Reverse video message

**Intensity attributes:**

- NORMAL Normal intensity
- HIGH High intensity

**EXAMPLE:** ATTR(COLOR(RED),HIGH(BLINK),INTEN(HIGH))

**DELDESC**(code,s,n:n,n-n)

Deletes the existing descriptor code or codes. You can specify 1 to 16 descriptor codes.

**DELRout**(code,s,n:n,n-n)

Deletes the existing route code or codes. You can specify 1 to 128 route codes.

**NEWCONS**(id/name)
Redirects the WTO to a new console location via a WTO trap. If you are using a version of MVS previous to 4.1, specify the new console ID as a decimal number. If you are running MVS Version 4.1 or higher, specify a 1 to 8 character console name.

NEWCONS does not work with JES3-owned consoles.

**HCOPY**  Sends output to the hardcopy log (SYSLOG).
**NOHCOPY**  Prevents output from going to the hardcopy log.

**NEWDESC**(code,s,n:n,n-n)

The existing descriptor code or codes is replaced by the specified code or codes. You can specify 1 to 16 codes.

**NEWROUT**(code,s,n:n,n-n)

The existing route code or codes is replaced by the specified code or codes. You can specify 1 to 128 codes.

**SUPPRESS**  The trapped message is not displayed. The issuing address space continues as if the message had been displayed normally, and the message appears in SYSLOG.

**ALLOW**  The trapped message is displayed as intended.

**Extended Action**

**ACTION**(‘command’)

When the trap is triggered, AF/OPERATOR issues the command specified in the subfield.

This is either an MVS operator command or an AF/OPERATOR command. It must be a single command no more than 128 characters in length. It may be an EXEC command that executes a REXX exec.

If the command is an MVS operator command and the UNDFTEXT(LOG) startup option has been specified, the OPER keyword must prefix the quoted command string.

**ALTACT**(‘command’)

If the MATCHLIM operand is used with the value n, every nth time the trap is triggered, AF/OPERATOR issues the command in the ALTACT subfield.

This is either an MVS operator command or an AF/OPERATOR command. It must be a single command no more than 128 characters in length. It may be an EXEC command that executes a REXX exec.

**DOMACT**(‘command’)

When an ADD TRAP WTO command is triggered, if DOMACT has been specified, then a DOM trap is set whose action is the action specified by the ‘command’ parameter.
**Trap Control**

*Note:* Defaults for ENABLE/DISABLE, LOG/NOLOG and RETAIN/NORETAIN trap control options are set in the TRAPDFLT startup option.

**AOTRAP**
The trap can be triggered by messages generated by AF/OPERATOR.

**NOAOTRAP**
The trap cannot be triggered by messages generated by AF/OPERATOR.

**DOMLOG**
A message is written to the AF/OPERATOR message log every time the DOM trap is triggered.

**NODOMLOG**
No message is written to the AF/OPERATOR message log when the trap is triggered. Also, the results of the trap action are not logged.

**ENABLE**
The trap is enabled (activated) as soon as it is defined.

*Note:* The ENABLE keyword is often used in TRAP ADD commands. However, it is generally not shown on TRAP ADD examples in this manual.

**DISABLE**
The trap is disabled (deactivated) when it is defined until it is enabled by a TRAP ENABLE command.

**JOURNAL**
The matched data is journaled on a trap. Information about each match is recorded for display by the Subsystem Logging Facility (SLF).

**NOJOURNAL**
The matched information is not recorded for SLF.

**LOG**
A message is written to the AF/OPERATOR message log every time the trap is triggered.

**NOLOG**
No message is written to the AF/OPERATOR message log when the trap is triggered. Also, the results of the trap action are not logged.

**RETAIN**
Information about each match is recorded for display by AF/OPERATOR’s RETAIN*VIEW facility.

**NORETAIN**
Match information is not recorded for AF/OPERATOR’s RETAIN*VIEW facility.

**TEST**
When the trap is triggered, no direct or extended action is taken. The only commands that execute are the ones that execute in warn mode. However, AF/OPERATOR logs a record of the action that would have been taken.

**NOTEST**
When the trap is triggered, the programmed action is taken.

**WAIT**
The address space from which the message was issued is suspended while AF/OPERATOR is executing the trap action.

**CONTINUE**
The address space from which the message was issued is not suspended. The trap action occurs while the address space continues normal execution.
**MSMF Support**

**NEWSYS(linkid)**

Sends the triggering message to the target system specified, using the Multi-System Management Facility.

*Note: If the trap also contains the SUPPRESS option, the message is not a candidate for MSMF and will not be sent to the target system; it will be suppressed on the local system.*

**Subsystem Logging Facility (SLF) Support**

**JOURATTR**

Specifies optional attributes of SLF data:

- **color** Valid colors are blue, red, pink, green, turq, yellow and white.
- **highlight** Valid highlight values are none, blink, rvideo, and underline.
- **intensity** Valid intensity values are normal and high.

**EXAMPLE:** `ATTR(COLOR(RED),HIGH(BLINK),INTEN(HIGH))`

**JOURHELP**

Specifies the name of an optional help panel to be displayed when viewing the SLF data.

**Return Codes**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Trap was successfully added or trap is already defined.</td>
</tr>
<tr>
<td>16</td>
<td>More than 99 console items are specified, no trap entries are free, invalid console name, or invalid syntax.</td>
</tr>
</tbody>
</table>

**Comments**

- For information about using quotes and ampersands in the ACTION or ALTACT field of a TRAP command, see the `AF/OPERATOR User’s Guide`.
- A WTO/WTOR triggers a WTO trap only if the trap is enabled at the time the WTO/WTOR is issued. This means, for example, that a WTO trap enabled at noon does not see any WTOs or WTORs that were issued in the morning.
- If a WTOR is sent with a route code of 9, and the reply is returned in a full reply form (such as `R nn,text`; not `nntext` or `nn, text`), the reply is suppressed. That is, when it is entered at the keyboard it is not echoed to the screen, and in SYSLOG and RKOLOGQM it is replaced by the word `SUPPRESSED`.

  Therefore, if you anticipate a response that is security sensitive or private (a password, for example), use ROUT(9) to prevent the response from going to any system activity logs.
The AOTRAP option allows you to trap messages issued by AF/OPERATOR itself. This can be useful. However, be careful that AF/OPERATOR does not reissue the same message it traps, since this will cause a nonterminating loop.

- The NEWSYS option enables you to consolidate automation in a network to a single system or console.

- The WAIT option can be used to allow time to set traps for subsequent messages or commands which will be issued by the address space. If the trap is designed to protect the system against some damaging action, WAIT will stop the address space from doing any more damage before AF/OPERATOR can correct the problem.

**Important**

Never use WAIT in traps for messages from COMMTASK or multi-user address spaces such as ACF2, CICS, IMS, JES, or VTAM.

**Note:** If you are using JES3, the trap will not detect the true originating system ID unless the startup parm LINKID is set to the name specified in the MAINPROC statement during JES3 initialization. For more information, refer to the AF/OPERATOR Configuration and Customization Guide.

**Example**

The following is an example of this command embedded in a REXX exec.

```
"TRAP ADD(NOJES) WTO('IEF450I') SUPPRESS"
```

The trap defined by this command suppresses IEF450I messages routed to any console.
TRAP ADD xtype

Description
TRAP ADD xtype adds (defines) a trap that is triggered by OMEGAMON exception messages. (Can be issued from the console.)

Syntax
```
subsys TRAP ADD(trapname) xtype('full-pattern') option/s
```

Options:
- **Triggering Conditions**
  - DAYS(option/s)
  - MATCHLIM(n)
  - SYSID(pattern)
- **Extended Action**
  - ACTION('command')
  - ALTACT('command')
- **Trap Control**
  - ENABLE|DISABLE
  - JOURNAL|NOJOURNAL
  - LOG|NOLOG
  - RETAIN|NORETAIN
  - TEST|NOTEST
  - SLF SUPPORT
  - JOURATTR(color,highlight,intensity)
  - JOURHELP(help panel name)

**subsys**
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

**trapname**
The name of the trap to be added.

**xtype('full-pattern')**
Specifies that the trap will be triggered by an exception issued by an OMEGAMON session of the indicated type, if the exception matches the pattern. The following types are supported:

- **XO2** OMEGAMON II for DB2
- **XOC** OMEGAMON for CICS
- **XOI** OMEGAMON for IMS
- **XOM** OMEGAMON for MVS

The pattern can contain pattern characters, pattern variables, and continuation signs. It can be up to 128 characters long. The maximum total length of the TRAP ADD command is 255 characters.
**TRAP ADD xtype**

**Triggering Conditions**

**DAYS(option/s)**

The trap is triggered only on the specified days. The options are:

- MON, TUE, WED, THU, FRI, SAT, SUN
- WEEKDAYS  Monday through Friday
- WEEKENDS  Saturday and Sunday

**MATCHLIM(n)**

Use the MATCHLIM parameter with the ALTACT parameter. Every \( n \)th time the trap is triggered, AF/OPERATOR issues the command in the ALTACT subfield instead of the command in the ACTION subfield.

**SYSID(pattern)**

Only commands from tasks with SMF system IDs that match a specified pattern can trigger the trap.

If you do not specify the SYSID, the default will be based upon what is set in the TRAPDFLT startup option. Please see the AF/OPERATOR Configuration and Customization Guide for details.

Using an asterisk (*) allows messages from any SMF system to trigger the trap.

**Extended Action**

**ACTION('command')**

When the trap is triggered, AF/OPERATOR issues the command specified in the subfield.

It can be either an MVS operator command or an AF/OPERATOR command. It must be a single command no more than 128 characters in length. It may be an EXEC command that executes a REXX exec.

If the command is an MVS operator command and the UNDFTEXT(LOG) startup option has been specified, the OPER keyword must prefix the quoted command string.

**ALTACT('command')**

If the MATCHLIM operand is used with the value \( n \), every \( n \)th time the trap is triggered, AF/OPERATOR issues the command in the ALTACT subfield.

This is either an MVS operator command or an AF/OPERATOR command. It must be a single command no more than 128 characters in length. It may be an EXEC command that executes a REXX exec.

**Trap Control**

**Note:** Defaults for the ENABLE/DISABLE, LOG/NOLOG, and RETAIN/NORETAIN trap control options are set in the TRAPDFLT startup option.

**ENABLE**

The trap is enabled (activated) as soon as it is defined.

**Note:** The ENABLE keyword is often used in TRAP ADD commands. However, it is generally not shown on TRAP ADD examples in this manual.

**DISABLE**

The trap is disabled (deactivated) when it is defined until it is enabled by a TRAP ENABLE command.
TRAP ADD xtype

**JOURNAL**
The matched data is journaled. Information about each match is recorded for display by the Subsystem Logging Facility (SLF).

**NOJOURNAL**
The matched information is not recorded.

**LOG**
A message is written to the AF/OPERATOR message log every time the trap is triggered.

**NOLOG**
No message is written to the AF/OPERATOR message log when the trap is triggered. Also, the results of the trap action are not logged.

**RETAI N**
Information about each match is recorded for display by the AF/OPERATOR RETAIN*VIEW facility.

**NORETAIN**
Match information is not recorded for the AF/OPERATOR RETAIN*VIEW facility.

**TEST**
When the trap is triggered, no direct or extended action is taken. The only commands that execute are the ones that execute in warn mode. However, AF/OPERATOR logs a record of the action that would have been taken.

**NOTEST**
When the trap is triggered, the programmed action is taken.

**Subsystem Logging Facility (SLF) Support**

**JOURATTR**
Specifies optional attributes of SLF data:

- **color**
  Valid colors are blue, red, pink, green, turq, yellow and white.

- **highlight**
  Valid highlight values are none, blink, rvideo, and underline.

- **intensity**
  Valid intensity values are normal and high.

**EXAMPLE:** `ATTR(COLOR(RED),HIGH(BLINK),INTEN(HIGH))`

**JOURHELP**
Specifies the name of an optional help panel to be displayed when viewing the SLF data.

**Return Codes**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Trap was successfully added or trap is already defined.</td>
</tr>
<tr>
<td>16</td>
<td>More than 99 console items are specified, no trap entries are free, invalid console name, or invalid syntax.</td>
</tr>
</tbody>
</table>

**Comments**
- For information about using quotes and ampersands in the ACTION or ALTACT field of a TRAP command, see the *AF/OPERATOR User’s Guide*.
- An xtype trap will not function for an OMEGAMON session unless the LOGON command that established the session specified a time interval with the INTERVAL parameter.

**Example**
For exception trap examples, see the AF/OPERATOR User's Guide.
TRAP CHANGE CMD

Description
TRAP CHANGE CMD modifies a trap added (defined) with the TRAP ADD CMD command. (Can be issued from the console.)

Syntax

```
subsys  TRAP CHANGE(pattern,s)  CMD  option/s  <
```

Options:
- Trap Selection
  - ENABLE
  - DISABLE
- Triggering Conditions
  - ASID(n)
  - CONSOLE(id/name,s)
  - DAYS(option/s)
    - Options: MON TUE WED THU FRI SAT SUN WEEKDAYS WEEKENDS
  - JOBCLASS(‘jobclasses’)
  - JOBNAME(pattern)
  - JOBTYPE(option/s)
    - Options: IMS JOB STC TSU
  - LINKID(value)
  - MATCHLIM(n)
  - SYSID(pattern)
  - USER(pattern)
  - USERID(id)
- Immediate Action
  - SUPPRESS|ALLOW
- Extended Action
  - ACTION(‘command’)
  - ALTACT(‘command’)
- Trap Control
  - AOTRAP | NOAOTRAP
  - JOURNAL | NOJOURNAL
  - LOG | NOLOG
  - RETAIN | NORETAIN
  - TEST | NOTEST
  - WAIT | CONTINUE
- SLF Support
  - JOURATTR(color,highlight,intensity)
  - JOURHELP(help panel name)

Note: The parameters have the same meanings as in the TRAP ADD CMD command except for ENABLE and DISABLE, which are used as selection criteria.

subsys
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.
(pattern,s) A name pattern intended to match one or more trap names. You can use up to 32 patterns.

CMD('full-pattern')
Specifies that the trap will be triggered by an operator command issued through SVC 34, the IMS or JES3 interface, or other user-defined sources, if the command text matches the pattern.

The pattern can contain pattern characters, pattern variables, and continuation signs. It can be up to 128 characters long and is optional unless you are changing patterns. The maximum total length of the TRAP CHANGE command is 255 characters.

**Trap Selection**

**ENABLE** Only enabled traps are changed.

**DISABLE** Only disabled traps are changed.

*Note:* If neither ENABLE nor DISABLE is coded, all CMD traps whose names match the trap name pattern will be changed.

**Triggering Conditions**

**ASID(n)** Only commands from jobs in the address space with ID n can trigger the trap. The variable n must be a decimal number.

**CONSOLE(id/name,s)**
Only WTOs directed to consoles with one of the specified IDs and/or console names can trigger the trap. The console ID must be a decimal number or name. You can specify from 1 to 99 console IDs or names. The console name must be alphanumeric; the first character can be alphanumeric or a special character ($ or #). The console name option is only supported for MVS Version 4 and above. If a console name is assigned under a previous version of MVS, it will be ignored. If this operand is omitted, commands from any console can trigger the trap.

**DAYS(option/s)**
The trap is triggered only on the specified days. The options are:

- MON, TUE, WED, THU, FRI, SAT, SUN
- WEEKDAYS Monday through Friday
- WEEKENDS Saturday and Sunday

**JOBCLASS('jobclasses')**
Only commands from jobs in the specified job classes can trigger the trap. You can specify from 1 to 38 job classes. For example, JOBCLASS('ABCDE') specifies 5 job classes. Specify a job class by a text character as follows:

- alphanumeric (A to Z, 0 to 9)
- @ for TSO users
- $ for started tasks

If this operand is omitted, commands from any job class can trigger the trap. Using an asterisk (*) as the contents of the subfield has the same effect as omitting the operand.
**JOBNAME(pattern)**

Only commands from jobs whose job names match the pattern, system tasks whose names match the pattern, or TSO users whose user IDs match the pattern can trigger the trap.

If this operand is omitted, commands from any jobs, system tasks, or TSO user IDs can trigger the trap. Using an asterisk (*) as the contents of the subfield has the same effect as omitting the operand.

**JOBTYPE(option/s)**

Only commands issued by the following task types can trigger the trap:
- IMS for IMS MTOs
- JOB for batch jobs
- STC for system-started tasks
- TSU for TSO users

If this operand is omitted, commands from any type of task can trigger the trap. Using an asterisk (*) as the contents of the subfield has the same effect as omitting the operand.

**LINKID(value)**

Only commands issued through this LINKID from the corresponding AF/OPERATOR address space can trigger the trap. The default is an asterisk (*), which means that LINKID filtration is not performed.

**MATCHLIM(n)**

Use the MATCHLIM operand with the ALTACT operand. Every \( n \)th time the trap is triggered, AF/OPERATOR issues the command in the ALTACT subfield instead of the command in the ACTION subfield.

**SYSID(pattern)**

Only commands from tasks with SMF system IDs that match a specified pattern can trigger the trap.

If you don’t specify the SYSID, the default will be based upon what is set in the TRAPDFLT startup option. Please see the AF/OPERATOR Configuration and Customization Guide for details.

Using an asterisk (*) as the contents of the subfield has the same effect as omitting the operand.

