Contents

List of Figures .............................................................................. 7
Preface .......................................................................................... 9
Adobe Portable Document Format .................................................. 11

What’s New in AF/OPERATOR Version 340 ...................................... 13

Chapter 1. Getting Started .......................................................... 17
AF/OPERATOR and Your Enterprise ................................................. 18
What AF/OPERATOR Can Do for You ............................................... 20
Multiple AF/OPERATOR Address Spaces ....................................... 21
Using REXX Shared Variables in AF/OPERATOR .............................. 26
Extended MCS Console Support ..................................................... 27
Using EMCS Consoles Services ...................................................... 28

Chapter 2. Starting and Stopping AF/OPERATOR .............................. 29
Starting AF/OPERATOR ................................................................. 30
Stopping AF/OPERATOR .............................................................. 33

Chapter 3. Trapping System Events ............................................... 35
Trap Basics .................................................................................. 36
How Traps Work .......................................................................... 39
Setting Traps ............................................................................... 41
Command (CMD) Traps ............................................................... 45
Time of Day (TOD) traps ............................................................... 47
Console Message (WTO) Traps ...................................................... 49
OMEGAMON Exception (xtype) Traps .......................................... 51
JES3 Traps ................................................................................. 54
Pattern Matching in Trap Statements ............................................. 55
Name Patterns ............................................................................. 59
Pattern Variables .......................................................................... 61
Variable Substitution ..................................................................... 63
Specifying Trap Responses ............................................................ 65
Enabling and Disabling Traps ........................................................ 69
Modifying and Deleting Traps ......................................................... 70
Getting Information about Traps .................................................... 71
Journaling Trap Data Using the Subsystem Logging Facility ............ 73
Cross-Reference of Traps and Command Files ................................. 74
Report on Command Files Performing Trap Management .............. 75
Report on Traps Manipulated by Command Files .............................. 76
Report on Traps Executing Command Files ..................................... 77
Chapter 8. Communicating with NetView
What You Need to Know
Using Communication Options
Controlling the NetView Interface
Starting and Stopping the AF/OPERATOR NetView Interface
Displaying AF/OPERATOR NetView Interface Status and Matches
Invoking AF/OPERATOR Commands from NetView
Retrieving Data from AF/OPERATOR or Other Candle Products
Trapping on a NetView Alert
Trapping NetView Console Commands
Trapping NetView Console VTAM Messages
Generating NetView Alerts from AF/OPERATOR
Invoking NetView REXX Execs or Commands from AF/OPERATOR

Chapter 9. Communicating with OMEGAVIEW
What You Need to Know
AF/OPERATOR REXX Functions
Trapping on Status Item Changes
Trapping on Problem Management Alerts
REXX Line Variables
OMEGAVIEW Return Codes
AF/OPERATOR Return Codes
Establishing a Session
Creating Status Items
Dynamic Rollup Status Items
Opening a Status Item
Updating Status Item Information
Assigning a Status Value
Returning OMEGAVIEW Column Information
Automated Session Control

Chapter 10. Monitoring CICS Transient Data Queues
What You Need to Know
Handling CICS Messages
Trapping CICS Messages in Transient Data Queues
Trapping CICS Terminal Deletion Messages

Chapter 11. Monitoring and Controlling IMS Regions
What You Need to Know
Summary of IMS Interface Commands
Issuing Commands to IMS Regions
<table>
<thead>
<tr>
<th>FIGURE</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIGURE 1</td>
<td>AF/OPERATOR and Other Products</td>
<td>18</td>
</tr>
<tr>
<td>FIGURE 2</td>
<td>Multiple AF/OPERATOR Address Spaces on an MVS System</td>
<td>22</td>
</tr>
<tr>
<td>FIGURE 3</td>
<td>Multiple AF/OPERATOR Address Spaces on Multiple MVS Systems</td>
<td>24</td>
</tr>
<tr>
<td>FIGURE 4</td>
<td>Function of a Trap</td>
<td>39</td>
</tr>
<tr>
<td>FIGURE 5</td>
<td>TRAP(CLEANUP) Definition</td>
<td>42</td>
</tr>
<tr>
<td>FIGURE 6</td>
<td>General TRAP ADD Definition</td>
<td>43</td>
</tr>
<tr>
<td>FIGURE 7</td>
<td>Report AOPRPT01</td>
<td>75</td>
</tr>
<tr>
<td>FIGURE 8</td>
<td>Report AOPRPT02</td>
<td>76</td>
</tr>
<tr>
<td>FIGURE 9</td>
<td>Report AOPRPT03</td>
<td>77</td>
</tr>
<tr>
<td>FIGURE 10</td>
<td>Report AOPRPT04</td>
<td>78</td>
</tr>
<tr>
<td>FIGURE 11</td>
<td>Report AOPRPT05</td>
<td>79</td>
</tr>
<tr>
<td>FIGURE 12</td>
<td>Cross-Reference Facility Error Report</td>
<td>80</td>
</tr>
<tr>
<td>FIGURE 13</td>
<td>JCL to be Edited to Run the Cross-Reference Reports</td>
<td>81</td>
</tr>
<tr>
<td>FIGURE 14</td>
<td>Rexx Error Recovery Example</td>
<td>114</td>
</tr>
<tr>
<td>FIGURE 15</td>
<td>Programmerless Open VTAM Interface Sample REXX Exec</td>
<td>161</td>
</tr>
<tr>
<td>FIGURE 16</td>
<td>NetView Interface Communications Scenario</td>
<td>203</td>
</tr>
<tr>
<td>FIGURE 17</td>
<td>NetView to AF/OPERATOR: Executing a NetView Exec</td>
<td>212</td>
</tr>
<tr>
<td>FIGURE 18</td>
<td>AOMVSCMD: Using NVIDATA() to Extract AF/OPERATOR Data</td>
<td>213</td>
</tr>
<tr>
<td>FIGURE 19</td>
<td>AF/OPERATOR to OMEGAMON for MVS: Executing an AF/OPERATOR Exec</td>
<td>216</td>
</tr>
<tr>
<td>FIGURE 20</td>
<td>MVDEMO Exec Example</td>
<td>246</td>
</tr>
<tr>
<td>FIGURE 21</td>
<td>MVUPDATE Exec Example</td>
<td>247</td>
</tr>
<tr>
<td>FIGURE 22</td>
<td>Console Output Example</td>
<td>248</td>
</tr>
</tbody>
</table>
About This Book

This User’s Guide describes AF/OPERATOR® facilities that can further automate your data center.

This guide tells you how to program AF/OPERATOR to automatically respond to events that indicate or affect the performance of a computer system. AF/OPERATOR runs under the IBM OS/390 or z/OS operating system.

Terminology

For an introduction to AF/OPERATOR terminology, see “Glossary” on page 321.

Documentation set information

The documentation listed in the following table is available for AF/OPERATOR. To order additional product manuals, contact your Candle Support Services representative.

- 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 User’s Guide
- AF/OPERATOR Started Task Manager
- AF/OPERATOR: Using the Subsystem Logging Facility
- Connecting AF/OPERATOR and AF/REMOTE Using TCP/IP
- Candle Products 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 User’s Guide Version 340" in the subject line.
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.
What’s New in AF/OPERATOR
Version 340

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(xxxxxxxxx) overrides the default name of the trusted hostnames member in RKANPAR. The LINKDEFS(xxxxxxxxx) 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.

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

This chapter describes AF/OPERATOR and gives examples of how you can use this product to streamline the operation of your data center and to increase productivity throughout your enterprise.

Chapter Contents

AF/OPERATOR and Your Enterprise ................................................. 18
What AF/OPERATOR Can Do for You ........................................... 20
Multiple AF/OPERATOR Address Spaces .................................... 21
Using REXX Shared Variables in AF/OPERATOR ....................... 26
Extended MCS Console Support ................................................. 27
Using EMCS Consoles Services .................................................. 28
AF/OPERATOR and Your Enterprise

Definition of AF/OPERATOR
AF/OPERATOR is a bridge between OMEGACENTER and the rest of your enterprise. It enables two-way communication between Candle’s OMEGACENTER products and other products, including:
- console automation products
- IBM’s NetView®
- ASCII hosts in a distributed systems environment

Working together with OMEGAVIEW™, AF/OPERATOR lets you see host and network status information on a single panel. OMEGAVIEW is Candle’s status display product that uses status lights to show, at a glance, the health of systems being monitored.

AF/OPERATOR and other products
Figure 1 illustrates the relationship between AF/OPERATOR and other products.

FIGURE 1. AF/OPERATOR and Other Products
Communication capabilities

As this figure shows, AF/OPERATOR accepts a wide variety of inputs and transmits a wide variety of outputs. For example, AF/OPERATOR:

- responds to OMEGAMON® exceptions
  OMEGAMON is Candle’s realtime software performance monitoring product line for IBM operating environments (MVS, CICS™, IMS, DB2®, and VSE). OMEGAMON alerts you to exceptional conditions automatically and also shows the status of the operating environment.

- receives messages from and sends messages to the MVS console

- AF/OPERATOR can respond to an OMEGAMON exception by issuing a message to the console.

- communicates with NetView
  AF/OPERATOR can respond to an OMEGAMON exception by issuing a generic NetView alert.

- communicates with OMEGAVIEW
  AF/OPERATOR can respond to NetView information by posting status to OMEGAVIEW.

- communicates with VTAM® applications through the Programmerless Open VTAM Interface (POVI)
  POVI is an AF/OPERATOR feature that passes information to and from VTAM applications such as CA7™. You can use POVI scripts to automate sequences of keystrokes and to insert variables into specific screen fields or cursor locations.

- communicates with ASCII hosts in a distributed systems environment through AF/REMOTE®
  If AF/REMOTE detects an exception on a UNIX machine, AF/OPERATOR can post status to OMEGAVIEW and send a message to the console.

- communicates with data center personnel by voice and beeper, through AF/REMOTE
What AF/OPERATOR Can Do for You

Introduction

Although AF/OPERATOR is a versatile, complex product, you can begin immediately to enjoy its benefits by using it for simple but powerful tasks. This section shows two examples of ways you can put AF/OPERATOR to use quickly and easily.

OMEGAMON for CICS exception results in a console message

You can write a simple trap (a rule statement that determines how AF/OPERATOR responds to system events) to issue a message to the console whenever OMEGAMON for CICS detects a response time problem:

```
"TRAP ADD(EXCSAOC) XOC('+ XCSA +') ACTION(''WTO ''&AOTEXT''') ENABLE"
```

Once this trap is defined, the following actions will occur when OMEGAMON for CICS detects a response time exception:

1. AF/OPERATOR collects the response time exception information from OMEGAMON for CICS.
2. AF/OPERATOR issues a message to the console.
3. A console automation product can take action to resolve the problem.

OMEGAMON for MVS exception triggers a NetView alert

You can also write a trap to respond to unsolicited error information from OMEGAMON (in this case, a TSO response time exception from OMEGAMON for MVS) by issuing a generic NetView alert. Here is an example of such a trap command, embedded in a REXX exec:

```
"TRAP ADD(EXTSOROM) XOM('+ TSOR +') ENABLE",
"ACTION(''NVALERT TYPE(PERM) ALID('''EEE98''')",
"DESC('''2000''') PC('''1001''')'',
"USER('''0111,1000''')'',
"HIER('''&AOJNAME,PROD''')'',
"TEXT('''&AOTEXT'''))"
```

Once this trap is defined, the following actions will occur when OMEGAMON for MVS detects a response time exception:

1. AF/OPERATOR collects the response time exception information from OMEGAMON for MVS.
2. AF/OPERATOR issues a generic NetView alert.
3. A NetView user, now aware that there is a problem, can investigate further. For example, if the NetView Interface and OMEGAMON Interface are installed, a NetView user could send a command through AF/OPERATOR to OMEGAMON for MVS (in this case, the TSOX command) and then retrieve more data about the problem.