**USER(pattern)**

Allows a trap to filter matches based on characters in the AOUSER field. If USER(pattern) matches the first 4 bytes of the AOUSER field, there is a match.

By default, the AOUSER field is the first 4 bytes of the TSO, STC, or JOB name. The AOUSER field can be overwritten when you generate a command or WTO using AOSIM.

If USER(pattern) is specified as an asterisk (*) or if USER(pattern) is not specified at all, then messages with any USER field trigger the trap.
USERID(id)

Allows a trap to filter matches based on the user who issued the command. This parameter works only with systems that have the RACF security product installed. If you do not specify a value for USERID, all traps will be matched because the USERID default is an asterisk (*).

Immediate Action

SUPPRESS

The trapped command is not issued. The issuing address space continues as if the command had been issued normally, and the command appears in SYSLOG.

Note: Commands issued by the IMS MTO cannot be suppressed by AF/OPERATOR. IMS does not allow any commands issued from the MTO console to be suppressed.

ALLOW

The trapped command is issued as intended.

Extended Action

ACTION('command')

When the trap is triggered, AF/OPERATOR issues the command specified in the subfield.

This is either an MVS operator command or an AF/OPERATOR command. It must be a single command no more than 128 characters in length. It may be an EXEC command that executes a REXX exec.

If the command is an MVS operator command and the UNDFTEXT(LOG) startup option has been specified, the OPER keyword must prefix the quoted command string.

ALTACT('command')

If the MATCHLIM operand is used with the value n, every nth time the trap is triggered, AF/OPERATOR issues the command in the ALTACT subfield.

This is either an MVS operator command or an AF/OPERATOR command. It must be a single command no more than 128 characters in length. It may be an EXEC command that executes a REXX exec.

Trap Control

Note: Defaults for LOG/NOLOG and RETAIN/NORETAIN trap control options are set in the TRAPDFLT startup option.

AOTRAP

The trap can be triggered by messages generated by AF/OPERATOR.

NOAOTRAP

The trap cannot be triggered by messages generated by AF/OPERATOR.

JOURNAL

The matched data is journaled. Information about each match is recorded for display by the Subsystem Logging Facility (SLF).

NOJOURNAL

The matched information is not recorded.
LOG  
A message is written to the AF/OPERATOR message log every time the trap is triggered.

NOLOG  
No message is written to the AF/OPERATOR message log when the trap is triggered. Also, the results of the trap action are not logged.

RETAIN  
Information about each match is recorded for display by the AF/OPERATOR RETAIN*VIEW facility.

NORETAIN  
Match information is not recorded for the AF/OPERATOR RETAIN*VIEW facility.

TEST  
When the trap is triggered, no direct or extended action is taken. The only commands that execute are the ones that execute in warn mode. However, AF/OPERATOR logs a record of the action that would have been taken.

NOTEST  
When the trap is triggered, the programmed action is taken.

WAIT  
The address space from which the message was issued is suspended while AF/OPERATOR is executing the trap action.

CONTINUE  
The address space from which the message was issued is not suspended. The trap action occurs while the address space continues normal execution.

Subsystem Logging Facility (SLF) Support

JOURATTR  
Specifies optional attributes of SLF data:

- **color**  
  Valid colors are blue, red, pink, green, turq, yellow and white.

- **highlight**  
  Valid highlight values are none, blink, rvideo, and underline.

- **intensity**  
  Valid intensity values are normal and high.

  **EXAMPLE:** ATTR(COLOR(RED),HIGH(BLINK),INTEN(HIGH))

JOURHELP  
Specifies the name of an optional help panel to be displayed when viewing the SLF data.

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Trap was successfully changed.</td>
</tr>
<tr>
<td>16</td>
<td>Invalid syntax.</td>
</tr>
</tbody>
</table>
Comments

- To change the definition of a trap, specify new options to replace old ones. You need only specify options to be changed, except that you must code CMD even if the CMD text pattern is not to be changed.
- For information about using quotes and ampersands in the ACTION or ALTACT field of a TRAP command, see the AF/OPERATOR User’s Guide.

Example

The following is an example of this command embedded in a REXX exec.

This TRAP CHANGE command changes the definition of the trap named OPERHELP. It changes the ID of the CONSOLE from which the command must be issued for the trap to be triggered.

```
“TRAP CHANGE(OPERHELP) CMD CONSOLE(2)”
```
TRAP CHANGE WTO

Description
TRAP CHANGE WTO modifies a trap added (defined) with the TRAP ADD WTO command. (Can be issued from the console.)

Syntax

```plaintext
TRAP CHANGE(pattern,s) WTO(option/s) WTO('full-pattern')
```

Options:
- Trap Selection
  - ENABLE|DISABLE
- Triggering Conditions
  - ASID(number)
  - CONSOLE(id/name,s)
  - DAYS(option/s)
    - Options: MON TUE WED THU FRI SAT SUN WEEKDAYS WEEKENDS
  - DESC(code,s,n:n,n-n)
  - JOBCLASS('jobclasses')
  - JOBNAME(pattern)
  - JOBTYPE(option/s)
    - Options: IMS JOB STC TSU
  - LINKID(value)
  - MATCHLIM(number)
  - ROUT(code,s,n:n,n-n)
  - SYSID(pattern)
  - USER(pattern)
  - USERID(id)
- Immediate Action
  - ATTRIBUTE(color,highlight,intensity)
  - DELDESC(code,s,n:n,n-n)
  - DELROUT(code,s,n:n,n-n)
  - NEWCONS(id/name)
  - HCOPY|NOHCOPY
  - NEWDESC(code,s,n:n,n-n)
  - NEWROUT(code,s,n:n,n-n)
  - SUPPRESS|ALLOW
- Extended Action
  - ACTION('command')
  - ALTACT('command')
- Trap Control
  - AOTRAP|NOAOTRAP
  - JOURNAL|NOJOURNAL
  - LOG|NOLOG
  - RETAIN|NORETAIN
  - TEST|NOTEST
  - WAIT|CONTINUE
- MSMF Support
  - NEWSYS(linkid)
- SLF Support
  - JOURATTR(color,highlight,intensity)
  - JOURHELP(help panel name)
The parameters have the same meanings as in the TRAP ADD WTO command except for ENABLE and DISABLE, which are used as selection criteria.

**subsys**  
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

**(pattern,s)**  
A name pattern intended to match one or more trap names. You can use up to 32 patterns.

**WTO(**'full-pattern')**  
Specifies that the trap will be triggered by a WTO/WTOR message, if the message matches the pattern.

The pattern can contain pattern characters, pattern variables, and continuation signs. It can be up to 128 characters long and is optional unless you are changing patterns. The maximum total length of the TRAP CHANGE command is 255 characters.

**Trap Selection**

**ENABLE**  
Only enabled traps are changed.

**DISABLE**  
Only disabled traps are changed.

**Note:** If neither ENABLE or DISABLE is coded, all WTO traps whose names match the trap name pattern will be changed.

**Triggering Conditions**

**ASID(number)**

Only messages from jobs in the address space with ID number can trigger the trap. The variable number must be a decimal number.

**CONSOLE(id/name,s)**

Only WTOs directed to consoles with one of the specified IDs and/or console names can trigger the trap. The console ID must be a decimal number or name. You can specify from 1 to 99 console IDs or names. The console name must be alphanumeric; the first character can be alphanumeric or a special character ($ or #). The console name option is only supported for MVS Version 4 and above. If a console name is assigned under a previous version of MVS, it will be ignored. If this operand is omitted, messages to any console can trigger the trap.

**DAYS(option/s)**

The trap is triggered only on the specified days. The options are:
- **MON, TUE, WED, THU, FRI, SAT, SUN**
- **WEEKDAYS** Monday through Friday
- **WEEKENDS** Saturday and Sunday
Only messages with a specified descriptor code can trigger the trap. You can specify up to 16 descriptor codes.

**JOBCLASS('jobclasses')**

Only messages from jobs in the specified job classes can trigger the trap. You can specify up to 38 job classes. For example, JOBCLASS('ABCDE') specifies 5 job classes. Specify a job class by a text character as follows:

- alphanumeric (A to Z and 0 to 9)
- @ for TSO users
- $ for started tasks

If this operand is omitted, messages from any job class can trigger the trap. Using an asterisk (*) as the contents of the subfield has the same effect as omitting the operand.

**JOBNAME(pattern)**

Only messages from jobs whose job names match the pattern, system tasks whose names match the pattern, or TSO users whose user IDs match the pattern can trigger the trap.

If this operand is omitted, messages from any jobs, system tasks, or TSO user IDs can trigger the trap. Using an asterisk (*) as the contents of the subfield has the same effect as omitting the operand.

**JOBTYPE(option/s)**

Only messages issued by the following task types can trigger the trap:

- IMS for IMS MTOs
- JOB for batch jobs
- STC for system started tasks
- TSU for TSO users

If this operand is omitted, messages from any type of task can trigger the trap. Using an asterisk (*) as the contents of the subfield has the same effect as omitting the operand.

**LINKID(value)**

Only WTOs issued through this LINKID from the corresponding AF/OPERATOR address space can trigger the trap. The default is an asterisk (*), which means that LINKID filtration is not performed.

**MATCHLIM(number)**

Use the MATCHLIM operand with the ALTACT operand. Every nth time the trap is triggered, AF/OPERATOR issues the command in the ALTACT subfield instead of the command in the ACTION subfield.

**ROUT(code,s:n:n:n-n)**

Only messages with a specified route code can trigger the trap. You can specify up to 128 routing codes.

**SYSID(pattern)**
Only commands from tasks with SMF system IDs that match a specified pattern can trigger the trap.

If you don’t specify the SYSID, the default will be based upon what is set in the TRAPDFLT startup option. Please see the *AF/OPERATOR Configuration and Customization Guide* for details.

Using an asterisk (*) as the contents of the subfield has the same effect as omitting the operand.

**USER(pattern)**

Allows a trap to filter matches based on characters in the AOUSER field. If USER(pattern) matches the first 4 bytes of the AOUSER field, there is a match. By default, the AOUSER field is the first 4 bytes of the TSO, STC, or JOB name. The AOUSER field can be overwritten when you generate a command or WTO using AOSIM.

If USER(pattern) is specified as an asterisk (*) or if USER(pattern) is not specified at all, then messages with any USER field trigger the trap.

**USERID(id)**

Allows a trap to filter matches based on the user who issued the command. This parameter works only with systems that have the RACF security product installed.

If you do not specify a value for USERID, all traps will be matched because the USERID default is an asterisk (*).

**Immediate Action**

**ATTRIBUTE(color,highlight,intensity)**

This will modify the attributes of the existing WTO. This parameter is optional.

**Color:**

- BLUE
- RED
- GREEN
- PINK
- YELLOW
- TURQ
- WHITE

**Highlight attributes:**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>No highlight for message.</td>
</tr>
<tr>
<td>BLINK</td>
<td>Blinking message.</td>
</tr>
<tr>
<td>UNDERLINE</td>
<td>Underlined message.</td>
</tr>
<tr>
<td>RVIDEO</td>
<td>Reverse video message.</td>
</tr>
</tbody>
</table>
Intensity attributes:

<table>
<thead>
<tr>
<th>Normal</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal intensity.</td>
<td>High intensity.</td>
</tr>
</tbody>
</table>

**EXAMPLE:** `ATTR(COLOR(RED),HIGH(BLINK),INTEN(HIGH))`

**DELDESC(code,s,n:n,n-n)**
Deletes the existing descriptor code or codes. You can specify 1 to 16 descriptor codes.

**DELROUT(code,s,n:n,n-n)**
Deletes the existing route code or codes. You can specify 1 to 128 route codes.

**NEWCONS(id/name)**
Redirects the WTO to a new console location via a WTO trap. If you are using a version of MVS previous to 4.1, specify the new console ID as a decimal number. If you are running MVS Version 4.1 or higher, specify a 1 to 8 character console name.

NEWCONS does not work with JES3-owned consoles.

**HCOPY**
Sends output to the hardcopy log (SYSLOG).

**NOHCOPY**
Prevents output from going to the hardcopy log.

**NEWDESC(code,s,n:n,n-n)**
The existing descriptor code or codes is replaced by the specified code or codes. You can specify up to 16 codes.

**NEWROUT(code,s,n:n,n-n)**
The existing route code or set of route codes is replaced by the specified code or codes. You can specify up to 128 codes.

**SUPPRESS**
The trapped message is not displayed. The issuing address space continues as if the message had been displayed normally, and the message appears in SYSLOG.

**ALLOW**
The trapped message is displayed as intended.

**Extended Action**

**ACTION('command')**
When the trap is triggered, AF/OPERATOR issues the command specified in the subfield. This is an MVS operator command or an AF/OPERATOR command. It must be a single command no more than 128 characters in length. It may be an EXEC command that executes a REXX exec.

If the command is an MVS operator command and the UNDFTEXT(LOG) startup option has been specified, the OPER keyword must prefix the quoted command string.
ALTACT('command')

If the MATCHLIM operand is used with the value \( n \), every \( n \)th time the trap is triggered, AF/OPERATOR issues the command in the ALTACT subfield.

This is an MVS operator command or an AF/OPERATOR command. It must be a single command no more than 128 characters in length. It may be an EXEC command that executes a REXX exec.

**Note:** If the trap is disabled and then enabled, or deleted and added again, it issues the ACTION command the next \( n \) times it is triggered.

**Trap Control**

**Note:** Defaults for LOG/NOLOG and RETAIN/NORETAIN trap control options are set in the TRAPDFLT startup option.

AOTRAP  The trap can be triggered by messages generated by AF/OPERATOR.

NOAOTRAP  The trap cannot be triggered by messages generated by AF/OPERATOR.

JOURNAL  The matched data is journaled. Information about each match is recorded for display by the Subsystem Logging Facility (SLF).

NOJOURNAL  The matched information is not recorded.

LOG  A message is written to the AF/OPERATOR message log every time the trap is triggered.

NOLOG  No message is written to the AF/OPERATOR message log when the trap is triggered. Also, the results of the trap action are not logged.

RETAIN  Information about each match is recorded for display by the AF/OPERATOR RETAIN*VIEW facility.

NORETAIN  Match information is not recorded for the AF/OPERATOR RETAIN*VIEW facility.

TEST  When the trap is triggered, no direct or extended action is taken. The only commands that execute are the ones that execute in warn mode. However, AF/OPERATOR logs a record of the action that would have been taken.

NOTEST  When the trap is triggered, the programmed action is taken.

WAIT  The address space from which the message was issued is suspended while AF/OPERATOR is executing the trap action.

CONTINUE  The address space from which the message was issued is not suspended. The trap action occurs while the address space continues normal execution.

**MSMF Support**

NEWSYS(linkid)

Transmits the triggering message to the system specified, using the Multi-System Management Facility.

**Note:** If the trap contains the SUPPRESS option, the message is not transmitted by MSMF.
Subsystem Logging Facility (SLF) Support

JOURATTR  Specifies optional attributes of SLF data:

  color       Valid colors are blue, red, pink, green, turq, yellow and white.
  highlight   Valid highlight values are none, blink, rvideo, and underline.
  intensity   Valid intensity values are normal and high.

  EXAMPLE:  ATTR(COLOR(RED),HIGH(BLINK),INTEN(HIGH))

JOURHELP    Specifies the name of an optional help panel to be displayed when viewing the SLF data.

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Trap was successfully changed.</td>
</tr>
<tr>
<td>16</td>
<td>Invalid syntax.</td>
</tr>
</tbody>
</table>

Comments

- To change the definition of a trap, specify new options to replace old ones. You need only specify options to be changed, except that you must code WTO even if the WTO text pattern is not to be changed.

- If a WTOR is sent with a route code of 9, and the reply is returned in a full reply form (such as R nn,text; not nntext or nn, text), the reply is suppressed. When it is entered at the keyboard it is not echoed to the screen, and in SYSLOG and RKOGLOGM, it is replaced by the word SUPPRESSED.

Therefore, if you anticipate a response that is security sensitive or private (a password, for example), use ROUT(9) to prevent the response from going to any system activity logs.

- For information about using quotes and ampersands in the ACTION or ALTACT field of a TRAP command, see the AF/OPERATOR User’s Guide.

Example

The following is an example of this command embedded in a REXX exec.

The following TRAP CHANGE command changes the definition of the trap named NOXYZMSG. It modifies the JOBTYPE from which the message must be issued for the trap to be triggered.

“TRAP CHANGE(NOXYZMSG) WTO JOBTYPE(STC)”
TRAP CHANGE xtype

Description
TRAP CHANGE xtype modifies a trap added (defined) with the TRAP ADD xtype command.
(Can be issued from the console.)

Syntax

```plaintext
TRAP CHANGE(pattern,s) xtype option/s -->
```

Options:

- **subsys**: This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

- **(pattern,s)**: A name pattern intended to match one or more trap names. You can use up to 32 patterns.

- **xtype('full-pattern')**: Specifies that the trap will be triggered by an exception issued by an OMEGAMON session of the indicated type, if the exception matches the pattern. The following types are supported:
  - **XO2**: OMEGAMON II for DB2
  - **XOC**: OMEGAMON for CICS

Note: The parameters have the same meanings as in the TRAP ADD xtype command except for ENABLE and DISABLE, which are used as selection criteria.
TRAP CHANGE xtype

Note:
If neither ENABLE nor DISABLE is coded, all xtype traps whose names match the trap name pattern will be changed.

Trap Selection
ENABLE
Only enabled traps are changed.
DISABLE
Only disabled traps are changed.