The chapters that follow give many more examples of ways to use AF/OPERATOR to improve the performance of your data center.
Multiple AF/OPERATOR Address Spaces

Introduction

Multiple copies of AF/OPERATOR Version 340 can run on the same MVS system. This separation into multiple address spaces allows you to:

- test new procedures on one AF/OPERATOR address space while isolating production work on another
- increase throughput by dispatching work to multiple AF/OPERATOR address spaces
- configure AF/OPERATOR address spaces to match corporate priorities by intercepting routine events at lower-priority address spaces and sending critical jobs to a higher-priority address space
- enhance the reliability and availability of AF/OPERATOR

Identifying AF/OPERATOR address spaces on an MVS system

The figure below illustrates an MVS system, System A, with several copies of AF/OPERATOR installed using Candle default names. The address spaces are identified by their subsystem names (as specified with the SUBSYS startup parameter during product customization). The AF/OPERATOR subsystem names on an MVS system must be unique. By Candle naming convention, the last digit of an AF/OPERATOR subsystem name also indicates its relative order (see “Order of event processing” on page 23). For example, the AF/OPERATOR identified as O150 should be the first AF/OPERATOR address space to start.
Issuing console commands

Because multiple AF/OPERATOR address spaces may be executing on a single MVS system, an AF/OPERATOR command issued from the console must be prefixed by the subsystem name of the AF/OPERATOR address space where you want the command to execute, or you will receive an error message.

If you want to run an AF/OPERATOR address space in compatibility mode, where prefixing commands with the subsystem name is not required, refer to the AF/OPERATOR Configuration and Customization Guide for a description of the KOGOCMDS REXX exec.
Order of event processing

The order in which AF/OPERATOR address spaces process messages and commands is documented by the RO (relative order) startup parameter. This should also be the order in which the AF/OPERATOR address spaces are started after the system IPL. An AF/OPERATOR with an RO of 0 should be the first AF/OPERATOR to start, the next AF/OPERATOR to start should have an RO of 1, and so on (assuming default customization values are used).

**Note:** If you do not start the AF/OPERATOR address spaces in the order indicated by the RO parameter, the order of event processing is unpredictable.

For AF/OPERATOR address spaces that have the same relative order, event processing is determined by the order in which these address spaces are started after the system IPL.

Displaying relative order: SHOW OG

To display the name, status, and relative order of AF/OPERATOR subsystems on the MVS system, issue the SHOW OG command.

The following is an example of this command when embedded in a REXX exec (note the subsystem name is not used):

```/* REXX */
"SHOW OG"
```

A typical response to this command follows:

```
<table>
<thead>
<tr>
<th>SSID</th>
<th>LINKID</th>
<th>ORD</th>
<th>PRODUCT</th>
<th>JOBNAME</th>
<th>STATE</th>
<th>DATE</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>O190</td>
<td>SYSA</td>
<td>000</td>
<td>OMC GATEWAY</td>
<td>OGMVS00</td>
<td>UP</td>
<td>02.191</td>
<td>10:52</td>
</tr>
<tr>
<td>O191</td>
<td>SYSAO191</td>
<td>001</td>
<td>OMC GATEWAY</td>
<td>OGMVS01</td>
<td>UP</td>
<td>02.191</td>
<td>11:51</td>
</tr>
<tr>
<td>O193</td>
<td>SYSAO192</td>
<td>003</td>
<td>OMC GATEWAY</td>
<td></td>
<td>DOWN</td>
<td>02.192</td>
<td>09:30</td>
</tr>
<tr>
<td>O195</td>
<td>SYSAO193</td>
<td>005</td>
<td>OMC GATEWAY</td>
<td>TESTOG5</td>
<td>UP</td>
<td>02.192</td>
<td>10:24</td>
</tr>
</tbody>
</table>
```

For the complete syntax of the SHOW OG command, see the *AF/OPERATOR Command Reference Manual*
Identifying AF/OPERATOR address spaces in a complex

AF/OPERATOR has VTAM LU 6.2 peer-to-peer communication capability which allows you to route message and command events from one AF/OPERATOR address space to another AF/OPERATOR address space. The sending AF/OPERATOR address space and the target AF/OPERATOR address space can reside on the same or on different MVS systems in your complex. This is useful because events on one AF/OPERATOR may require actions on another.

Figure 2 on page 22 illustrates several MVS systems in a complex, with copies of AF/OPERATOR installed on each. The AF/OPERATOR address spaces on an MVS system are identified by their subsystem name (as discussed in “Identifying AF/OPERATOR address spaces on an MVS system” on page 21). The AF/OPERATOR address spaces in an MVS complex, which could be made up of many MVS systems, are identified by their LINKID. Each AF/OPERATOR address space in an MVS complex must have a unique LINKID.

FIGURE 3. Multiple AF/OPERATOR Address Spaces on Multiple MVS Systems

Candle recommends that you name LINKIDs by concatenating the SMFID of an MVS system with the subsystem name of an AF/OPERATOR address space to yield a unique name.
LINKIDs named in this way are shown in Figure 2 on page 22. Following this naming convention, an AF/OPERATOR address space with a subsystem name of O121 running on an MVS with an SMFID of SYSA has a LINKID of SYSAO121.

An exception to this naming convention is made for AF/OPERATOR address spaces with a relative order of 0. For the convenience of customers migrating from previous Candle products which did not include multiple address space support, the recommended LINKID of AF/OPERATOR address spaces with a relative order of 0 is the SMFID alone. This allows automation procedures developed using pre-AF/OPERATOR Version 150 Candle products to function without modifications.

For more about AF/OPERATOR peer-to-peer communication, see “Understanding the Communications Environment” on page 93.
Using REXX Shared Variables in AF/OPERATOR

Where to find variables information
Information about the types and scopes of variables you can use with AF/OPERATOR, as well as information on REXX shared variables, appears in the AF/OPERATOR Command Reference Manual.

REXX shared variables
REXX shared variables (RSV) allow you to share variables between REXX execs. They 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 and can be used by all execs executing within a single AF/OPERATOR address space. 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 the AF/OPERATOR Command Reference Manual for information on REXX shared variables and functions.
Extended MCS Console Support

Introduction
AF/OPERATOR Extended MCS (EMCS) Console Support allows the OPER command to use EMCS consoles.

In a sysplex environment the competition for subsystem consoles can be intense. Regardless of the size of the sysplex, the combined total of physical consoles plus subsystem consoles is 99. If these resources become exhausted before a particular AF/OPERATOR address space initializes, the address space is unable to detect responses to commands that it issues. EMCS helps you eliminate this problem by allowing you to use EMCS consoles, which are not limited in number across a sysplex.

*Note:* The limit argument of the CONSOLE startup parameter establishes a limit on how many EMCS consoles can be active on AF/OPERATOR at any one time.

Intended users
This feature is intended for users who write automation programs (either in AF/AL or REXX) that use the OPER command with the RESP keyword.

Storage
All existing involved programs have (or will be) rewritten to operate with a 31-bit mode addressing. All new programs will be rewritten to take advantage of 31-bit addressing and residency modes. Where possible, existing programs will also be changed to 31-bit residency modes.

Installation Customizer
The AF/OPERATOR Customizer has been updated to configure AF/OPERATOR startup parameters in support of EMCS consoles.
Using EMCS Consoles Services

Invoking EMCS

If your MVS system is at level 4.2 or above, you indicate use of EMCS by specifying the CONSOLE(TYPE(EMCS)) startup parameter. Once you specify this parameter, no further explicit action is required to initialize the feature.

Note: The feature is disabled if your MVS version is below 4.2 or if you specify the CONSOLE(TYPE(SUBSYS)) startup parameter or allow the console type to default to SUBSYS.

Assigning console names

Default console names (used by OPER commands with no CONSOLE operand) are in the form prefix####, where

<table>
<thead>
<tr>
<th>prefix</th>
<th>A 1–4 character console name prefix (from the CONSOLE(PREFIX) startup parameter; defaults to the value of the SUBSYS startup parameter).</th>
</tr>
</thead>
<tbody>
<tr>
<td>####</td>
<td>A 4-digit sequential number.</td>
</tr>
</tbody>
</table>

Default console names are assigned at startup. Console names can also be assigned explicitly with the CONSOLE operand of the OPER command.

Note: Console names must be unique throughout a sysplex. If you use the same subsystem ID for AF/OPERATOR address spaces operating on different CPUs in the same sysplex, then conflicts are possible.

Coding automation procedures

You can code automation procedures that rely on EMCS consoles and then subsequently start AF/OPERATOR using subsystem consoles. In such a case, you must anticipate and react appropriately to non-zero return codes and differing behaviors when moving back and forth between the two console types.
Introduction

This chapter describes starting and stopping AF/OPERATOR. To successfully start
AF/OPERATOR, all required automated and manual customization steps must already be
completed.

Chapter Contents

Starting AF/OPERATOR .................................................. 30
Stopping AF/OPERATOR ............................................. 33
Starting AF/OPERATOR

Introduction

AF/OPERATOR startup procedures are established during customization. These options are contained in a PDS member referenced by the startup procedure. The default name of the parameter member is OGPRM\text{c##}, where:

- \text{c} is the last character of the system ID. For example, if the system ID were SP22, \text{c} would be 2.
- \text{##} is the address space ID.

Standard start procedure

You can start AF/OPERATOR by issuing the MVS START command from any operator’s console:

\text{START OMCGATE}

This command starts AF/OPERATOR using the default startup procedure name OMCGATE. If you used a different procedure name during the customization of AF/OPERATOR, use the appropriate name.

\text{Note: The first time you start AF/OPERATOR you may receive message AOP0744, which reads:}

\text{WRONG LRECL/BLKSIZE ON RKOGLHnn - OK TO MODIFY (Y OR N)?}

Respond Y to allow AF/OPERATOR to override the LRECL/BLKSIZE of the OGLOG\text{nn} file.

\text{Note: For whatever name you choose for the AF/OPERATOR started tasks, the priority assigned to that task must be as high as that for your JES system.}

Alternate start procedures

If necessary, you can create additional startup procedures with varying startup parameters. For example, you can have the option of restoring TOD traps at startup and not restoring user variables (or the reverse).

Create the following two members to contain alternate startup options:

- \text{RESTODS} Member used to restore TOD traps, but not previously checkpointed user variables. It must contain the parameter RELOAD\text{(TODS)}.
- \text{RESVARS} Member used to restore previously checkpointed user variables, but not TOD traps. It must contain the parameter RELOAD\text{(VARS)}.

You can use the following start commands to override the default startup parameter list.

\text{START omcgate,RKANPAR=RESTODS}
\text{START omcgate,RKANPAR=RESVARS}
An alternate method of overriding the default startup parameters is to specify parameters on the START command. The format is:

```
START omcgate,TYPE='parm1,parm2,...,parmn'
```
Here is an example for a system with a 1-character system ID of A:

```
START omcgate,TYPE='RELOAD(TODS,LIST),CID(1)'
```

Once AF/OPERATOR is running, you may want to issue the AF command from the console in order to display the current operating mode and the options in effect. See the AF/OPERATOR Command Reference Manual for a complete description of the AF command.

**Initial command**

One of the parameters specified while customizing AF/OPERATOR is COMMAND (CMD). COMMAND defines the name of the first command executed when AF/OPERATOR starts. By default, this is 'EX $$OGINIT'. The sample $$OGINIT file provided with AF/OPERATOR performs customization verification tests. Your initial file can contain commands to vary active the major node and activate the OMEGAMON virtual sessions, the NetView Interface, the Programmerless Open VTAM Interface, and the peer-to-peer connections.