Triggering Conditions
DAYS(option/s)
The trap is triggered only on the specified days. The options are:
- MON, TUE, WED, THU, FRI, SAT, SUN
- WEEKDAYS Monday through Friday
- WEEKENDS Saturday and Sunday

MATCHLIM(n)
Use the MATCHLIM operand with the ALTACT operand. Every nth time the trap is triggered, AF/OPERATOR issues the command in the ALTACT subfield instead of the command in the ACTION subfield.

SYSID(pattern)
Only commands from tasks with SMF system IDs that match a specified pattern can trigger the trap.
If you don’t specify the SYSID, the default will be based upon what is set in the TRAPDFLT startup option. Please see the AF/OPERATOR Configuration and Customization Guide for details.
Using an asterisk (*) as the contents of the subfield has the same effect as omitting the operand.

Extended Action
ACTION('command')
When the trap is triggered, AF/OPERATOR issues the command specified in the subfield.
This is either an MVS operator command or an AF/OPERATOR command. It must be a single command no more than 128 characters in length. It may be an EXEC command that executes a REXX exec.
If the command is an MVS operator command, and the UNDFTEXT(LOG) startup option has been specified, the OPER keyword must prefix the quoted command string.
ALTACT('command')

If the MATCHLIM operand is used with the value $n$, every $n$th time the trap is triggered, AF/OPERATOR issues the command in the ALTACT subfield. This is either an MVS operator command or an AF/OPERATOR command. It must be a single command no more than 128 characters in length. It may be an EXEC command that executes a REXX exec.

**Note:** If the trap is disabled and then enabled, or deleted and added again, it issues the ACTION command the next $n$ times it is triggered.

**Trap Control**

**Note:** Defaults for LOG/NOLOG and RETAIN/NORETAIN trap control options are set in the TRAPDFLT startup option.

**JOURNAL**
The matched data is journaled. Information about each match is recorded for display by the Subsystem Logging Facility (SLF).

**NOJOURNAL**
The matched information is not recorded.

**LOG**
A message is written to the AF/OPERATOR message log every time the trap is triggered.

**NOLOG**
No message is written to the AF/OPERATOR message log when the trap is triggered. Also, the results of the trap action are not logged.

**RETAIN**
Information about each match is recorded for display by the AF/OPERATOR RETAIN*VIEW facility.

**NORETAIN**
Match information is not recorded for the AF/OPERATOR RETAIN*VIEW facility.

**TEST**
When the trap is triggered, no direct or extended action is taken. The only commands that execute are the ones that execute in warn mode. However, the AF/OPERATOR log generates a record of the action that would have been taken.

**NOTEST**
When the trap is triggered, the programmed action is taken.

**Subsystem Logging Facility (SLF) Support**

**JOURATTR**
Specifies optional attributes of SLF data:

- **color**
  Valid colors are blue, red, pink, green, turq, yellow and white.

- **highlight**
  Valid highlight values are none, blink, rvideo, and underline.

- **intensity**
  Valid intensity values are normal and high.

**EXAMPLE:** `ATTR(COLOR(RED),HIGH(BLINK),INTEN(HIGH))`

**JOURHELP**
Specifies the name of an optional help panel to be displayed when viewing the SLF data.
Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Trap was successfully changed.</td>
</tr>
<tr>
<td>16</td>
<td>Invalid syntax.</td>
</tr>
</tbody>
</table>

Comments

- To change the definition of a trap, specify new options to replace old ones. You need only specify options to be changed, except that you must code the xtype even if the xtype text pattern is not to be changed.

- For information about using quotes and ampersands in the ACTION or ALTACT field of a TRAP command, see the AF/OPERATOR User's Guide.

- An xtype trap will not function for an OMEGAMON session unless the LOGON command that established the session specified a time interval with the INTERVAL operand.
TRAP DELETE

Description
TRAP DELETE deactivates one or more traps and deletes their definitions. (Can be issued from the console.)

Important
Use any form of TRAP DELETE(*) FORCE with extreme caution. You might delete more traps than you intended, including AF/OPERATOR’s internal command traps.

Syntax

```
TRAP DELETE(pattern/s) <subsys>

   CMD        DISABLE        FORCE
   TOD        ENABLE
   WTO

   xtype
```

subsys This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

(pattern/s) A name pattern intended to match one or more trap names. You may use up to 32 patterns.

CMD Deletes only CMD traps.
TOD Deletes only TOD traps.
WTO Deletes only WTO traps.

xtype Deletes only OMEGAMON exception traps. The following types are supported:

- XO2 OMEGAMON for DB2
- XOC OMEGAMON for CICS
- XOI OMEGAMON for IMS
- XOM OMEGAMON for MVS

DISABLE Deletes only disabled traps.
ENABLE Deletes only enabled traps.
FORCE Deletes all traps, including the command traps that operate AF/OPERATOR. Use this command with extreme caution.
Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Trap is deleted.</td>
</tr>
<tr>
<td>16</td>
<td>Invalid syntax.</td>
</tr>
</tbody>
</table>

Example

The following is an example of this command embedded in a REXX exec.

```
"TRAP DELETE(SUP*) WTO"     Deletes all WTO/WTOR traps with names beginning with SUP.
"TRAP DELETE(*) WTO"        Deletes all WTO/WTOR traps.
"TRAP DELETE(*) CMD"        Deletes all command traps, excluding the command traps that operate AF/OPERATOR.
"TRAP DELETE(*)"            Deletes all traps.
"TRAP DELETE(*) FORCE"      Deletes all traps, including the traps that operate AF/OPERATOR. *This command should be used with extreme caution.*
TRAP DISABLE

Description
TRAP DISABLE deactivates a trap previously added (defined) with a TRAP ADD command and previously enabled. The trap definition is not deleted. (Can be issued from the console.)

Important
Use any form of TRAP DISABLE(*) FORCE with caution. You might disable more traps than you intended, including AF/OPERATOR’s internal command traps.

Syntax

```
subsys TRAP DISABLE(pattern/s) < t subsystem
                CMD < t CMD
                TOD < t TOD
                WTO < t WTO
                xtype < t xtype
                FORCE
```

subsys
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

(pattern/s)
A name pattern intended to match one or more trap names. You may use up to 32 patterns.

CMD
Disables only CMD traps.

TOD
Disables only TOD traps.

WTO
Disables only WTO traps.

xtype
Disables only OMEGAMON exception traps. The following types are supported:

- XO2  OMEGAMON for DB2
- XOC  OMEGAMON for CICS
- XO1  OMEGAMON for IMS
- XOM  OMEGAMON for MVS

FORCE
Disables all traps, including the traps that operate AF/OPERATOR. Use this command with extreme caution.
Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Trap is disabled.</td>
</tr>
<tr>
<td>16</td>
<td>Trap is in use and cannot be disabled, or invalid syntax.</td>
</tr>
</tbody>
</table>

Comments

- A successfully executed DISABLE command results in two or more messages: a TRAP trapname ENABLED message for each trap enabled, and a single n TRAPS DISABLED message displaying the total number of traps disabled. If you try to disable a trap that has not been defined, an error message displays.
- When a TRAP DISABLE command is issued, actions in progress continue until they are complete. To stop these actions, use the AF/OPERATOR AF CANCEL command.

Example

The following is an example of this command embedded in a REXX exec.

```
“TRAP DISABLE(SUPNODD)”  Disables a TRAP named SUPNODD.
“TRAP DISABLE(SUP*)”     Disables all traps with names beginning with SUP.
“TRAP DISABLE(SUP*) WTO” Disables all WTO/WTOR traps with names beginning with SUP.
```
TRAP ENABLE

Description
TRAP ENABLE activates a trap that (a) was previously added (defined) by a TRAP ADD command and (b) was either not enabled or was disabled. (Can be issued from the console.)

Syntax

```
TRAP ENABLE(subsys(pattern/s)|CMD|TOD|WTO|xtype)
```

- **subsys**
  This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The `subsys` name is assigned by the SUBSYS startup parameter.

- **(pattern/s)**
  A name pattern intended to match one or more trap names. You can use up to 32 patterns.

- **CMD**
  Enables only CMD traps.

- **TOD**
  Enables only TOD traps.

- **WTO**
  Enables only WTO traps.

- **xtype**
  Enables only OMEGAMON exception traps. The following types are supported:
  - **XO2** OMEGAMON for DB2
  - **XOC** OMEGAMON for CICS
  - **XOI** OMEGAMON for IMS
  - **XOM** OMEGAMON for MVS

If neither CMD, TOD, WTO, or xtype is used, every trap whose name matches a specified pattern is enabled.

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Trap is enabled.</td>
</tr>
<tr>
<td>16</td>
<td>No free trap entries, or invalid syntax.</td>
</tr>
</tbody>
</table>

Comments
TRAP ENABLE

- The TRAP ENABLE command activates traps either during product startup or during the course of the product's operation. Enabled traps become active immediately.
- If you try to enable a trap that has not been added, or that is already enabled, the following error message displays:

  !AOP0170 - NO TRAPS ENABLED

Example
The following is an example of this command embedded in a REXX exec.

  "TRAP ENABLE(SUP*)" Enables all traps whose names begin with SUP.
  "TRAP ENABLE(SUP*) WTO" Enables all WTO/WTOR traps whose names begin with SUP.
  "TRAP ENABLE(SUPNODD)" Enables a trap named SUPNODD.

A typical response to this command follows:

  !AOP0171 WTO TRAP SUPNODD ENABLED
  !AOP0170 1 TRAPS ENABLED
WAIT

Description
WAIT stops execution of a REXX exec until a specified command or message is received by AF/OPERATOR or until a specified time elapses.

Syntax

WAIT

CMD('full-pattern,s')

WTO('full-pattern,s') option/s

->

AFTER('command') SECONDS(number)

Options:
- Triggering Conditions
  - ASID(number)
  - CONSOLE(id/name,s)
  - DESC(code,s,n:n,n-n)
  - JOBCLASS('jobclasses')
  - JOBNAMR(pattern)
  - JOBTYPF(option)
    - Options:
      - IMS JOB STC TSU
    - ROUT(code,s,n:,n-n)
    - SYSID(pattern)
- Immediate Action
  - SUPPRESS|ALLOW
- Trap Control
  - WAIT|CONTINUIR
  - AFTER('command')
  - SECONDS(number)

CMD('full-pattern,s')

Specifies that AF/OPERATOR will halt and wait to resume operation for an operator command issued through SVC 34, the IMS or JES3 interface, or other user-defined sources, if the command text matches the pattern.

The pattern can be up to 128 characters long. It can contain pattern characters, pattern variables, and continuation signs. You can use up to 10 patterns; however, the total length of the WAIT command cannot exceed 255 characters.
WAIT

WTO('full-pattern,s')

Specifies that AF/OPERATOR will halt and wait to resume operation for a WTO/R if the WTO/R text matches the pattern.

The pattern can be up to 128 characters long. It can contain pattern characters, pattern variables, and continuation signs. You can use up to 10 patterns; however, the total length of the WAIT command cannot exceed 255 characters.

Note: ALLSYS does not apply to traps generated internally by the WAIT command.

Triggering Conditions

ASID CONSOLE DESC JOBCLASS JOBNAME JOBTYPe ROUT SYSID

These operands have the same meaning as they do in the TRAP ADD CMD and TRAP ADD WTO commands. They select operator commands or WTO/WTOR messages by origin. The origin can be specified in terms of address space, console, descriptor code, job type, job name, job class, route code, and SMF system IDs.

Immediate Action

SUPPRESS

The trapped message is not displayed. The issuing address space continues as if the message had been displayed normally, and the message appears in SYSLOG.

ALLOW

The trapped message is displayed as intended.

Trap Control

WAIT

The address space from which the message was issued is suspended while AF/OPERATOR is executing the trap action.

CONTINUE

The address space from which the message was issued is not suspended. The trap action occurs while the address space continues normal execution.

AFTER('command')

The waiting period will begin when the specified operator command is issued. If this command returns a non-zero return code, the AOLASTCC variable will contain the return code of this command. The temporary WAIT traps will not be set. The following table indicates how to use AOLASTCC to determine the cause of the error.

<table>
<thead>
<tr>
<th>If . . .</th>
<th>then the timer value set by WAIT SECONDS() expired and . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOLASTCC is 0</td>
<td>there was no WAIT AFTER() command specified or if there were, it worked.</td>
</tr>
<tr>
<td>AOLASTCC is not 0</td>
<td>the WAIT AFTER() command was specified and it failed.</td>
</tr>
</tbody>
</table>

SECONDS(number)

The length of the maximum waiting period in seconds. If this period is exceeded, the REXX exec resumes execution.
Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>More than 99 console items are specified, no trap entries are free, no major keyword is specified, invalid console name, or invalid syntax.</td>
</tr>
</tbody>
</table>

Comments

- You can create pattern variables using the WAIT command. Use the same rules as in creating pattern variables with the TRAP ADD command. You can use the same variable name in any one or more of the 10 patterns.

- Qualify a WAIT command as fully as possible (by including information such as job name, ASID, and system ID) to assure that any message or command that matches a specified pattern is the one you want.

If a WAIT command is not properly qualified, it may be matched by the wrong command or message. Furthermore, if more than one match is active at a time, temporary traps from the wrong match may have to wait. Then when triggering events occur for this match, there are no traps available to act on them.

- The WAIT command can be satisfied by messages issued by AF/OPERATOR or by MVS or MVS subsystems.

- The AFTER keyword allows you to synchronize commands and awaited responses. You can think of the statement as saying, “WAIT for this message AFTER issuing this command.”

Use AFTER(‘command’) to be sure the WAIT trap is set before ’command’ is issued. This prevents the command response from appearing before the WAIT trap is set.

- Use the WAIT option of the WAIT statement with caution. Consider the length of time it takes to complete the action for a trap match and the type of address space. For example, most operator commands and certain system messages are issued from the console address space. Putting the console address space in a WAIT results in the loss of all console functions.

'Mechanism of the WAIT Command'

The WAIT command sets 1 to 10 temporary wait traps. Their names are of the form:

- \( AnCmmmnnn \) AF/OPERATOR CMD traps
- \( AnWmmmmm \) AF/OPERATOR WTO traps
- \( A0Tmmmmm \) AF/OPERATOR TOD traps

The variable \( n \) is a digit from 0 through 9 and the variable \( mmmmm \) is the last five digits of the match number under which the WAIT was executed. The SECONDS keyword causes a temporary trap named \( A0Tmmmmm \) to be added.
All temporary traps are deleted when the WAIT is satisfied. If a match becomes hung, use the AF CANCEL(matchnumber) command to cancel the match and delete all associated temporary wait traps. See “AF CANCEL” on page 195.

‘Determining Which Event Ended the Waiting Period’

The REXX exec can determine which of the messages or commands ended the WAIT by testing the variable AOCASE.

- AOCASE is assigned a value from 1 to 10, corresponding to the first through tenth WTO or command pattern entered in the WAIT statement.
- A value of 11 from AOCASE indicates an error from the WAIT AFTER ('command') or that the WAIT SECONDS() timeout limit expired, or both. AOCASE puts the value of the return code from the WAIT statement in AOLASTCC. You can use AOLASTCC to determine the cause of the error. For example, consider the following results associated with the WAIT command:
  - If the AFTER('command') was issued successfully (or was not there), and the SECONDS expired, the return code (AOLASTCC) equals 0 and AOCASE=11.
  - If the AFTER('command') did not work, and SECONDS was coded, the WAIT return code (AOLASTCC) is the return code from the AFTER('command') and AOCASE=11.
  - If the AFTER('command') did not work, and SECONDS was not coded, the WAIT return code (AOLASTCC) is the return code from the AFTER('command') and AOCASE=11.

Example

The following is an example of this command embedded in a REXX exec.

```
“WAIT WTO(‘ALL FILES CLOSED’),
   “AFTER(‘OPER ‘F TASK,CLOSE’ CONS(1)’),
   “SECONDS(300)”
```

A more sophisticated example of the WAIT command involves waiting for more than one event:

```
“WAIT WTO(‘ALL FILES CLOSED’;’FILE CLOSE ERROR’),
   “AFTER(‘OPER ‘F TASK,CLOSE’ CONS(1)’),
   “SECONDS(300)”
```

This command instructs AF/OPERATOR to wait for either the message ALL FILES CLOSED or the message FILE CLOSE ERROR.

Note that if AOCASE=1, the ALL FILES CLOSED WTO ended the wait. If AOCASE=2, the FILE CLOSE ERROR WTO ended the wait. Otherwise, AOCASE is set to the value 11 to indicate that the 300 second time limit has expired.
WRITE

Description
WRITE adds text to a file previously opened for output.

Syntax

```
WRITE filename 'text'
```

**filename**  The name of the file to which the text will be written.

**'text'**  A line or block of text to be written to the specified file.

Example

```rexx
/* REXX */
indsn = "'VALID.INPUT.DSN'"
outdsn= "'VALID.OUTPUT.DSN'"
ADDRESS AFHOST
 "ALLOC DDNAME(IFILE) DSN(''|indsn|') SHR"
 "ALLOC DDNAME(OFILE) DSN(''|outdsn|') SHR"
 "OPEN IFILE INPUT GLOBAL"
 "OPEN OFILE OUTPUT GLOBAL"
 iorecord="'
 status='GLBVPUT('IORECORD')
 "READ IFILE IORECORD"
 status='GLBVGET('IORECORD')
 DO WHILE(iorecord ^= '')
   "WRITE OFILE ''||iorecord||'''
   "READ IFILE IORECORD"
   status='GLBVGET('IORECORD')
 END
 "CLOSE IFILE"
 "CLOSE OFILE"
 "FREE DDNAME(IFILE)"
 "FREE DDNAME(OFILE)"
ADDRESS
RETURN
```
WTO

Description
WTO issues a message to one or more consoles. (Can be issued from the console.)

Syntax

```
subsys

WTO 'message'

CONSOLE(id/name)

ROUT(code,s,n:n,n-n)

DESC(code,s,n:n,n-n)

> MSGID(id)

NOMSGID

WTONO(varname)

SYSID(linkid)
```

`subsys`  
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The `subsys` name is assigned by the SUBSYS startup parameter.

`'message'`  
A message from 1 to 125 characters in length. (The message will be truncated to a maximum of 115 characters if a MSGID is required, either by specification or by default.) You can use continuation signs, but not in column 72.

`CONSOLE(id/name)`  
A console ID or name which is the message destination.

`ROUT(code,s,n:n,n-n)`  
Route codes for the message. The value of the code is an integer from 1 to 128. If not specified, the default from the MSGDFLT startup option will be used.

`DESC(code,s,n:n,n-n)`  
Descriptor codes for the message. The value of the code is an integer from 1 to 16. If not specified, the default from the MSGDFLT startup option will be used.

`MSGID(id)`  
Messages issued by the WTO command will be prefixed by the default prefix. If you don't specify MSGID or NOMSGID, the default prefix for AF/OPERATOR messages is !AOP0000. You can use this parameter to specify a user-defined ID. The value of this ID is a string of up to 8 characters.

`NOMSGID`  
Messages issued by the WTO command will not be prefixed by any user-specified ID or the default prefix (!AOP0000).

`WTONO(varname)`
A character variable in which the message number will be stored. If the variable is not already defined, the system creates it in the global pool. If the variable is already defined, it must have been declared with:
- global scope
- character type

**Note:** This option is not supported when the WTO is being issued to a remote system.

**SYSID(linkid)**

Defines a target SYSID in a network where the WTO command will be transmitted and executed when using the Multi-System Management Facility (MSMF). The SYSID must match a linkid from a previously-defined link definition.

## Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Message is issued to a console.</td>
</tr>
<tr>
<td>1</td>
<td>MSMF is not active.</td>
</tr>
<tr>
<td>4</td>
<td>Message exceeded the 125-character maximum, or 115-character maximum with MSGID, and was truncated.</td>
</tr>
<tr>
<td>12</td>
<td>Peer-to-peer is not active for specified SYSID.</td>
</tr>
<tr>
<td>16</td>
<td>Invalid console name, invalid syntax, or message length greater than 125 characters.</td>
</tr>
<tr>
<td>50</td>
<td>Peer-to-peer link is not active.</td>
</tr>
<tr>
<td>51</td>
<td>Peer-to-peer link is not defined.</td>
</tr>
</tbody>
</table>

### Comments

- If CONSOLE and ROUT are omitted, the default routing set by AF/OPERATOR will depend upon the type of match created as a result of the original triggering event. If the WTO is being issued as a result of a CMD type match, AF/OPERATOR does not set any default route code; instead, it sends the message to the console that issued the originating command. If the message is being issued as a result of any other match type (for example, WTO or TOD match type), then AF/OPERATOR sets a default descriptor code determined by the MSGDFLT startup option. If DESC is omitted, AF/OPERATOR uses its default route code determined by the MSGDFLT.

- A default AF/OPERATOR startup REXX exec can be used to define a number of system variables for commonly used routing and descriptor codes. These names can then be used in a WTO command through symbolic substitution.

  For example, if the startup REXX exec contains the command
  ```rexx
  IMMECT = ’2’
  ```

  then
  ```rexx
  “WTO **** CODE RED **** DESC(&IMMECT)”
  ```
sends a message with a descriptor code of 2.

- If SYSID is specified, the WTO command output is routed for execution through the MSMF interface to the targeted site, provided the supporting network environment is defined and operational.

Example

The following is an example of this command embedded in a REXX exec.

```
"WTO 'CRITICAL PROBLEM ON SYSTEM A' DESC(2) WTONO(TOKEN)"
```

The following example shows a WTO that is sent to SYSB from the local system using the MSMF interface:

```
"WTO 'GOOD MORNING' SYSID(SYSB)"
```
WTOR

Description
WTOR issues a message to one or more consoles and waits for an operator reply. (Can be issued from the console.)

Syntax

```
subsys
WTOR 'message' -- REPLY(varname) --
CONSOLE(id/name) ROUT(code) 
DESC(code) MSGID(id) TIMEOUT(sec 'reply' )

subsys
This operand is only required when issuing the command from the console. This is the name of the AF/OPERATOR address space where you want the command to execute. The subsys name is assigned by the SUBSYS startup parameter.

'message'
A message from 1 to 125 characters in length. (The message will be truncated to a maximum of 115 characters if a MSGID is required, either by specification or by default.) You can use continuation signs, but not in column 72.

REPLY(varname)
A character variable where the system stores the reply for this WTOR. The system stores the reply in an existing local or global variable if it finds a variable name matching the varname you enter. If both variables exist, the system uses the local variable. If the system finds a matching variable too short to contain the reply, it sets AOlASTcc to 4 and truncates the reply to fit the variable length. If no matching local or global variables exist, the system creates a global variable and deletes any trailing blanks.

CONSOLE(id/name)
A console ID or name which is the message destination.

ROUT(code,s,n:n,n-n)
A route code for the message. The value of routecode is an integer from 1 to 128.

DESC(code)
A description code for the message. The value of the code is an integer from 1 to 16.

MSGID(id)
Messages issued by the WTOR command with a coded MSGID will be prefixed by messageid instead of the default prefix !AOP0000. The value of messageid is a string of up to 8 characters.
**NOMSGID** If NOMSGID is coded, then no prefix is appended to the WTO. If NOMSGID is not coded, then the default prefix !AOP0000 is appended to the WTO.

**TIMEOUT**
- `sec` - Specifies the amount of time to wait before deleting the operator message. The value can be an integer from 5 to 3600.
- `reply` - Supplies a default reply. The reply can be up to 119 characters. If no value is specified, the default is a single character blank.

## Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Message is issued to a console, with a wait for reply.</td>
</tr>
<tr>
<td>4</td>
<td>Message exceeded the 125-character maximum, or 115-character maximum with MSGID, and was truncated.</td>
</tr>
<tr>
<td>16</td>
<td>Invalid console name or ID, invalid syntax, or invalid message length (greater than 125 characters).</td>
</tr>
</tbody>
</table>

## Comments
- If CONSOLE and ROUT are omitted, the default routing set by AF/OPERATOR will depend upon the type of match created as a result of the original triggering event. If the WTOR is issued as a result of a CMD type match, AF does not set any default route code; instead, it sends the message to the console that issued the originating command. If the message is being issued as a result of any other match type (for example, WTO or TOD match type), then AF/OPERATOR sets a default descriptor code determined by the MSGDFLT startup option. If DESC is omitted, AF/OPERATOR uses its default route code determined by the MSGDFLT.
- Use the ROUT option to route messages to multiple consoles. If the WTOR is routed to more than one console, any console can reply to the message.
- If a WTOR is sent with a route code of 9, and the reply is returned in a full reply form (such as `R nn,text`; not `nn,text` or `nn,text`), the reply is suppressed. That is, when it is entered at the keyboard it is not echoed to the screen, and in SYSLOG and RKOGLOGM it is replaced by the word **SUPPRESSED**.

If you anticipate a response that is security sensitive or private (a password, for example), use ROUT(9) to prevent the response from going to any system activity logs.
You can use the Programmerless Open VTAM Interface (POVI) commands to accelerate script development in the POVI environment. These commands expedite the construction process because they allow you to manipulate variables and screens, and to simulate pressing function keys. See “Programmerless Open VTAM Interface” on page 327 for a list of operator commands you can use from the console.

**Note:** Many of the POVI environment commands can be replaced by POVI REXX functions. These functions provide better error checking and, therefore, are easier to use. See “POVI Functions” on page 121 for more information.

This section describes the following POVI commands:

- `\@TYPE` Types a variable at a specified row and column.
- `\#TYPE` Types a variable at a specified field.
- `\K` Simulates pressing a function key.
- `\LOC` Positions the cursor at a specified row and column.
- `\RC` Plays back the named script.
- `\S` Starts a session between POVI and the specified VTAM application.
- `\SEE` Displays variables.
- `\SET` Sets variables.
- `\STATUS` Inspects and returns the status of a VTAM application.
- `\TYPE` Types a variable at the cursor position.
- `\WAIT` Suspends execution for a time specified in seconds.
\@TYPE

**Description**
\@TYPE types a variable at a specified row and column in an underlying VTAM application.

**Syntax**
\@TYPE (rrccc)varnam

**rr** The screen row to place the data. Values 00-42 are valid. You must specify leading zeros.

**ccc** The screen column to place the data. Values 00-131 are valid. You must specify leading zeros.

**varnam** The POVI variable to place into the screen buffer.

**Comments**
- This command removes the need to record a keystroke script to place a variable into a specific screen location.
- In some instances, \#TYPE can be more advantageous to use because it is less dependent upon specific screen locations.
\#TYPE

Description
\#TYPE types a variable at a specified input field in an underlying VTAM application.

Syntax

\#TYPE (nnn) varnam

nnn The unprotected field in which to place the data. The command ignores leading zeros. Any number of digits is allowed. Zero and negative numbers are valid.

varnam The POVI variable to place into the screen buffer.

Comments
- This command removes the need to record a keystroke script to place a variable into a specific screen field.
- Positive field numbers (nnn) for specific screens begin at the first unprotected input field 1, and count across and down the screen as you press TAB. To locate field 1, press HOME, or any key that moves the cursor to the first input field on the screen. Thereafter, to land on field (nnn), press the TAB key (nnn-1) times.
- Negative field numbers (-nnn) begin at the last unprotected input field -1, and count up and across the screen as you press the BACK TAB key. To locate field -1, press any key that moves the cursor to the first input field 1, then press the BACK TAB key once. Thereafter, to land on field -3 for example, press the BACK TAB key twice.
- Specify zero (0) varnam to place a variable at the current cursor position.
- Keep in mind that field numbers vary according to the screen displayed.

Note: Always verify that you enter data into unprotected input fields.

- \#TYPE offers an advantage over @TYPE; \#TYPE is less dependent upon specific screen locations.


Description

\K simulates pressing a function key.

Note: See “POVI Functions” on page 121 for information on the POVISEND function, which can simulate an AID or function key.

Syntax

// keyval

**keyval** Name of the function key to be pressed. The following entries are valid:

- F1 through F24
- PA1 through PA3
- HOME
- CLEAR
- ENTER
- UP
- DOWN
- LEFT
- RIGHT
- TAB
- BACKTAB
- ERASEEOF
- ERASEINP

Comments

- This command removes the need to record a keystroke script to playback a single function key.
- This command transmits the requested keystroke to the current VTAM application.
\LOC

**Description**
\LOC positions the cursor at a specified row and column in an underlying VTAM application.

**Note:** See “POVI Functions” on page 121 for information on the POVILOC function, which positions the cursor on a simulated 3270 screen.

**Syntax**

\LOC (rrcc)

**rr** The screen row to place the cursor. Values 00-23 are valid. You must specify leading zeros.

**cc** The screen column to place the cursor. Values 00-79 are valid. You must specify leading zeros.

**Comments**
- This command removes the need to record a keystroke script to position the cursor at a specific screen location.
- (0000) is the top-left screen position.
- You must specify four digits. For example, to specify row 4 and column 2, enter **(0402)**.
\RC

**Description**

\RC plays back a previously recorded script.

**Note:** See “POVI Functions” on page 121 for information on the POVIPBS function, which initiates playback of a script.

**Syntax**

\RC (scriptname)

**scriptname**  The name of the script. It can be a maximum of 8 characters, the first of which must be alpha or national. The remainder may be alphanumeric.

**Comments**

- In order to use the \RC command, you need to have recorded and stored the script in the OVIUSER VTAM session.
- The effect of issuing \RC is that a multiple of other commands discussed are issued.
- When \RC completes, the screen that VTAM has presented to POVI in the script is presented in line variables.
- Any POVI variables referenced by the script must be set using the \SET command prior to issuing \RC.
Description
\$ starts a session between POVI and the specified VTAM application.

**Note:** See “POVI Functions” on page 121 for information on the POVILGON function, which creates a POVI environment and then starts a session with a VTAM application.

Syntax

```
\$ applid [M] [D] c
```

- **applid** The applid of the VTAM application.
- **M** The logmode to be used for the session.
- **D** Up to 40 characters of data to be passed to the application when the session begins.
\SEE

**Description**
\SEE displays POVI variable contents.

**Syntax**

```
\SEE variable
  .
  .
  'see end'
```

**variable** Name of the POVI variable to display.

**Comments**

- When invoked from the Programmerless Open VTAM Interface, the command displays a full screen and begins displaying output at location 0,0. Pressing ENTER returns you to the previous screen.
- This two-screen aspect of \SEE implies a definite construction style when used within REXX execs. As a result, the line following \SEE variable must manipulate the line variable displayed, and the next line must be a throwaway line that simulates pressing ENTER. For example, the following 3 lines exhibit proper construction for \SEE:

```
/* REXX */
"\SEE varname"
RET = GLBVGET('varname')
"'see end'"
```
**

**SET**

**Description**

SET sets variables.

**Note:** See “POVI Functions” on page 121 for information on the POIVSET function, which sets variables.

**Syntax**

```
\SET variable
'literal' or 'varname'
```

**variable** Name of the POVI variable to set.

**Comments**

- This command places AF/OPERATOR-determined variables into POVI variables, which later can be placed into specific screen locations by user-recorded scripts and Programmerless Open VTAM Interface commands.

- When invoked from the Programmerless Open VTAM Interface, the command presents a blank screen for data input. The system places all input data that you type on this screen into the named variable. Pressing ENTER returns you to the screen on which you typed \SET.

- This two-screen aspect of \SET implies a definite construction style when used within REXX execs. As a result, the line following \SET variable must set the line variable value within quotes. The value can be a literal value or an AF/OPERATOR-defined variable, such as ‘&USERID’.

For example, the following two lines exhibit proper construction for \SET:

```
/* REXX */
"\SET varname"
"'USER01'"
```
\STATUS

Description
\STATUS inspects and returns the status of a VTAM application.

Syntax

/* REXX */
\STATUS sessid
.
.
'status end'

sessid The session ID (not the applid) of the VTAM application to test.

Comments

- This command allows you to check the status of a VTAM application before executing any recorded scripts or Programmerless Open VTAM Interface commands against it.
- When invoked from the Programmerless Open VTAM Interface, the command displays a full screen and begins displaying one the following status messages at location 0,0:

  ACTIVE Application is active.
  INACTIVE Application is inactive.
  UNKNOWN SESSION Session ID is not defined to the Programmerless Open VTAM Interface.
  UNDEFINED Session is undefined to the Programmerless Open VTAM Interface, but not to VTAM.
  STOPPED Application has issued SETLOGON OPTCD=STOP.
  QUIESCED Application has issued SETLOGON OPTCD=QUIESCE.
  STATUS UNKNOWN An unexpected status code was returned.

Pressing ENTER returns you to the previous screen.

The following is an example of how to use the \STATUS command:

/* REXX */
"\STATUS " SESSID
RET = GLBVGET('LINE1')
"'status end'"
say "Status " SESSID LINE1
if line1 <> "ACTIVE" then return(4)
TYPE

Description

TYPE types a variable at the current cursor location in an underlying VTAM application.

Syntax

\TYPE varnam

varnam The variable to place into the screen buffer.

Comments

- This command removes the need to place a variable into a specific screen location to record a keystroke script.
\WAIT

Description
\WAIT inserts wait times specified in seconds.

Note: See “POVI Functions” on page 121 for information on the POVIDLAY function, which specifies a wait time and then refreshes the terminal buffer.

Syntax

\WAIT time

time The number of seconds you want the system to wait before exiting to the underlying VTAM application.

Comments
- This command refreshes the terminal screen after waiting a specified time and then exits to the underlying VTAM application.
- The command allows underlying VTAM applications to finish screen updates, so complete screens are transmitted back to AF/OPERATOR at command termination for use in REXX execs. This is an important command to use, because AF/OPERATOR cannot handle incomplete screens.
- Due to system variances caused by factors such as workload fluctuations, there is no guarantee that a wait time that works once will always work. This is especially true when working with applications that deliver streams of partial screen updates with the keyboard unlocked.
Programmerless Open VTAM Interface Operator Commands

The set of commands described in this section is provided to control the Programmerless Open VTAM Interface address space. You can execute these commands from the Operator facility (that is, the POVI management applications CUA Operator or Basic Operator), or you can execute them from the MVS console (except for MONITOR). From the MVS console, the command format to use is:

```
F POVIPROC,command
```

where POVIPROC is the started task name for the POVI address space under which all of the POVI applids run. Your data center defines this name during AF/OPERATOR customization.

Note:
1. You can only issue these commands from the console or from the Operator Facility. They are not available from REXX.
2. Access to the Operator facility may be restricted to the POVI administrator at your site.

To access the POVI CUA Operator from a VTAM terminal, enter:

```
logon applid(OVICUA)
```

OVICUA is the Candle default APPLID for the POVI management application CUA Operator. The APPLID at your site may differ if the default names were not used during customization at your site.