**Cold starting a new version of AF/OPERATOR**

Normally, you will want to start a new version of AF/OPERATOR with the RELOAD(NONE) parameter. This option starts AF/OPERATOR without using previous checkpointing information by reformatting the checkpoint dataset.

**Error messages**

If you experience problems starting AF/OPERATOR and have verified that you have met VTAM requirements, check the error messages in the RKOGLOGM. If you cannot resolve the problem, produce copies of the AF/OPERATOR message log and the RKANPAR member, and contact Candle Support Services. Candle telephone numbers are listed in the front matter of this document.

If you need to provide a dump to Candle Support Services, produce an unformatted dump in machine readable format. Copy the dump to tape and accompany it with the JCL used to create it.
Stopping AF/OPERATOR

Procedure
To stop AF/OPERATOR and allow current matches to complete, enter the AF STOP command:

```
O340 AF STOP
```

This message appears:

```
!AOP0000  AF/OPERATOR V340 COPYRIGHT 20nn CANDLE CORP.
!AOP0999  taskname AF/OPERATOR ENDED
!AOP0004  taskname AF/OPERATOR MESSAGE LOG STOPPED
```

Do not use the AF STOP product control command unless you want to stop AF/OPERATOR entirely.

To provide downwards compatibility with previous versions of AF/OPERATOR, the AO STOP command functions as a command synonym for the AF STOP command.

Technical considerations

- If you want AF/OPERATOR to remain active but to take no action when traps are triggered, put AF/OPERATOR in warn mode. See the AF/OPERATOR Command Reference Manual for information on the AF WARN command.

- If you want to prevent AF/OPERATOR from trapping particular events, use the TRAP DISABLE command to disable those specific traps.
Stopping AF/OPERATOR
Introduction

This chapter explains what AF/OPERATOR traps are and how to use them. Understanding traps is central to understanding the capabilities of AF/OPERATOR. You can use traps to detect system events and take action when those events occur.

Chapter Contents

- Trap Basics ................................................................. 36
- How Traps Work ............................................................ 39
- Setting Traps ................................................................. 41
- Command (CMD) Traps ................................................... 45
- Time of Day (TOD) traps ................................................ 47
- Console Message (WTO) Traps ........................................ 49
- OMEGAMON Exception (xtype) Traps .............................. 51
- JES3 Traps ................................................................. 54
- Pattern Matching in Trap Statements ............................... 55
- Name Patterns ............................................................ 59
- Pattern Variables .......................................................... 61
- Variable Substitution ..................................................... 63
- Specifying Trap Responses .............................................. 65
- Enabling and Disabling Traps .......................................... 69
- Modifying and Deleting Traps .......................................... 70
- Getting Information about Traps ....................................... 71
- Journaling Trap Data Using the Subsystem Logging Facility 73
- Cross-Reference of Traps and Command Files .................... 74
- Report on Command Files Performing Trap Management ....... 75
- Report on Traps Manipulated by Command Files ................. 76
- Report on Traps Executing Command Files ......................... 77
- Report on Command Files Executed by Traps ..................... 78
- Report on Trap Definitions Sorted by Argument .................. 79
- Cross-Reference Facility Error Report .............................. 80
- Running the Cross-Reference Reports ............................... 81
Introduction

Traps allow AF/OPERATOR to monitor the availability of systems in your enterprise. Examples show you how to:

- set and enable traps both normally and dynamically
- modify, disable, and delete traps
- specify immediate and extended trap actions
- write a trap to translate a message
- write a trap to issue an operator command
- write a trap to execute an action at a time interval
- write a trap to warn the operator about a system failure
- use pattern matching to specify events to trap
- get information about traps
- journal trapped or user-specified data

Definition of a trap

A trap is a rule statement you write to define how AF/OPERATOR responds to one or more system events, such as messages, commands, OMEGAMON exceptions, or date and time information.

What you need to know

Writing traps requires a working knowledge of the system events you wish to trap (such as MVS messages, MVS commands, and OMEGAMON exceptions) plus a familiarity with AF/OPERATOR commands.

To use the advanced program control, data manipulation, and extended action capabilities of AF/OPERATOR you should also be familiar with REXX. If you need to brush up on REXX, review the following IBM references:

- TSO/E REXX/MVS Reference
- TSO/E REXX/MVS User’s Guide

To familiarize yourself with AF/OPERATOR commands, review the AF/OPERATOR Command Reference Manual.
Trapping System Events

Trap Basics

Trap terminology

We use the following terms in talking about traps:

- **Setting** a trap means defining the event you want AF/OPERATOR to trap and specifying the action you want AF/OPERATOR to take when the event occurs. Setting a trap leaves the trap disabled but ready to enable.

- **Enabling** a trap makes it ready to respond to the event you want to trap.

- The occurrence of the event **triggers** or **fires** the trap if you have enabled it.

- A **match** occurs each time an event triggers a trap.

- **Disabling** a trap prevents the event from triggering it, but the trap remains defined in AF/OPERATOR. **Deleting** a trap disables it and removes its definition from AF/OPERATOR.

Elements of a trap

AF/OPERATOR trap statements consist of the following elements:

- **Name**
  - A unique name you assign to identify the trap.

- **Type**
  - One of the following:
    - **CMD (Command)**—triggered by a command issued from an operator console or other user-defined source (defined using AOSIM). CMD traps respond to operator commands from MVS, VTAM, NetView®, JES2, and JES3; multiuser subsystems such as CICS, DB2, and IMS; and the OMEGAMONs for CICS, DB2, IMS, and MVS.
    - **TOD (Time of Day)**—triggered by a date-time combination. TOD traps respond to time of day, day of week, date, or interval for scheduling, repeating, and catch-up processing.
    - **WTO (Write to Operator)**—triggered by a WTO/WTOR, IMS, JES3 message to an operator console, or other user-defined source (defined using AOSIM). WTO traps respond to MVS or subsystem messages issued as WTOs or WTORs, and IMS and JES3 messages.
    - **xtype (OMEGAMON exception)**—triggered by an OMEGAMON exception

- **Text Pattern**
  - The pattern you specify to define events that will trigger the trap. For example, you can specify ‘**IST**’ as the text pattern to match where only messages beginning with IST will trigger the trap.