When the system prompts you for a user ID and password, enter the same user ID and password that you usually use to log onto POVI. To log off, execute the LOGOFF command with no options.

The CUA Operator has an online Help system that provides help for individual field information and extended help for panels. To access the Help system, either press PF1, when the cursor is positioned on the panel or field where you need help, or tab to the Help option on the Action Bar and press Enter.

This section describes the following VTAM operator commands:

- **ECHO**
  - Writes the specified string to the terminal and empties the message buffer.
- **LOGOFF**
  - Requests termination of a logical unit.
- **MONITOR**
  - Varies message types received by VTAM operators.
- **REFRESH**
  - Loads a new copy of a member in your panel library.
- **VSHOW**
  - Displays the status of POVI users, devices, or sessions.
**ECHO**

**Description**
ECHO allows you to echo text back to the terminal and force all buffered messages to the POVI log (KLVLOG).

**Syntax**
```
ECHO string
```

**string** Any character string to be echoed back to the issuing terminal.

**Comments**
- Use ECHO to verify that the Operator facility is working properly and to force all buffered messages to the log.
- Enclosing `string` in single quotes is necessary only if you want to preserve leading blanks.

**Example**
To test the Operator facility, issue the following command:
```
ECHO 'OPER.TEST, 12-21-02, 2100'
```
LOGOFF

Description
LOGOFF requests termination of a logical unit (LU).

Syntax

LOGOFF <option/s>

Options:

- session

The LU name (1 to 8 characters) of the session to be terminated. If you do not specify session, the system logs off the issuing operator session.

Example
Here is an example of logging a session off POVI:

LOGOFF L001234
MONITOR

Description
MONITOR allows you to control the message types to be displayed on the POVI Operator facility console. (This command cannot be issued from the MVS console.)

Syntax

MONITOR pcccc ❯ pcccccc...

p Specifies enabling (+) or disabling (-) display of a message class (cccc).
cccccc Specifies the message class:
  ALERT
  ALL (all message types)
  INFO
  LOG
  REPLY
  VIEW
  WARN

Comments
- You can specify multiple parameter values (cccccc).
- The POVI initialization member (KOGPSTRT) controls whether diagnostic logging messages are sent to VIEWLOG. The MONITOR command can limit the types of messages sent.
- Under normal conditions, you may want to monitor only those messages that POVI issues as log messages, but for diagnostic activities, issue MONITOR +ALL.
- The MONITOR command is completely unrelated to the MONITOR initialization parameter found in the VIGINVGW member of VTPILIB. The MONITOR parameter specifies in minutes the interval between application status tests. When the interval expires, each application defined in an APPLDEF statement is interrogated by VTAM INQUIRE. The status returned is then displayed on the menu.

Example
To monitor log messages at the terminal, issue:

MONITOR +LOG
REFRESH

Description
REFRESH loads a new copy of the specified member in your panel library (hilev.TKANPENU).

Syntax

REFRESH PANEL name

name The name (1 to 8 characters) of a member in your panel library.

Comments
- Use REFRESH to enable changes made to members in TLVPNLS after POVI has been started.

Example
To refresh the member KOGPSTRT, issue the following command:

REFRESH PANEL KOGPSTRT
VSHOW

Description
VSHOW displays the status of selected Programmerless Open VTAM Interface users or devices.

Syntax

VSHOW option/s

Options:
  userid
    /  POOL=pool
    PLU=plu
    SLU=slu
    ID=termid
    ACTIVE
    INACTIVE
    STATS
    DISCONNECTED
    FOREGROUND
    SUMMARY

userid     Specifies the user ID (1 to 8 characters).
  - An asterisk (*) can be used as a wild card.
  - If entry validation is not being used (that is, if your installation options specify that this user can log on without entering a user ID and password), specify the terminal ID instead.
  /     Requests all users along with their terminal IDs.

pool      Specifies the virtual terminal pool name. If this parameter is specified, only users with this virtual terminal pool name (pool) are displayed.

plu       The primary logical unit name (1 to 8 characters). This is the VTAM network name of the application program. If this parameter is specified, only users with a session defined for this application (plu) are displayed.

slu       The secondary logical unit name (1 to 8 characters). This is the VTAM network name of the virtual terminal that has been allocated to a virtual session. If this parameter is specified, only users in session with this virtual terminal name (slu) are displayed.

termid    Limits the display to users with the specified terminal ID.

ACTIVE     Displays all active sessions for qualifying users.

INACTIVE   Displays all inactive sessions for qualifying users.

STATS      Displays statistics on the number of bytes and the number of messages for each virtual terminal name.

DISCONNECTED
Displays all disconnected users.

**FOREGROUND**
Displays all users in foreground.

**SUMMARY**
Displays a summary of the users logged on.

**Example**
To list all active sessions, users, and terminals, issue:

```
VSHOW /
```
Sample Exec to Interface with OMEGAMON

This appendix contains an example of a REXX exec using AF/OPERATOR commands. Its operation is explained in the comments before the exec. The example assumes that the OMEGAMON for MVS session was previously established with a LOGON command.

FIGURE 9. OMEGAMON Interface Exec

/* REXX *******************************************************
 * This routine must have room for sequence numbers in columns*
 * 1-8 and 73-80 so that it can be installed by the user      *
 * in either a fixed or variable length record library. It     *
 * must not be numbered until after installation.             *
***************************************************************

Name       - TESTREXX
Synopsis    - EX TESTREXX or CALL "TESTREXX"
Function    - Send a message to all TSO users that have
              accumulated more than a threshold number of CPU
              seconds.
Parameters  - None.
Operation   -
  1) Issue Omegamon TSOJ major command and CPU minor command
  2) Parse display image returned in AF/OPERATOR global line
     variables.
  3) Examine each TSO id and compute total CPU seconds if not
     in minutes and seconds.
  4) If a TSO user has accumulated more than a threshold
     amount of CPU seconds, parse the TSO userid from the
     appropriate AF/OPERATOR line variable and issue an
     operator command to send a humorous message.
Returns:

0 - no errors detected.
other - error in some invoked routine.

***************************************************************
* To assist inexperienced REXX coders, all REXX keywords are *
* coded in uppercase, and all variable names are coded in     *
* lower case. This isn’t required; it is merely a convention. *
* REXX function names are quoted to reduce the search path. *
***************************************************************

threshold = 60 /* Set CPU seconds limit constant */
count = 0 /* Set count of notified users to 0 */
msgtext = "Excessive use of CPU resources for this TSO ID"

/* Initialize line prefix name and issue OMEGAMON commands */
aolinpfx = "OM" /* Set line variable prefix to "OM" */
status = 'GLBVPUT’("AOLINPFX") /* Copy to global variable */
ADDRESS AFHOST /* Make AFHOST the REXX environment */
  "AFADDR OMVS" /* AF/OPERATOR access to OMEGAMON */
  IF rc ^= 0 THEN DO
    SAY 'Unable to access Omegamon. address RC=' rc
    EXIT rc
  END
  "TSOJ"
  "CPU"
  "RC"
  "AFADDR" /* End of OMEGAMON access */

  /* NOTE: Until the AF/OPERATOR all address is restored, as 
done here, or an AF/OPERATOR command (not just a REXX 
instruction) is executed, the batch of commands is not 
sent to Omegamon. As a result, any attempt to use the 
line variables will give undefined results. */

ADDRESS /* Resume original REXX environment */

/* Process line variables in pairs */
status = GLBVGET('OM#') /* Get number of line variables */
DO lnum = 1 TO om#-2 BY 2
  status = GLBVGET('OM'lnum+1)
  line = VALUE('OM'lnum+1)
/* Parse CPU utilization line */
PARSE VAR line,
  7 use.1 +5 unit.1 +3,
  16 use.2 +5 unit.2 +3,
  25 use.3 +5 unit.3 +3,
  34 use.4 +5 unit.4 +3,
  43 use.5 +5 unit.5 +3,
  52 use.6 +5 unit.6 +3,
  61 use.7 +5 unit.7 +3,
  70 use.8 +5 unit.8 +3
Sample REXX Exec 445

/* Parse TSO id's */

status = GLBVGET('OM\'lnum)
line = VALUE('OM\'lnum)

PARSE VAR line,
    7 tsoj.1 +8,
    16 tsoj.2 +8,
    25 tsoj.3 +8,
    34 tsoj.4 +8,
    43 tsoj.5 +8,
    52 tsoj.6 +8,
    61 tsoj.7 +8,
    70 tsoj.8 +8

/* Check time used by each TSO id */

DO i = 1 TO 8
    IF unit.i = "MN" THEN DO
        /* Time returned in minutes and seconds */
        PARSE VAR use.i min ': ' sec
        totsec = min*60+sec
    END
    ELSE IF unit.i = "SEC" THEN DO
        /* Time returned in seconds only */
        totsec = use.i
    END
    ELSE DO
        /* TSO id has not used any time */
        totsec = 0
    END

    IF totsec > thrshold THEN DO
        /* TSO user has used more than threshold time */
        /* Issue AF/OPERATOR command to send message */
        "AFADDR AFHOST",
        "OPER 'SE '""msgtext"",USER=('STRIP'(tsoj.i))'
        count=count+1
    END
END

SAY count "users notified to go home"
RETURN
The OMEGAVIEW database is a table that contains one row for each status item defined to OMEGAVIEW. As status items are created, updated, or destroyed, the rows containing them are inserted, updated, or removed. This appendix describes the columns associated with this table.

After you issue an STGET() function, the data is returned in REXX variables whose names correspond exactly to the column names listed below.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDMADESC</td>
<td>CHAR(50)</td>
<td><strong>PROBLEM DESCRIPTION.</strong> A brief description of the problem that the status reflects and/or the action taken to resolve it. This field may be updated.</td>
</tr>
<tr>
<td>SDMAGENT</td>
<td>CHAR(20)</td>
<td><strong>ACTION DESCRIPTION.</strong> A short string identifying the agent taking action to resolve the problem. This field should contain abbreviated values for the action agent, the action, and the action object whenever appropriate. For example, if the agent is AF/OPERATOR, the action is telephoning, and the object is a person named John Smith, the value of this field might be “AFO phone Smith”. This field may be updated.</td>
</tr>
</tbody>
</table>
| SDMALERT   | CHAR(1)    | **ALERT INDICATOR.** Indicates whether or not the item generates NetView or AF/OPERATOR alerts.  
‘ ’ No alert will be generated.  
1 An alert will be generated.  
This field may be updated. |
<p>| SDMCOMNT   | TEXT(32K)  | <strong>COMMENT.</strong> A text field of 32K. This field allows an action agent to provide a detailed description of the problem and any progress toward its resolution. This field may be updated. |
| SDMDESCR   | VARCHAR(30)| <strong>DESCRIPTION.</strong> A textual status description. This field may be updated. |</p>
<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDMLHV</td>
<td>ARRAY(16)</td>
<td><strong>LONG-TERM HISTORY VALUES.</strong> An array of long-term historical values containing 16 entries that collectively span a 4-hour interval. Each entry represents an aggregation of the values received during its 15-minute period. The most recent value is entry number 1; the oldest value is number 16. The entry names are formed by concatenating the index number to the base variable name, yielding SDMLHV1 through SDMLHV16. Entries for which a value is not yet available will contain the null string. SDMLHV0 contains the number of active entries in the rest of the array. This field cannot be updated.</td>
</tr>
<tr>
<td>SDMNVDLG</td>
<td>VARCHAR(8)</td>
<td><strong>NAVIGATION DIALOG.</strong> The name of the product dialog that manages enhanced zooming. This field may be updated.</td>
</tr>
<tr>
<td>SDMNVPKM</td>
<td>VARCHAR(32K)</td>
<td><strong>NAVIGATION PARAMETER.</strong> The zoom destination to be passed to the dialog named in SDMNVDLG. This field may be updated.</td>
</tr>
<tr>
<td>SDMRULE</td>
<td>TEXT(32K)</td>
<td><strong>DERIVATION RULE.</strong> A text field containing a rule specification that dictates how the status value is to be derived. It will deliver the values when the rule processor interprets it. This field cannot be updated.</td>
</tr>
<tr>
<td>SDMSFXDT</td>
<td>TIME</td>
<td><strong>START FIX TOD.</strong> A time stamp provided by the action agent to indicate when the problem resolution action began. It has the format hh:mm. This field may be updated.</td>
</tr>
<tr>
<td>SDMSHV</td>
<td>ARRAY(8)</td>
<td><strong>SHORT-TERM HISTORY VALUES.</strong> An array of short-term historical values containing 8 entries that collectively span a 15-minute interval. Each entry represents an aggregation of the values received during its 112.5-second period. The most recent value is entry number 1; the oldest value is number 8. The entry names are formed by concatenating the index number to the base variable name, yielding SDMSHV1 through SDMSHV8. Entries for which a value is not yet available will contain the null string. SDMSHV0 contains the number of active entries in the rest of the array. This field cannot be updated.</td>
</tr>
<tr>
<td>SDMSMPDT</td>
<td>TIME STAMP</td>
<td><strong>SAMPLE TOD.</strong> A time stamp identifying when the associated sample of system data was obtained. It has the format mm/dd/yy hh:mm:ss. This field cannot be updated.</td>
</tr>
<tr>
<td>SDMSTNAM</td>
<td>CHAR(20)</td>
<td><strong>STATUS NAME.</strong> A value that uniquely identifies a status item. It may consist of two subfields separated by a period (.). In this case, the subfields may be separately masked in search arguments. This field cannot be updated.</td>
</tr>
<tr>
<td>Column</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| SDMSTYPE   | ENUM      | **USER STATUS TYPE.** The type of a user-defined status item. A null value for this column indicates that the status item was probe-defined as opposed to user-defined. Possible values are:  
|            |           | **1 - ROLLUP** The value is a status level derived by applying a rule to a set of subordinate status items.  
|            |           | **2 - BASIC** The value is an integer.  
|            |           | This field cannot be updated.                                                 |
| SDMTEXT    | VARCHAR(128) | **USER TEXT.** A column provided to you for user data. This field may be updated. |
| SDMTMPLT   | VARCHAR(32) | **TEMPLATE.** A definitional model used for managed objects that inherit their status and situations from their templates. This field can be updated. |
| SDMTRTXT   | VARCHAR(128) | **USER TEXT.** A column provided to you for transient user data. The value will have a duration equal to the shortest of the next emit, or session termination. This field can be updated. |
| SDMTYPE    | ENUM      | **VALUE TYPE.** The type of the status item value. Possible values are:  
|            |           | **1 - DATA** Floating-point numbers (numbers containing decimals) must be converted to integers by multiplying by 1000.  
|            |           | **2 - STATUS** The value is an integer representing a status level.  
|            |           | **3 - ROLLUP** The value is a status level derived by applying a rule to a set of subordinate status items.  
<p>|            |           | This field cannot be updated.                                                 |
| SDMVALUE   | INTEGER   | <strong>CURRENT VALUE.</strong> The most recent status determined for the status item. Valid values are 0, 10, 20, and 30 (corresponding to NONE, OK, WARNING, and CRITICAL); and 5, 15, 25, and 35 (corresponding to NONE, OK, WARNING, and CRITICAL) with an action indicator. This field cannot be updated. |
| SDMXA1     | VARCHAR(20) | <strong>COUNTRY.</strong> If the value does not apply for a particular status item, this column will contain the null string. This field may be updated. |
| SDMXA2     | VARCHAR(20) | <strong>REGION.</strong> If the value does not apply for a particular status item, this column will contain the null string. This field may be updated. |</p>
<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDMXA3</td>
<td>VARCHAR(20)</td>
<td><strong>STATE.</strong> If the value does not apply for a particular status item, this column will contain the null string. This field may be updated.</td>
</tr>
<tr>
<td>SDMXA4</td>
<td>VARCHAR(20)</td>
<td><strong>CITY.</strong> If the value does not apply for a particular status item, this column will contain the null string. This field may be updated.</td>
</tr>
<tr>
<td>SDMXA5</td>
<td>VARCHAR(20)</td>
<td><strong>DATA CENTER.</strong> If the value does not apply for a particular status item, this column will contain the null string. This field may be updated.</td>
</tr>
<tr>
<td>SDMXA6</td>
<td>VARCHAR(20)</td>
<td><strong>SYSTEM ID.</strong> If the value does not apply for a particular status item, this column will contain the null string. This field may be updated.</td>
</tr>
<tr>
<td>SDMXA7</td>
<td>VARCHAR(20)</td>
<td><strong>SUBSYSTEM ID.</strong> If the value does not apply for a particular status item, this column will contain the null string. This field may be updated.</td>
</tr>
<tr>
<td>SDMXA8</td>
<td>VARCHAR(20)</td>
<td><strong>APPLICATION.</strong> If the value does not apply for a particular status item, this column will contain the null string. This field may be updated.</td>
</tr>
<tr>
<td>SDMXFXDT</td>
<td>TIME</td>
<td><strong>ESTIMATED FIX TOD.</strong> A time stamp provided by the action agent to indicate when the problem should be fixed. It has the format hh:mm. This field may be updated.</td>
</tr>
</tbody>
</table>
Introduction

Candle Corporation is committed to producing top-quality software products and services. To assist you with making effective use of our products in your business environment, Candle is also committed to providing easy-to-use, responsive customer support.

Precision, speed, availability, predictability—these terms describe our products and Customer Support services.