- **Triggering Condition**
  - The condition or conditions you specify to limit the number of events that will trigger the trap. For example, you can specify **jobname** as a condition where only events generated with that jobname will trigger the trap.

To activate the trap to watch for the specified event you must also specify the ENABLE keyword or issue the TRAP ENABLE command.
The trap creation process

The following steps outline the process you follow to create and use AF/OPERATOR traps:

1. Set, or define, the traps you need.
2. Enable the traps you have set.
3. Change or disable traps as necessary.
4. Display information about traps.
5. Delete traps you no longer need.
6. Journal trapped data.

Summary of TRAP and SHOW commands

You can manipulate traps and get information about their operation using the following commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>See Also</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHOW MATCHES</td>
<td>Display which matches AF/OPERATOR is currently processing</td>
<td>page 71</td>
</tr>
<tr>
<td>SHOW TRAPS</td>
<td>Display definitions for traps already set</td>
<td>page 71</td>
</tr>
<tr>
<td>TRAP ADD</td>
<td>Set CMD, TOD, WTO, or xtype traps</td>
<td>page 41</td>
</tr>
<tr>
<td>TRAP CHANGE</td>
<td>Modify a trap definition</td>
<td>page 70</td>
</tr>
<tr>
<td>TRAP DELETE</td>
<td>Delete a trap definition</td>
<td>page 70</td>
</tr>
<tr>
<td>TRAP DISABLE</td>
<td>Deactivate a trap already set</td>
<td>page 69</td>
</tr>
<tr>
<td>TRAP ENABLE</td>
<td>Activate a trap already set</td>
<td>page 69</td>
</tr>
</tbody>
</table>

For a complete description and the syntax of each command, see the AF/OPERATOR Command Reference Manual.
How Traps Work

AF/OPERATOR trap processing

AF/OPERATOR traps allow you to monitor MVS system events, OMEGAMON exceptions, CICS, IMS, and JES3, as well as the system date and time. When you have enabled the trap, and the event or combination of events you have defined in the trap occurs, AF/OPERATOR initiates the action or actions you have defined for this trap. These might include simple actions, such as suppressing a message or command, or complex action, such as starting a system shutdown process.

You can use the immediate action of the trap to suppress the message or command that triggered the trap or let the message or command complete normally. You can use the extended action of the trap to have the trap issue new messages or commands, or execute REXX execs. Figure 4 illustrates this process.

FIGURE 4. Function of a Trap
Directing commands to specific AF/OPERATOR products

In environments where more than one AF/OPERATOR product is running on a single MVS system, commands entered from the console must be preceded by the subsystem name parameter identifying the AF/OPERATOR product to receive the command. The following syntax is for the TRAP ADD WTO command, but the subsystem name would be required for any product control command:

```
subsys TRAP ADD(trapname) WTO('full-pattern') option/s
```

For example:

```
O190 TRAP ADD(CLEANUP) WTO('IST*') SUPPRESS
```

directs the TRAP ADD WTO command to the AF/OPERATOR subsystem named O190 in a multiple address space environment. The subsystem name is only specified when entering AF/OPERATOR commands from the console. For a more on this topic see “Multiple AF/OPERATOR Address Spaces” on page 21.
Setting Traps

Introduction
There are two ways to set a trap or group of traps:
- embed the trap in a REXX exec
- enter the trap dynamically from the AF/OPERATOR console

Guidelines for setting traps
You set, or define, a trap when AF/OPERATOR is running by executing a TRAP ADD command specifying the following trap elements:
- trap name
- trap type (WTO, CMD, TOD, or xtype)
- trigger criteria
- action to take when the trap fires
- control options (if not specified, AF/OPERATOR supplies default values)

For example:

```
"TRAP ADD(CLEANUP) WTO('IST*') SUPPRESS"
```

sets a trap, named CLEANUP, with the following logic:

```
IF
  AF/OPERATOR detects a WTO starting with IST
THEN
  Suppress display of the WTO at the console.
```

The TRAP ADD command defines CLEANUP in the following way:
The asterisk (*) in the message pattern is a wildcard character that matches anything in a message after IST. (See “Pattern characters” on page 55.)

**Note:** Defining a trap does not activate it; you must also enable it. (Enabling and disabling traps is covered in “Enabling and Disabling Traps” on page 69.)

This TRAP ADD command:

```
“TRAP ADD(trapname) CMD("pattern") CONSOLE(n) SUPPRESS”
“ACTION("command")”
```

sets a trap that has the following logic:

```
IF
   AF/OPERATOR detects a command
   with a certain text pattern
   and issued from console n
THEN
   Don’t execute the command
   Issue a new command with a certain text
```

The TRAP ADD command instructs AF/OPERATOR to set a trap as follows:
Examples

Following are two examples of traps. Each trap command is enclosed in quotation marks, which are required for any trap included in a REXX exec.

1. When enabled, the following trap suppresses the display on the console of any message beginning with the string $HASP1.
   
   "TRAP ADD(NOJES1) WTO('$HASP1') ENABLE SUPPRESS"

2. The more complex trap shown below will stop a TSO user from issuing P commands from the SDSF facility.
   
   "TRAP ADD(NOSDSF) CMD('P *') ENA JOBTYPE(TSU) SUPPRESS",
   
   "ACTION('TPUT 'YOU MAY NOT &&AOTEXT' USERID(&&AOJNAME)')"

The trap writer supplies the message text when defining the &&AOTEXT variable. That text appears on the user’s terminal when they attempt the prohibited action.
Setting Traps

Setting traps dynamically

You can set or otherwise modify traps dynamically. For example, you can use one trap to set another trap or disable a series of traps.

The following trap disables all traps whose names start with CICS when CICS ends.

```
“TRAP ADD(NOCICS) WTO(‘IEF404I CICSPROD *’),
“ACTION(‘TRAP DISABLE(CICS*)’)”
```

This trap:

```
“TRAP ADD(SUPPRCMD) CMD(‘SUPPRESS *MSGID’),
“SUPPRESS ACTION(‘EX MSGSUP’)”
```

executes the REXX exec MSGSUP, that includes the following:

```
/* REXX */
z = ‘GLBVGET(“MSGID”)’
IF z <> 1 THEN
return 2
ELSE
“TRAP ADD(“MSGID”),
“WTO(“MSGID”.”) SUPPRESS ENABLE”
```

In this way the trap SUPPRCMD creates a new console command, SUPPRESS. Entering the command SUPPRESS ABC123 at the console creates a suppression trap for message ABC123. Thereafter, AF/OPERATOR will trap and suppress message ABC123. This example also illustrates the use of the Candle-supplied REXX function GLBVGET to access the value assigned to the global variable MSGID.
Command (CMD) Traps

Description

CMD traps intercept operator-issued commands and respond to them before the system executes them. CMD traps can:

- monitor or secure usage of specific operator commands
- abbreviate a series of frequently used commands (create new operator commands)
- simplify MVS, CICS, and IMS command syntax for computer operators

AF/OPERATOR monitors all commands issued by SVC 34, the AF/OPERATOR IMS or JES3 interfaces, or other user-defined source (defined using AOSIM). These include all operator commands issued from MVS operator consoles, those issued from TSO terminals in OPER mode, and those issued internally by software products that process operator commands.

CMD trap logic

A CMD trap uses the following logic:

```
IF
  AF/OPERATOR detects a command with a certain pattern
  [AND the command was issued from a certain origin]
THEN
  [Suppress the command]
  [AND execute a certain new command or REXX exec]
  [AND perform certain other actions
    For example, log the event in MSGLOG]
```

You must specify the command pattern AF/OPERATOR will search for. All the other items are optional.

Command selection criteria can be one, or a combination of, the following:

- The pattern of the text. For example, AF/OPERATOR can look for a command beginning with ACT and ending with NODE.
- The name and type of the task that issued the command.
- The origin of the command. For example, it can look for a command originating in a certain jobclass and address space, or from a console ID.

The AF/OPERATOR response can be one or more of the following:

- Suppress the trapped command. For example, AF/OPERATOR may suppress a command not authorized for use by a certain operator.
- Modify the trapped command. AF/OPERATOR modifies a command by first suppressing it and then issuing a new one. Variables in the new command can contain text from the suppressed command.
- Execute a REXX exec or an operator command.
CMD trap considerations

When you create a command trap, you effectively create a new MVS command that takes precedence over any existing command of the same name. As a result, when designing command traps, you need to consider how your traps could affect MVS operations. Intercepting existing command traffic can cause unexpected results.

You need to use caution when choosing and placing special characters in user-defined AF/OPERATOR commands. Do not place special characters as first characters because other subsystems, such as DB2, use commands that start with special characters. Avoid using special characters that other systems and subsystems assign special meanings to. For example, JES2 defines a character-delete character, and the latest MVS versions define a command-delimiting character.

Note: Use the WAIT option of the TRAP command with caution. Most operator commands and certain system messages are issued from the CONSOLE address space. Putting the console address space in a WAIT may result in the loss of all console functions.

Using a CMD trap to issue an operator command

You can use a command trap to allow you to type a shortened version of an operator command. You normally type the whole operator command to display the status of VTAM nodes, as shown in the example below:

```
D NET,ID=nodename
```

To shorten this command, you can define a trap as follows:

```
"TRAP ADD(NAME)",
"CMD('DNET 'NODE')",
"ACTION('OPER "D NET,ID=&NODE" ')",
"SUPPRESS ENABLE NOLOG"
```

Then, when you issue the shortened version of the command on the console as follows:

```
DNET nodename
```

AF/OPERATOR traps that command, captures the node name in the variable NODE specified in the trap, and issues the full operator command specified in the ACTION keyword.
**Time of Day (TOD) traps**

**Description**

Date-specific and time-specific traps schedule actions by

- time of day
- date
- recurring time interval
- day of week

You can schedule an action to take place periodically throughout the day. If the system (or AF/OPERATOR) is not active at the specified time, you can set AF/OPERATOR to perform catch-up processing or to skip the action before rescheduling it.

**TOD trap logic**

A TOD trap uses the following logic:

```
IF (when)
    A certain date and time combination occurs
THEN
    [Execute a certain command or REXX exec]
    [AND repeat this response at certain intervals]
```

You must specify the date and time combination AF/OPERATOR should wait for. All the other items are optional.

**Note:** *Time specification TOD(00:00:00 mm/dd/yy) is not a valid entry.*

Triggering criteria can be one, or a combination of, the following:

- the current time (you request AF/OPERATOR to trigger a trap as soon as you define and enable it)
- a date and time of day
- a day of the week, or the weekdays, or the weekends

The AF/OPERATOR response can be one or more of the following:

- execute an operator command or REXX exec
- repeat its action at regular intervals (from 1 second to 1 day)
**TOD trap considerations**

A TOD trap specifies the time at which you want the trap to trigger; however, because of both MVS and AF/OPERATOR dispatching considerations, what you actually specify is the earliest time the trap will trigger. Depending on the volume of matches AF/OPERATOR is processing and the system load, the action of a TOD trap may not occur until several seconds after the specified time. To minimize this delay, we recommend you run AF/OPERATOR at an MVS dispatching priority equal to that of VTAM and slightly above JES.