Included in this Guide to Candle Customer Support is information about the following:

Base Maintenance Plan .......................................................... 452
- Telephone Support
- eSupport
- Description of Severity Levels
- Service-level objectives
- Recording and monitoring calls for quality purposes
- Customer Support Escalations
- Above and Beyond

Enhanced Support Services ...................................................... 456
- Assigned Support Center Representative (ASCR)
- Maintenance Assessment Services (MAS)
- Multi-Services Manager (MSM)

Customer Support Contact Information ................................. 457
- Link to Worldwide Support Telephone and E-mail information
Base Maintenance Plan

Overview

Candle offers a comprehensive Base Maintenance Plan to ensure that you realize the greatest value possible from your Candle software investments. We have more than 200 technicians providing support worldwide, committed to being responsive and to providing expedient resolutions to support requests. Technicians are available worldwide at all times during the local business day. In the event of an after-hours or weekend emergency, our computerized call management and forwarding system will ensure that a technician responds to Severity One situations within one hour. For customers outside of North America, after-hours and weekend support is provided in English language only by Candle Customer Support technicians located in the United States.

Telephone support

Candle provides consistently reliable levels of service—thanks to our worldwide support network of dedicated experts trained for specific products and operating systems. You will always work with a professional who truly understands your problem.

We use an online interactive problem management system to log and track all customer-reported support requests. We give your support request immediate attention by routing the issue to the appropriate technical resource, regardless of geographic location.

**Level 0 Support** is where your call to Candle Customer Support is first handled. Your support request is recorded in our problem management system, then transferred to the appropriate Level 1 support team. We provide Level 0 manual interaction with our customers because we support more than 170 products. We feel our customers would prefer personal interaction to a complex VRU or IVR selection menu.

**Level 1 Support** is the service provided for initial support requests. Our Level 1 team offers problem determination assistance, problem analysis, problem resolutions, installation assistance, and preventative and corrective service information. They also provide product usage assistance.

**Level 2 Support** is engaged if Level 1 cannot provide a resolution to your problem. Our Level 2 technicians are equipped to analyze and reproduce errors or to determine that an error is not reproducible. Problems that cannot be resolved by Level 2 are escalated to Candle’s Level 3 R&D support team.

**Level 3 Support** is engaged if a problem is identified in Candle product code. At Level 3, efforts are made to provide error correction, circumvention or notification that a correction or circumvention is not available. Level 3 support provides available maintenance modifications.
and maintenance delivery to correct appropriate documentation or product code errors.

**eSupport**

In order to facilitate the support process, Candle also provides **eSupport**, an electronic full-service information and customer support facility, via the World Wide Web at [www.candle.com/support/](http://www.candle.com/support/). **eSupport** allows you to open a new service request and update existing service requests, as well as update information in your customer profile. New and updated service requests are queued to a support technician for immediate action. And we can respond to your request electronically or by telephone—it is your choice.

**eSupport** also contains a continually expanding knowledge base that customers can tap into at any time for self-service access to product and maintenance information.

The Candle Web Site and **eSupport** can be accessed 24 hours a day, 7 days a week by using your authorized Candle user ID and password.

**Description of Candle severity levels**

Responses to customer-reported product issues and usage questions are prioritized within Candle according to Severity Code assignment. Customers set their own Severity Levels when contacting a support center. This ensures that we respond according to your individual business requirements.

<table>
<thead>
<tr>
<th>Severity 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crisis</td>
</tr>
<tr>
<td>A crisis affects your ability to conduct business, and no procedural workaround exists. The system or application may be down.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Severity 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
</tr>
<tr>
<td>A high-impact problem indicates significant business effect to you. The program is usable but severely limited.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Severity 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moderate</td>
</tr>
<tr>
<td>A moderate-impact problem involves partial, non-critical functionality loss or a reasonable workaround to the problem. A “fix” may be provided in a future release.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Severity 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
</tr>
<tr>
<td>A low-impact problem is a “how-to” or an advisory question.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Severity 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enhancement Request</td>
</tr>
<tr>
<td>This is a request for software or documentation enhancement. Our business units review all requests for possible incorporation into a future release of the product.</td>
</tr>
</tbody>
</table>

**Candle has established the following service-level objectives:**

<table>
<thead>
<tr>
<th>Call Status</th>
<th>Severity 1 Goal</th>
<th>Severity 2 Goal</th>
<th>Severity 3 Goal</th>
<th>Severity 4 Goal</th>
<th>Severity 5 Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Call Time to Answer</td>
<td>90% within one minute</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The above information is for guideline purposes only. Candle does not guarantee or warrant the above service levels. This information is valid as of October 1999 and is subject to change without prior notice.
## Base Maintenance Plan

<table>
<thead>
<tr>
<th>Call Status</th>
<th>Severity 1 Goal</th>
<th>Severity 2 Goal</th>
<th>Severity 3 Goal</th>
<th>Severity 4 Goal</th>
<th>Severity 5 Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Level 1 Response</strong></td>
<td>90% within 5 minutes</td>
<td></td>
<td></td>
<td>90% within one hour</td>
<td></td>
</tr>
<tr>
<td>(Normal Business Hours)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Level 2 Response</strong></td>
<td>Warm Transfer</td>
<td>90% within two hours</td>
<td></td>
<td>90% within eight hours</td>
<td></td>
</tr>
<tr>
<td>(Normal Business Hours)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Scheduled follow-up</strong></td>
<td>Hourly or as agreed</td>
<td>Daily or as agreed</td>
<td>Weekly or as agreed</td>
<td></td>
<td>Notification is made when an enhancement is incorporated into a generally available product.</td>
</tr>
<tr>
<td>(status update)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The above information is for guideline purposes only. Candle does not guarantee or warrant the above service levels. This information is valid as of October 1999 and is subject to change without prior notice.
Recording and Monitoring Calls for Quality Purposes

Candle is committed to customer satisfaction. To ensure that our customers receive high levels of service, quality and professionalism, we’ll monitor and possibly record incoming and outgoing Customer Support calls. The information gleaned from these calls will help us serve you better. If you prefer that your telephone call with Candle Customer Support in North America not be monitored or recorded, please advise the representative when you call us at (800) 328-1811 or (310) 535-3636.

Customer Support Escalations

Candle Customer Support is committed to achieving high satisfaction ratings from our customers. However, we realize that you may occasionally have support issues that need to be escalated to Candle management. In those instances, we offer the following simple escalation procedure:

If you experience dissatisfaction with Candle Customer Support at any time, please escalate your concern by calling the Candle support location closest to you. Ask to speak to a Customer Support manager. During standard business hours, a Customer Support manager will be available to talk with you or will return your call. If you elect to hold for a manager, you will be connected with someone as soon as possible. If you wish a return call, please tell the Candle representative coordinating your call when you will be available. After contacting you, the Customer Support manager will develop an action plan to resolve your issue. All escalations or complaints received about support issues are logged and tracked to ensure responsiveness and closure.

Above and Beyond

What differentiates Candle’s support services from our competitors? We go the extra mile by offering the following as part of our Base Maintenance Plan:

- Unlimited multi-language defect, installation and operations support
- eSupport using the World Wide Web
- Regularly scheduled product updates and maintenance provided at no additional charge
- Over 200 specialized technicians providing expert support for your Candle products
Enhanced Support Services

Overview

Our Base Maintenance Plan provides a high level of software support in a packaged offering. However, in addition to this plan, we have additional fee-based support services to meet unique customer needs.

The following are some examples of our added-value support services:

- **Assigned Support Center Representative Services (ASCR)**
  - An assigned focal point for managing support escalation needs
  - Proactive notification of available software fixes
  - Proactive notification of product version updates
  - Weekly conference calls with your ASCR to review active problem records
  - Monthly performance reviews of Candle Customer Support service levels
  - Optional on-site visits (extra charges may apply)

- **Maintenance Assessment Service (MAS)**
  - On-site assessment services
  - Advice about product maintenance and implementation
  - Training your staff to develop efficient and focused procedures to reduce overall cost of ownership of your Candle software products
  - Analysis of your Candle product environment: versions, updates, code correction history, incident history and product configurations
  - Reviews to ensure that purchased Candle products and solutions are used effectively

- **Multi-Services Manager (MSM)**
  Multi-Services Manager provides highly valued services to customers requiring on-site full time expertise to complement their technical resources.
  - Dedicated on-site Candle resource (6 months or one year) at your site to help ensure maximum use and effectiveness of your Candle products
  - Liaison for all Candle product support activities, coordination and assistance with implementation of all product updates and maintenance releases
  - Works with your staff to understand business needs and systems requirements
  - Possesses technical and systems management skills to enhance your staff’s knowledge and expertise
  - Other projects as defined in Statement of Work for MSM services
Customer Support Contact Information

Link to Worldwide Support Telephone and E-mail Information

To contact Customer Support, the current list of telephone numbers and e-mail addresses can be found on the Candle Web site, www.candle.com/support/.

Select Support Contacts from the list on the left of the page.
Symbols

\TYPE command 425
\#TYPE command^ 424
\@TYPE command^ 424
\k command^ 426
\loc command^ 427
\rc command^ 428
\s command^ 429
\see command^ 430
\set command^ 431
\wait command^ 434
\type command^ 433

A
abend recovery
  disabling 190
  enabling 211
accessing a file 308
action
  specifying alternate 366, 387, 392
  specifying in a trap
    specifying alternate 367, 387
activating
  command traps 368
  exception traps 387
  MSMF 316, 317
  MSMF link 325
  MSMF links 320
  traps 411
ACTIVE function 141
active matches, displaying 338
activity summary for AF/OPERATOR 349
adding
  command trap definition 362
  command traps 364
  exception traps 386
text to files 417
  TOD traps 371
  WTO traps 376
  xtype traps 386
ADDRESS command 313
address space
  displaying AF/OPERATOR 342
  ID as trap condition 365, 391
  suspend during trap action 368

Adobe portable document format 15
AF (no operand) 192
AF CMD command 190
AF command 189
AF COMTRACE command 190
AF DOM command 190
AF FREE command 190, 206
AF KILL command 191
AF LOG command 190, 207
AF MATINFO command 190
AF NOCMD command 190
AF NODOM command 190
AF NOLOG command 190, 207
AF NORECOVER command 190
AF NOWTO command 191
AF PROCESS command 191
AF product control commands 215
  AUTOREF
    NOAUTOREF 193
  CMD 190
    NOCMD 197
  COMTRACE 190, 199
  DOM 190
  FREE 190, 206
  issued with no operand 189, 192
  KILL 191
  LOG 190
    NOLOG 207
  MATINFO 190, 208
  NOCMD 190
  NODOM 190
  NOLOG 190
  NOQUIET 190
  NORECOVER 190
  NOWTO 191
  PROCESS 191, 221
  QUIET 190
    NOQUIET 209
  RECOVER 190
    NORECOVER 211
  RELOAD 212
  RELOAD(HOSTNAME) 191
  RXSEARCH 191, 213
  SSIDRAIN 191
  SSIRESET 191
  SSIDRAIN 214
STOP 191
KILL 217
TODSTART 191, 219
WARN 191, 220
WTO 191
NOWTO 222
AF QUIET command 190
AF RECOVER command 190
AF RELOAD(HOSTNAME) command 191
AF RXSEARCH command 191
AF SSIDRAIN command 214
AF SSIRESET command 214
AF STOP command 191
AF TODSTART command 191
AF WARN command 191
AF WTO command 191
AF/OPERATOR
activity summary 349
commands 231
defined match variables 47
displaying address spaces 342
displaying settings 189
displaying status 349
exception traps
adding 386
displaying 353
line-variable facility 60
product control commands
services 24
settings 192
showing files 335
FILENAME 226
AFDISPAT function 141
AFHOST 30
AFOPER
See AO(AFOPER)
AFSHOW function 141
alert, generating 301
ALLOC
AFCALL filename 226
allocating a dataset 190, 206
ampersand
defined 48
number of 48
prefixes 48
variables 48
ANTRQST error codes
on the PPRCCMD function 157
AO command 191
AOACCNT variable 59, 60
AOACCTN variable 57, 58
AOAREAID variable 52
AOARM variable 50
AOASCII variable 50
AOASID variable 52
AOAUTOTK variable 52
AOCACHE variable 50
AOCASE variable 52
AOCLNTID variable 52
AOCMD() 106
AOCMDNO variable 52
AOCMSAD variable 50
AOCMSLN variable 50
AOCONNID variable 52
AOCONNM variable 52
AOCONS 52
AOCONS variable 52
AOCONSID variable 50
AOCONUSE variable 50
AOCOUNT variable 52
AOCPULIM variable 50, 53
AOCPULIM variables, compared 143
AOCPUMOD variable 50
AOCPUSER variable 50
AOCSAFRE variable 50
AOCSASIZ variable 50
AOCTLFLG variable 53
AODESC variable 53
AODOMNO variable 53
AODOMWTN variable 53
AODSPFLG variable 53
AOEBCDIDC variable 50
AOECBFLG variable 53
AF/REMOTE 231
defining a link to
NOHRTBT on LINK DEFINE 276
NOID on LINK DEFINE 276
sending data to
datatype USRDATA 171
AFADDR command
alternate environment 223
applname, LOGON 223
directing commands to an environment 223
LOGON 293
previous environment 223
AFCALL
ALLOC
OPEN 226
defined 226
filename 226
LINKLIST
LPA 226
PARMS 226
parmstring
460   AF/OPERATOR Command Reference Manual Version 340
AOENDCC used in REXX 37
AOENDCC variable 53
AOERRLN variable 53
AOGDATE variable 53
AOHCOPYID variable 53
AONDE variable 57, 58
AOJ3GLBL variable 60
AOJ3GRP variable 60
AOJCLASS variable 53
AOJID variable 53
AOJNAME variable 53
AOJOBID variable 57, 58, 60
AOJTYPE variable 53
AOLASTCC not defined in REXX 37
AOLASTCC variable 53
AOLID variable 53
AOLINKID variable 50
AOLINPREFIX variable 63
AOLINTYP variable 53
AOLSYSNM variable 50
AOMATENT variable 54
AOMATNUM variable 54
AOMAXCC used in REXX 37
AOMAXCC variable 54
AOMAXMAT variable 51
AOMAXTRP variable 51
AOMCLASS variable 60
AOMCSFLG variable 54
AOMLWFGL variable 54
AOMSCFLG variable 54
AOMSGPFX variable 54
AOMSGTYP variable 54
AONONDE variable 57, 58, 60
AONVCLST variable 54
AONVICMD variable 51
AONVICTM variable 51
AOOMWAIT variable 64
AOPNAME variable 57, 58, 60
AOPTYPE variable 54
AOREPID variable 54
AORLS variable 51
AOROOMN variable 57, 58
AOROUT variable 54
AOSID variable 54
AOSMFID variable 51
AOSSTAD variable 51
AOSTATUS variable 54
AOSUBSYS variable 51
AOSYSCNM variable 51
AOSYSN variable 51
AOSYSNUM variable 51
AOTASK variable 51
AOTASKAS variable 51
AOTEST variable 54
AOTIME variable 54
AOTRAP option/TRAP CHANGE WTO command 401
AOTRAP variable 54
AOTSID variable 57, 58
AOTYPE variable 54
AOUSERID variable 54
AOUSTR variable 54
AOVERS variable 54
AOVSADD variable 51
AOVSDEL variable 51
AOVSMOD variable 51
AOVSN variable 51
AOVUTIMO variable 55
AOWTASID variable 56
AOWTNCID variable 56
AOWTCONM variable 56
AOWTDESC variable 56
AOWTJCLS variable 56
AOWTJID variable 56
AOWTJNAM variable 56
AOWTONO variable 55
AOWTROUT variable 56
AOWTSYS variable 56
AOWTTEXT variable 56
AOWTX# variable 56
AOWTXN variable 56
AOXANvariable 55
AUXANSID variable 55
ASCII
application session terminating 292
APPLID, assigning 294
ARM function 142
ASCR
assigned support center representative 456
asterisk caution with pattern variables 89
pattern character 86
to create pattern variables 89
wildcard character 86
asterisk wildcard
JOBCLASS option 397
JOBTYPE option  
SYSID option 398  
REXX exec examples  
TRAP DELETE command 408  
TRAP DISABLE command 410  
SYSID option, TRAP CHANGE xtype command 404  
TRAP DELETE command, using with caution 407  
TRAP ENABLE command 412  
authorized commands 294  
automatic beeper paging 231  

B  
BEEP command 231  
beeper paging 231  
bit position 162  
blanks  
in a full pattern 87  
in pattern characters 86  

C  
cancelling  
hung matches 195  
traps 195  
Candle-supplied REXX functions 24  

case sensitivity 83  
catchup processing 373  
caution, asterisk pattern variables 89  
changing  
command trap definition 362, 390  
default descriptor codes 419, 422  
definitions 402  
traps 396, 403  
xtype traps 403  
character variables 43  
Characters 86  
checking whether a link is active 176  
checkpointing  
TOD traps 373  
variables 68  
CICS command 310  
closing  
logs 190, 207  
SYSOUT datasets 268  
COMADMIN function  
return codes 179  
syntax 178  
COMDATA.1  
response data stored in 173  
comma, in syntax diagrams 14  
command files  
passing execution with AFCALL 226  
command processing  
ending 214  
command trap  
activating/deactivating 368  
adding 364  
deleting 407  
disabling 409  
displaying 353  
enabling 411  
modifying 390  
show matches 338  
testing 368  
commands 223, 258, 272, 273  
CANCELED 195  
COMM 236  
from a REXX exec 187  
from a trap 187  
help for 270  
issued to OMEGAMON 61  
issuing 187  
from a TSO terminal 187  
from an operator console 188  
from the host 187  
LINK 274  
number monitored  
matched  
suppressed  
altered 349  
separating multiple values in 188  
communicating with AF/REMOTE  
datatype USRDATA 171  
NOHRTBT on LINK DEFINE 276  
NOID on LINK DEFINE 276  
communication  
external functions 170  
communication component  
displaying information 237  
releasing suspension 245  
starting 248  
stopping 252  
suspending 255  
communication link 320  
defining 322  
deleting 323  

displaying 324  
starting 325  
stopping 326  
communication trace 199  
compound variable groups 79  
COMSDRCV function  
return codes 175  
syntax 170  
COMTRACE  

Index 463

specifying 199
console
  execution 49
  highest number used 50
  ID 50, 142, 310
  as trap condition 365, 391
  issuing commands from 310
  issuing messages to 418
CONSOLE function 142
console name 365
console status displaying
  subsystem console definitions 346
consoles, issuing messages to 421
contents of storage 160
copying a shared variable value 72
CP variable 55
CPULIM REXX function 143
CTLRGN 272
CTYPE
  specifying 199
customer service
  telephone support 452
customer support
  base maintenance plan 452
  contact information 457
  enhanced support services 456
eSupport 453
  severity levels 453
customizing SHOW TRAPS command display 355
CVGs 79
defining 396
  command trap 364
  exception traps 386
  MSMF links 320, 322
  TOD traps 371
  variables 47, 88
  WTO traps 376
  xtype traps 386
definitions, changing 402
delaying REXX execs 413
deleting
  MSMF link 323
  MSMF links 320
  non-scrollable WTOs 260
  shared variables from a pool 77
  system variables 259
  traps 407
  variables 34
DEQUEUE Function 144
descriptor codes, changing defaults 422
DETAIL
  specifying 238
determining number of ampersands
  using in a REXX exec 48
diagnostic information 199
diagrams of syntax 11
directing commands to alternate environment 223
directing to alternate environment 223
disabled traps 356
  changing 397, 403
  deleting 407
  displaying 353
disabling
  abend recovery 190, 211
  command traps 363, 368
  exception traps 387
  traps 409
displaying 332
AF/OPERATOR address space 342
AF/OPERATOR files 334, 335
AF/OPERATOR settings 189, 192
AF/OPERATOR statistics 349
communication server information 237
file information
  matches 330
  PTFs 330
  VTAM links 330
GROUP 240
help for commands 270
link status information
  link status information 283
logons 330
maintenance level 337

D

data compression option 296
data returned, length 172, 173
data, outbound
  ensuring delivery 289, 290
datasets 206
  allocating 228
  closing
    reopening 268
  deallocating/reallocate 190
date/time functions 94
DAYOFWK 94
ddname, specifying 308
deactivating
  command traps 363, 368
  exception traps 387
  MSMF 316, 318
  traps 407, 409
deallocating a dataset 190, 206
debugging 199
default descriptor codes, changing 419, 422
defaults in syntax diagrams 14
defaults in syntax diagrams 14
Index

MMDDYYYY 94
SECONDS 94
YYYYDD
YYDDD 94
DAYOFWK 94
deleting variables from a pool 77
error condition 93
NetView
   AOCCMD() 106
   NVDAT() 108
OMEGAVIEW
   STCLOSE() 110
   STCREATE() 111
   STGET() 113
   STOPEN() 113
   STSIGN() 114
   STSST() 115
   STSTOD() 116
   STUPDATE() 117
POVI
   POVIDATA() 121
   POVIDLAY() 122
   POVI() 123
   POVILO() 125
   POVILOGOF() 125
   POVILOGON() 126
   POVILOC() 128
   POVIPBPS() 130
   POVI() 130
   POVISAVE() 131
   POVISEND() 132
   POVI() 135
retrieving a value 72
REXX external 24
shared variable 69
system
   ACTIVE 141
   AFDISPAT 141
   AFSHOW 141
   ARM 142
   CONSOLE 142
   CPLIM 143
   COMMAND 148
   GPLO 149
   LOGMSG 149
   MATCH 149
   MSGSLIM 150
   PQUERY 150
   RPID 158
   RTSTORAGE 160
   TESTBIT 162
   TRPQUERY 162
   variable
   GLBVDEL 167
   GLBVGET 165
   GLBVPUT 166
   GLBVUPD 168
   SYMVGET 161
   SYSVDEL 169
   SYSVGET 169
   SYSVPUT 169
   SYSVUPD 169
   verifying a variable value 74
   generating alerts 301
   GLBVDEL function 167
   GLBVGET function 165
   GLBVPUT function 166
   GLBVUPD function 168
global variables 258
defined 43
   displaying 358
   system-defined 47
Gregorian date 94
GROUP
displaying 240
   specifying 237, 245, 248, 252, 255, 276
heartbeat checking
   compared with LNKCHECK function 176
defined 277
determining status of 284
HELP (HE) command 270
displaying all information 270
displaying function information 270
displaying keyword information 270
displaying syntax information 270
hexadecimal variables 43
HHMMSS function 94
host commands, issuing 187
HOSTNAME
   specifying 276
hung matches 195, 339
immediate actions, for command traps 367
IMS command, trapping 364
IMS region
   issuing commands to
   WTOR reply ID 272
IMSCMD
   used to issue commands 95
   used to issue transactions 96
IMSCMD function 95, 96
required configuration 95
In 93
input/output 308
integer variables 43
interpreting variables 48
IRXEXCOM return codes 82
issuing commands 32
from a an operator console 188
from a console 310
from a REXX exec 187
from a trap 187
from a TSO terminal 187
from REXX execs 187
host 187
from REXX 32
messages to a terminal 360
messages to console 418
subsystem commands
MVS commands 310
to IMS 272
to JES3 273
to VM host 258
issuing to terminals
JES3
using TPUT command 360

J
JES command 310
JES2 interface 57, 58
AOACCTN variable 57, 58
AOINDEV variable 57, 58
AOJCLASS variable 57, 58
AOJOBID variable 57, 58
AONONDE variable 57, 58
AOPNAME variable 57, 58
AOROOMN variable 57, 58
AOTMCLASS variable 57, 58
AOTSUID variable 57, 58
JES3
issuing commands to 273
trapping 364
Version 2.2.1 273
JES3 interface 59, 60
AOACCTN variable 59, 60
AOJ3GLBL variable 59, 60
AOJ3GRP variable 59, 60
AOJCLASS variable 59, 60
AOJOBID variable 59, 60
AOMCLASS variable 59, 60
AONONDE variable 59, 60
AOPNAME variable 59, 60
jobclass as trap condition 365, 391
jobname
as trap condition 365, 391
JOBSTEP function 148
jobtype as trap condition 366, 392
JOURNAL
option on TRAP command 368, 383, 388, 393, 401, 405
Julian date 94

K
K command 426

L
LEAPYEAR function 94
length
of an PRODNAME line 173
of response data 172, 173
line variables structure 63
line-count variables 63
lines, reading 328
line-variable facility 60
LINE 63
LINK DEFINE command
NOHRTBT parameter 276
NOID parameter 276
return codes 279
LINK DELETE command 280, 282
LINK DISPLAY command 283, 285
LINK STOP command 289, 291
LINKID
specifying 199, 275, 280, 283, 286, 289
linkid
displaying 237
reusing 281
linking AF/OPERATOR to NetView 306
links
defining 275, 322
deleting 280, 323
displaying 283, 324
status 336
starting 286, 325
stopping 326
terminating 289
listing REXX execs 267
LLUNAME
specifying 248
LNKCHECK function 176, 177
LOC command 427
local variables 43, 358
LOG function 149
LOG, in AF command 190
logging
activating AF/OPERATOR facility 190
functions 149
messages 207
onto OMEGAMON 294
onto OMEGAVIEW 295
onto VTAM 293
trapped message 368, 387
LOGMSG function 149
LOGOFF command
AF/OPERATOR 292
POVI 437
LOGON 293
LOGON command 293, 294
LOGONS, display 330
logs
closing 190, 207
writing messages to 190, 207
loops, preventing 384

M
maintenance assessment service
MAS 456
maintenance level, showing 337
MAS
maintenance assessment service 456
master console ID 142
MATCH function 149
match variables 43
AOACNT variable 59, 60
AOACCTN variable 57, 58
AOAREAID variable 52
AOASID variable 52
AOAUTOTK variable 52
AOCLNTEX variable 52
AOCLNTID variable 52
AOCMDNO variable 52
AOCOMMID variable 52
AOCOMNN variable 52
AOCOCONS variable 52
AOCOUNT variable 52
AOCPU1IM variable 53
AOCRTLFLG variable 53
AOEDesc variable 53
AODOMNO variable 53
AODOMWATN variable 53
AODSFLFLG variable 53
AOECBFULG variable 53
AOENDCC variable 53
AOERRLNL variable 53
AOGDATE variable 53
AOHCPYID variable 53
AOINDEV variable 57, 58
AOJ3GLBL variable 59, 60
AOJ3GRP variable 59, 60
AOJCLAS variable 53
AOJCLASS variable 57, 58, 59, 60
AOjid variable 53
AOJNAME variable 53
AOJOjid variable 57, 58, 59, 60
AOJTYPE variable 53
AOLASTCC variable 53
AOLID variable 53
AOLINTYP variable 53
AOMATENT variable 54
AOMATNUM variable 54
AOMAXCC variable 54
AOMCLASS variable 57, 58, 59, 60
AOMCSFLG variable 54
AOMLWFLG variable 54
AOMSCFLG variable 54
AOMSGPFX variable 54
AOMSGTYP variable 54
AONONDE variable 57, 58, 59, 60
AONVCLST variable 54
AOPNAME variable 57, 58, 59, 60
AOPTYPE variable 54
AOREPID variable 54
AOROOMN variable 57, 58
AOROUT variable 54
AOSID variable 54
AOSTATUS variable 54
AOTEXT variable 54
AOTIME variable 54
AOTRAP variable 54
AOTSUID variable 57, 58
AOTTYPE variable 54
AUSERID variable 54
AUSR variable 54
AOVERS variable 54
AOWTASID variable 56
AOWTCLNID variable 56
AOWTDESC variable 56
AOWTJLCL variable 56
AOWTJID variable 56
AOWTIJNAM variable 56
AOWTONO variable 55
AOWTROUT variable 56
AOWTSTR variable 56
AOWTTEXT variable 56
AOWT# variable 56
AOWTXN variable 56
AOXANSID variable 55
CP variable 55
JES2 interface 57, 58
JES3 interface 59, 60
OVUTIMO variable 55
system-defined 47
WAIT command 56
matched
suppressed 349
matches
CMD traps, displaying 338
displaying 330
displaying active 338
enabling monitoring for 197
ending monitoring for 222
exception traps, displaying 338
hung 195
titling 339
message for completion 190
number, displaying 338
starting monitoring for 190, 222
stopping monitoring 190, 197
terminating 190
TOD, displaying 338
WTO, displaying 338
MEMBER
specifying 200, 237, 245, 249, 252, 255, 276
messages 360
ALL FILES CLOSED
FILE CLOSE ERROR 416
completed match 190
default for SHOW TRAPS 353
deleting WTO messages 260
for completed matches
showing CPU time for match 208
issuing to consoles 298, 418, 421
routing to multiple consoles 422
sending to TSO user 360
stopping suppression 190, 210
suppressing 190, 209, 366, 367, 392
TRAP DISABLE command 410
write to operator 298
MLWTO command 298
MMDDYY function 94
MODENAME
specifying 276
modifying
command traps 390
definitions 402
traps 396, 403
WTO traps 396
xtype traps 403
MONITOR command 438
monitoring for matches 190, 191, 197, 222
MSGLIM function 150
MSGLIM parameter 353
MSM
multi-services manager 456
MSMF
activating 310
start link 325
terminating 316
deactivating
terminating
stopping
force 318
define link 322
delete link 323
display link 324
stop link 326
multi-line information, LINE facility 61
multiple values, separating 188
multi-services manager
MSM 456
MVS
issuing VM commands 258
MVS operator command 310

N
name pattern
cautions using 92
defined 91
named variables 167
naming conventions, pattern matching 85
naming variables 46
NetView
alerts
via the Program-to-Program Interface 301
functions
AOCMD() 106
NVIDATA() 108
showing communication status 304
starting communication with 306
stopping communication with 307
NODOM command 190
NOHRTBT
specifying 276
NOID
specifying 276
NOJOURNAL
option on TRAP command 368, 383, 388, 393, 401, 405
non-interactive transactions
using IMSCMD to issue 96
non-scrollable WTO, change to scrollable 260
Index

number monitored altered 349
NVALERT commandl 301
NVIDATA() 108
NVIDATA() function 108
NVISHOW command 304
NVISTART command 306
NVISTOP command 307

OG/MVS
displaying status 349
OMEGACENTER Gateway for OS/2 231
OMEGAMON
exception messages 386
exception traps 338, 411
exceptions 294
internal security
external security
password
userid 294
interval 64
LINE facility
line-count variable 63
logging onto 293, 296
sample exec to interface 443
showing connection to NetView 304
starting communication to NetView 306
stopping communication to NetView 307
structure of LINE variables 63
wait variable 64
OMEGAMON exception traps
changing type 403
deleting 407
OMEGAMON external security 295
OMEGAVIEW
database of status items 447
functions
STCLOSE() 110
STCREATE() 111
STGET() 113
STOPEN() 113
STSSIGNAL() 114
STSTO() 115
STSTOD() 116
STUPDATE() 117
OMEGAVIEW Interface
return codes 118
REXX functions 109
OPEN
AFCALL filename 226
OPEN command 308
opening
files 308
SYSOUT datasets 268
OPER command 310
specifying response text variables 311
specifying target console 310
operator commands
issuing 310
monitoring for matches 190
trapping 364
operator console, issuing commands from 188
Operator Facility, Programmerless Open VTAM Interface 435
operator reply to message 421
PARMS 226
PASSWORD parameter for OMEGAMON 294
pattern characters
adjacent to 88, 89
pattern matching 86
pattern variables 86
pound sign 86
question mark 86
pattern matching 88
asterisk to match text 86
blanks in pattern 87
case sensitivity 83
full pattern 85
general 83–92
name pattern matching 84
pattern variables 88
pattern variables 83, 88
adjacent to character 88, 89
asterisk
question mark
pound sign 88
asterisk to match text 86
blanks in pattern 87
cautions with asterisk 89
creating with asterisks 89
defined 88
defining a pattern variable 47
full pattern matching 85
name pattern matching 84
pattern characters 86
termination characters 88
using WAIT command 415
patterns
creating 90
developing from software documentation 90
example 88
wildcard 86
PEER command 316
   See MSMF interface 316
PEER START command 317
PEER STOP command 318
PEERLINK command 320
PEERLINK DEFINE command 322
PEERLINK DELETE command 323
PEERLINK DISPLAY command 324
PEERLINK START command 325
PEERLINK STOP command 326
period pattern character 86
PLUNAME
   specifying 275
PLXQUERY function 150
PORTNUMBER 237
   specifying 237, 246, 249, 252, 256, 276
pound sign, pattern character 86
POVI
   environment commands 423
   functions 121
      POVIDATA() 121
      POVIDLAY() 122
      POVIFIND() 123
      POVILGOF() 125
      POVLGON() 126
      POVILOC() 128
      POVIPBS() 130
      POVIRSTR() 131
      POVISAVE() 131
      POVISEND() 132
      POVIVSET() 135
   operator commands 435
      POVIDLAY() function 122
      POVIFIND() function 123
      POVILGOF() function 125
      POVLGON() function 126
      POVILOC() function 128
      POVIPBS() function 130
      POVIRSTR() function 130
      POVISAVE() function 131
      POVISEND() function 132
      POVIVSET() function 135
      PPRCCMD function 157
         ANTRQST error codes 157
            syntax 156
   printing problems 15
   problem determination 199
   process mode
      AF PROCESS 221
         description 191, 221
   Programmerless Open VTAM Interface
      environment commands 423
      TYPE 425
         TYPE 424
         K 426
         LOC 427
         RC 428
         S 429
         SEE 430
         SET 431
         STATUS 432
         WAIT 434
         TYPE 433
   executing commands 435
   logging onto 293
   operator commands
      ECHO 436
         executing 435
      LOGOFF 437
      MONITOR 438
      REFRESH 439
      VSHOW 440
   Operator Facility 435
   VTAM operator commands 435
   programming scenario 443
   Program-to-Program Interface
      sending NetView alerts using 301
   PTFLIB dataset
      changing default descriptor code 422
         defining at startup 419
      punctuation, in syntax diagrams 14
      question mark, as pattern character 86
      quotes, using in REXX execs 32
      R
      RC command 428
      reading files 328
      RECOVERY
         specifying 276
      REDIRECT function 136
      REFRESH command 439
      REPID function 158
      REPINFO function 158
      replies, suppressing 402, 422
      reply ID 158
      reply length 172, 173
      rerouting messages to target system 384
      reserved words 46
      response length 172, 173
      RETAIN*VIEW (AF/OPERATOR)
         recording matches for 368, 388
         retrieving
Index
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHOW ENQUEUE command</td>
<td>332</td>
</tr>
<tr>
<td>SHOW FILES command</td>
<td>334</td>
</tr>
<tr>
<td>SHOW FORMAT</td>
<td>355</td>
</tr>
<tr>
<td>SHOW HOSTNAME command</td>
<td>335</td>
</tr>
<tr>
<td>SHOW LOGONS command</td>
<td>336</td>
</tr>
<tr>
<td>SHOW MAINT command</td>
<td>337</td>
</tr>
<tr>
<td>SHOW MATCHES command</td>
<td>338</td>
</tr>
<tr>
<td>SHOW OG command</td>
<td>342</td>
</tr>
<tr>
<td>SHOW RSVS command</td>
<td>343</td>
</tr>
<tr>
<td>SHOW SSCONS command</td>
<td>346</td>
</tr>
<tr>
<td>SHOW SSCTS command</td>
<td>347</td>
</tr>
<tr>
<td>SHOW STATS command</td>
<td>349</td>
</tr>
<tr>
<td>SHOW SVCS command</td>
<td>351</td>
</tr>
<tr>
<td>SHOW TITLE</td>
<td>355</td>
</tr>
<tr>
<td>SHOW TRAPS command</td>
<td>353</td>
</tr>
<tr>
<td>SHOW TRAPS TITLE FORMAT command</td>
<td>355</td>
</tr>
<tr>
<td>SHOW VARS command</td>
<td>358</td>
</tr>
<tr>
<td>SSCONS, displaying</td>
<td>330</td>
</tr>
<tr>
<td>SSCTS, displaying</td>
<td>330</td>
</tr>
<tr>
<td>SSIDRAIN command</td>
<td>191</td>
</tr>
<tr>
<td>SSRESET command</td>
<td>191</td>
</tr>
<tr>
<td>START command</td>
<td>191</td>
</tr>
<tr>
<td>STOP command</td>
<td>191</td>
</tr>
<tr>
<td>VTAM application session</td>
<td>293</td>
</tr>
<tr>
<td>startup</td>
<td></td>
</tr>
<tr>
<td>LOGON</td>
<td>296</td>
</tr>
<tr>
<td>OMEGAMON logon</td>
<td>296</td>
</tr>
<tr>
<td>statistics, displaying AF/OPERATOR</td>
<td>349</td>
</tr>
<tr>
<td>status column definitions</td>
<td>447</td>
</tr>
<tr>
<td>STATUS command</td>
<td>432</td>
</tr>
<tr>
<td>status information, displaying communication services</td>
<td>237</td>
</tr>
<tr>
<td>link status</td>
<td>283</td>
</tr>
<tr>
<td>status, displaying variables</td>
<td>358</td>
</tr>
<tr>
<td>STCLOSE() function</td>
<td>110</td>
</tr>
<tr>
<td>STCREATE() function</td>
<td>111</td>
</tr>
<tr>
<td>stem variables</td>
<td>79</td>
</tr>
<tr>
<td>STGET() function</td>
<td>113</td>
</tr>
<tr>
<td>STOPEN() function</td>
<td>113</td>
</tr>
<tr>
<td>stopping AF/OPERATOR with AF STOP command</td>
<td>191</td>
</tr>
<tr>
<td>AF/OPERATOR with the AF KILL command</td>
<td>191</td>
</tr>
<tr>
<td>execution of REXX execs</td>
<td>413</td>
</tr>
<tr>
<td>links</td>
<td>289</td>
</tr>
<tr>
<td>message suppression</td>
<td>190, 210</td>
</tr>
<tr>
<td>monitoring for matches</td>
<td>190, 197</td>
</tr>
<tr>
<td>MSMF</td>
<td>316, 318</td>
</tr>
<tr>
<td>MSMF link</td>
<td>326</td>
</tr>
<tr>
<td>stopping PRODNAME with the AF KILL command</td>
<td></td>
</tr>
<tr>
<td>storing retrieved data</td>
<td>328</td>
</tr>
<tr>
<td>REXX execs</td>
<td>266</td>
</tr>
<tr>
<td>STSIGNAL() function</td>
<td>114</td>
</tr>
<tr>
<td>STSSTO() function</td>
<td>115</td>
</tr>
<tr>
<td>STSSTOD() function</td>
<td>116</td>
</tr>
<tr>
<td>STUPDATE() function</td>
<td>117</td>
</tr>
<tr>
<td>STYPE specifying</td>
<td>200</td>
</tr>
<tr>
<td>subsystem command</td>
<td>310</td>
</tr>
<tr>
<td>subsystems</td>
<td>330</td>
</tr>
<tr>
<td>communications table display type</td>
<td>347</td>
</tr>
<tr>
<td>console display</td>
<td>346</td>
</tr>
<tr>
<td>display console definitions</td>
<td>330</td>
</tr>
<tr>
<td>multiple address space</td>
<td>342</td>
</tr>
<tr>
<td>RSVS</td>
<td>343</td>
</tr>
<tr>
<td>suppressing log entries</td>
<td>422</td>
</tr>
<tr>
<td>message</td>
<td>209</td>
</tr>
<tr>
<td>messages</td>
<td>190</td>
</tr>
<tr>
<td>replies</td>
<td>422</td>
</tr>
<tr>
<td>trapped message</td>
<td>366, 367, 392</td>
</tr>
<tr>
<td>suspending</td>
<td></td>
</tr>
</tbody>
</table>

VTAM application session 293
startup
LOGON 296
OMEGAMON logon 296
statistics, displaying AF/OPERATOR 349
status column definitions 447
STATUS command 432
status information, displaying communication services 237
link status 283
status, displaying variables 358
STCLOSE() function 110
STCREATE() function 111
stem variables 79
STGET() function 113
STOPEN() function 113
stopping AF/OPERATOR with AF STOP command 191
AF/OPERATOR with the AF KILL command 191
execution of REXX execs 413
links 289
message suppression 190, 210
monitoring for matches 190, 197
MSMF 316, 318
MSMF link 326
stopping PRODNAME with the AF KILL command
storing retrieved data 328
REXX execs 266
STSIGNAL() function 114
STSTOS() function 115
STSTOD() function 116
STUPDATE() function 117
STYPE specifying 200
subsystem command 310
subsystems 330
communications table display type 347
console display 346
display console definitions 330
multiple address space 342
RSVS 343
suppressing log entries 422
message 209
messages 190
replies 422
trapped message 366, 367, 392
suspending
communication component 255
releasing suspension 245
suspending address space during trap action 368
SVC
  displaying 330
  entries/display
    table 351
  trapping 364
symbolic substitution
  console execution 49
  determining number of ampersands 48
  obtaining the value of a variable 48
  of variables 48
  resolving a variable 48
  write a symbolic variable 48
symbolic variables 45
SYMDEST
  specifying 248, 275
SYMVGET function 161
synchronizing commands 415
syntax
  conventions 11
  diagram 11
SYSLOG, SUPPRESSED entries 402, 422
sysout datasets, allocating/deallocating 206
system 223
  functions 141
    ACTIVE 141
    AF/DISPAT 141
    AF/HOW 141
    ARM 142
    CONSOLE 142
    CPULIM 143
    JOBSTEP 148
    LOG 149
    LOGMSG 149
    MATCH 149
    MSG/LIM 150
    PLXQUERY 150
    REPID 158
    RSTORAGE 160
    TESTBIT 162
    TRPQUERY 162
variables 44, 161, 169
  ADDRESS 313
  delete non-AF 259
  displaying 358
  OPER RESP 313
system dump 217
system variables 50
  AF/OPERATOR-supplied 44
  AO/ARM variable 50
  AO/ASCII variable 50
  AOCACHE variable 50
  AO/CMSAD variable 50
  AO/CMSLN variable 50
  AO/CONSIDI variable 50
  AO/CONUS variable 50
  AO/CPU/variable 50
  AO/CPU/C/variable 50
  AO/CPU/S variable 50
  AO/CPU/G variable 50
  AO/CPU/variable 50
  AO/CSAFRE variable 50
  AO/CSASIZ variable 50
  AO/ECDC variable 50
  AO/LINK variable 50
  AOSYSNM variable 50
  A/O/MAXMAT variable 51
  A/O/MAXTRP variable 51
  A/O/MSGLIM variable 51
  A/O/NICMD variable 51
  A/O/NVICTM variable 51
  AORLS variable 51
  AOSMFID variable 51
  AOS/CTAD variable 51
  AOS/USYS variable 51
  AOS/SCNM variable 51
  AOS/SCON variable 51
  AOS/SID variable 51
  AOS/YSNM variable 51
  AOS/TASK variable 51
  AOS/TASKAS variable 51
  AOS/VSADD variable 51
  AOS/VSDEL variable 51
  AOS/SMOD variable 51
  AOS/N variable 51
  defined 44
  deleting 259
system-defined variables 47
system-supplied match variables 52
system-supplied variables 50
SYSVDEL function 169
SYSVGET function 169
SYSVPUT function 169
SYSVUPD function 169

T
  target system, sending messages to 384
  target, string 83
task type as trap condition 366, 392
TCP/IP
  displaying 240
telephone support
  customer service 452
terminating
  matches 190
MSMF 318
MSMF link 326
MSMF links 320
traps 195
VTAM application session 292
termination character
pattern variables
variable name 88
TESTBIT function 162
testing
AF WARN command 220
command traps 368
exception traps 388
text string 84
text, adding to files 417
time 94
time-of-day traps
adding 371
checkpointing 373
deleting 407
disabling 409
displaying 353
enabling 411
show matches 338
starting 191, 219
timeout, setting values for 116
TITLE FORMAT, for SHOW TRAPS command 355
title variables
for SHOW TRAPS 356
list of 340
titling
matches display 339
SHOW TRAPS command 355
to a shared variable 69
TPNAME
displaying 239
specifying 200, 237, 245, 252, 255, 275
TPUT command 360
trace facility
AF COMTRACE command 199
displaying results 201
trace information 199
trace results 201
transactions
using IMSCMD to issue 96
TRAP (TR) command 362
TRAP ADD CMD command 364
TRAP ADD command 49
TRAP ADD TOD command 371
TRAP ADD WTO command 376
TRAP ADD xtype command 386, 403
TRAP CHANGE CMD command 390
TRAP CHANGE command, using ampersands 49
TRAP CHANGE pattern variable 88
TRAP CHANGE WTO command 396
TRAP CHANGE xtype command 403
trap definitions, displaying 353
TRAP DELETE command 407
TRAP DISABLE command 409
TRAP ENABLE command 411
message 410
trap, components 25
trapping
AF/OPERATOR commands
SUPPRESS TSO LOGON command 367
AF/OPERATOR exceptions 386
operator commands 364
TRAPS
customizing display 355
displaying 330
ENABLED message 410
enabling 356
traps
changing 396, 403
customizing display 355
deactivating 409
deleting definitions 407
displaying 353
enabling 411
issuing commands from 187
monitoring for matches 197
sending 189
temporary wait 415
terminating 190, 195
triggering conditions 364
command traps 364
exception traps 386
troubleshooting 199
TRPQUERY function 162
TSO LOGON command, suppressing 367
TSO terminal
issuing commands from 187
issuing messages to 360
TYPE
specifying 237, 245, 252, 255, 275, 280, 283, 286, 289
TYPE command 424, 425, 433
UNALC 268
underlining in syntax diagrams 14
undocumented variables 358
user-defined variables 47
USERID 295
USERID parameter for OMEGAMON 294
using stem variables as shared variables 79

variables 108
AF/OPERATOR 44
AFADDR command
address to alternate environment 223
AOOMWAIT 64
classifying 43
defining 88
defining in full patterns 47
with a match 47
displaying 330, 343, 358
status values 358
duplicate names 44
external functions
format 165
format 340, 357
functions
GLBVDEL 167
GLBVGET 165
GLBVPUT 166
GLBVUPD 168
SYMVGET 161
SYSVDEL 169
SYSVGGET 169
SYSVPUT 169
SYSVIPD 169
general 41, 52
global 43, 47
hexadecimal 43
integer 43
internal 45
interpreting 48
length 43
line-count LINE 63
local 43
match 43, 52, 56
JES2 interface 57
JES3 interface 59
named 167
naming 46
pattern 47
pools 43
REXX shared 44, 65
rules 46
scope of 43
shared functions 69
stem 79
storing 328
structure of LINE
LINE 63
symbolic substitution 48
system 44, 50
system-defined 47
system-supplied 50
termination 88
title 340
for show matches
format
for show matches 339
for SHOW TRAPS/MATCHES
format
for SHOW TRAPS/MATCHES 35
6
types 43
undocumented 358
user-defined 47
variables 63
WAIT command 56
verifying a variable value 74
VM
operator command, issuing
console 258
VSHOW command 440
VTAM
application
initiating a session 293
LOGON appltype 293
terminating a session 292
links
displaying 336
operator commands 435
session 293
VTAM application session
linking to
initiating
starting
beginning
naming 293
multiple 296
terminating
ending 292

W

WAIT command 413, 434
address space continuation 414
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>address space suspension</td>
<td>414</td>
</tr>
<tr>
<td>displaying messages</td>
<td>414</td>
</tr>
<tr>
<td>setting wait period</td>
<td>414</td>
</tr>
<tr>
<td>suppressing messages</td>
<td>414</td>
</tr>
<tr>
<td>WAIT command match variables</td>
<td>56</td>
</tr>
<tr>
<td>WAIT option, TRAP CHANGE WTO command</td>
<td>401</td>
</tr>
<tr>
<td>WAIT parameter, cautions for using</td>
<td>385</td>
</tr>
<tr>
<td>wait traps, temporary</td>
<td>415</td>
</tr>
<tr>
<td>waiting for executing command traps</td>
<td>368</td>
</tr>
<tr>
<td>warn mode</td>
<td>191, 220</td>
</tr>
<tr>
<td>wildcard characters (*)</td>
<td></td>
</tr>
<tr>
<td>JOBCLASS option</td>
<td>397</td>
</tr>
<tr>
<td>JOBTYPE option</td>
<td></td>
</tr>
<tr>
<td>SYSID option</td>
<td>398</td>
</tr>
<tr>
<td>REXX exec examples</td>
<td></td>
</tr>
<tr>
<td>TRAP DELETE command</td>
<td>408</td>
</tr>
<tr>
<td>TRAP DISABLE command</td>
<td>410</td>
</tr>
<tr>
<td>SYSID option</td>
<td>404</td>
</tr>
<tr>
<td>TRAP DELETE command, using with caution</td>
<td>407</td>
</tr>
<tr>
<td>TRAP ENABLE command</td>
<td>412</td>
</tr>
<tr>
<td>wildcard patterns</td>
<td></td>
</tr>
<tr>
<td>asterisk</td>
<td>86</td>
</tr>
<tr>
<td>See also asterisk</td>
<td>86</td>
</tr>
<tr>
<td>write to operator message</td>
<td>298</td>
</tr>
<tr>
<td>writing messages to logs</td>
<td>190, 207</td>
</tr>
<tr>
<td>writing to files</td>
<td>417</td>
</tr>
<tr>
<td>WTO</td>
<td></td>
</tr>
<tr>
<td>changing non-scrollable</td>
<td></td>
</tr>
<tr>
<td>deleting non-scrollable</td>
<td>260</td>
</tr>
<tr>
<td>defined</td>
<td>222</td>
</tr>
<tr>
<td>WTO number monitored</td>
<td>349</td>
</tr>
<tr>
<td>WTO command</td>
<td></td>
</tr>
<tr>
<td>naming variables</td>
<td>299</td>
</tr>
<tr>
<td>setting descriptor codes</td>
<td>298</td>
</tr>
<tr>
<td>setting message id</td>
<td>298</td>
</tr>
<tr>
<td>setting route codes</td>
<td>298</td>
</tr>
<tr>
<td>WTO processing</td>
<td></td>
</tr>
<tr>
<td>ending</td>
<td>214</td>
</tr>
<tr>
<td>starting</td>
<td>214</td>
</tr>
<tr>
<td>WTO traps</td>
<td>396</td>
</tr>
<tr>
<td>adding</td>
<td>376</td>
</tr>
<tr>
<td>deleting</td>
<td>407</td>
</tr>
<tr>
<td>disabling</td>
<td></td>
</tr>
<tr>
<td>OMEGAMON exception traps</td>
<td></td>
</tr>
<tr>
<td>disabling</td>
<td>409</td>
</tr>
<tr>
<td>enabling</td>
<td>411</td>
</tr>
<tr>
<td>format variables</td>
<td>340</td>
</tr>
<tr>
<td>modifying</td>
<td>396</td>
</tr>
<tr>
<td>show matches</td>
<td>338</td>
</tr>
<tr>
<td>WTO traps, displaying</td>
<td>353</td>
</tr>
<tr>
<td>WTOR command</td>
<td>421</td>
</tr>
<tr>
<td>reply ID</td>
<td>158, 272</td>
</tr>
<tr>
<td>ROUT (9)</td>
<td></td>
</tr>
<tr>
<td>suppressing replies</td>
<td>422</td>
</tr>
<tr>
<td>SUPPRESSING the reply</td>
<td></td>
</tr>
<tr>
<td>rout (9)</td>
<td>402</td>
</tr>
<tr>
<td>WTOR, sending to remote consoles</td>
<td>231</td>
</tr>
<tr>
<td>XCFID</td>
<td></td>
</tr>
<tr>
<td>specifying</td>
<td>200</td>
</tr>
<tr>
<td>xtype traps, interval</td>
<td>294</td>
</tr>
<tr>
<td>YYDDD function</td>
<td>94</td>
</tr>
</tbody>
</table>