This information also applies to the WAIT command with the SECONDS operand because it builds an internal TOD trap.

**Example: Using a TOD trap to execute an action at a time interval**

The following example shows a trap that executes a REXX exec at 8:00 a.m. every day (an interval of 24 hours).

```
“TRAP ADD(TTSCICS),
“TOD(08:00:00) INTERVAL(24:00:00),
“ENABLE ACTION(‘EX CICSUP’)”
```

The REXX exec **CICSUP** contains commands and logic to start CICS. The exec contents are not shown here.
Console Message (WTO) Traps

Description

WTO traps respond to a specified pattern of WTOs or WTORs. They can

- highlight messages that require attention
- issue one or more MVS or subsystem operator commands
- execute a REXX script

AF/OPERATOR monitors all messages issued by SVC 35, the AF/OPERATOR IMS or JES3 interfaces, or other user-defined source (defined using AOSIM). These include all messages issued from batch programs shown in the JES job log and all those issued by the operating system in response to operator commands.

Note: 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.

WTO trap logic

A WTO trap uses the following logic:

\[
\text{IF} \quad \text{AF/OPERATOR detects a message with a certain pattern} \\
[\text{AND the message has a certain origin}] \\
[\text{AND the message destination is a certain console}] \\
[\text{AND the message has certain descriptor codes}] \\
\text{THEN} \\
[\text{Execute a certain command or REXX exec}] \\
[\text{AND perform certain other actions}] \\
[\text{For example, issue a new message}] \\
\]

You must specify the message pattern AF/OPERATOR should search for. All other items are optional. Message selection criteria can be one, or a combination of, the following:

- The pattern of the text. For example, AF/OPERATOR may look for a message beginning with \$HASP.
- The name or type of the job that issued the WTO/WTOR.
- The origin of the message. For example, AF/OPERATOR may look for a message from a certain job class and address space, or the SMF system ID of the system executing the job.
- The destination of the message. For example, AF/OPERATOR may look for a message routed to console 1.
- The features controlled by message descriptor codes. For example, AF/OPERATOR may look for a message requiring eventual action.

The AF/OPERATOR response may be to execute a REXX exec or an operator command.
Example: Using a WTO trap to translate a message

The following example shows a trap that translates a standard CICS message into a simpler message for console display. The CICS system message might be:

```
+DFH1517 CICSPROD : CONTROL IS BEING GIVEN TO CICS
```

The following AF/OPERATOR trap will suppress the original message and issue its translated version, which will appear on the console:

```
"TRAP ADD(TWSCICS)",
"WTO(‘+DFH1517 *REGN : CONTROL *’)",
"ENABLE SUPPRESS",
"ACTION(‘WTO ‘&&REGN IS UP’ DESC(2)’)"
```

When the original message triggers the trap, the new message will appear as follows:

```
CICSPROD IS UP
```

The asterisk (*) following CONTROL is a wildcard that represents any text. The asterisk before REGN identifies a pattern variable and the ampersands (&&REGN) allow you to use the pattern variable in the ACTION portion of the TRAP ADD command. Pattern variables and variable substitution are discussed later in this chapter.
OMEGAMON Exception (xtype) Traps

Description

The following OMEGAMON exception traps are available:

- **XOC**: Trap OMEGAMON for CICS exceptions
- **XOI**: Trap OMEGAMON for IMS exceptions
- **XOM**: Trap OMEGAMON for MVS exceptions
- **XO2**: Trap OMEGAMON for DB2 exceptions

This manual refers to OMEGAMON exception traps as *xtype* traps.

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.

Using OMEGAMON exception traps

OMEGAMON uses a set of user-defined parameters to monitor your computer system. These are called thresholds. They tell OMEGAMON when to notify you that certain conditions exist which are exceeding their thresholds. These conditions, called exceptions, may indicate current or potential problems in your system. You can monitor and respond to OMEGAMON exceptions with OG/MVS's OMEGAMON exception traps.

The structure of all OMEGAMON exception traps (XOC, XO2, XOI, and XOM) is basically the same. The following example shows an XOC trap that warns the operator when a particular CICS fails to update the time-of-day clock.

An exception message from OMEGAMON for CICS might be:

```
+ XTOD CICS Time Of Day not being updated - CSACTODB = 10:25:47.02
```

A REXX exec OCEXXTOD defines a trap (also named OCEXXTOD) to respond to the message:

```
"TRAP ADD(OCEXXTOD)",
"XOC(\+ XTOD * )",
"ENABLE NOLOG",
"ACTION(\+ OCXTOD)"
```

The trap, when triggered, executes a REXX exec named OCXTOD. Both the OCEXXTOD and OCXTOD execs are in the AF/OPERATOR sample library.

An exception trap is triggered by an OMEGAMON exception and typically responds to the exception with remedial action. For example, the command:

```
"TRAP ADD(EXIMIRT) XOI(\+ MIRT*) MATCHLIM(5) ALTACT(\+ EXEC IMIRT)"
```

defines a trap named EXIMIRT that:
1. Monitors OMEGAMON for IMS for exceptions beginning with MIRT.

AF/OPERATOR monitors IMS exceptions in the form produced by the command LXIMS; that is, they begin with the name of a 4-letter minor command like MIRT. Similarly, it monitors MVS, CICS, and DB2 exceptions as beginning with their 4-letter minor command.

2. Executes the REXX exec IMIRT on the fifth MIRT exception (and on the tenth, the fifteenth, and so on). The MATCHLIM operand determines the frequency.

In the following exception trap, the TRAP ADD command traps OMEGAMON exceptions that occur when a job named PAYROLL goes into a wait state. The command is set up so that the trap does not execute until the tenth time the exception occurs. At that time, the trap executes the REXX exec MWAIT, which performs actions such as marking the job nonswappable and sending a message. The REXX exec MWAIT is described in “Issuing OMEGAMON Commands” on page 195.

```
“TRAP ADD(EXMWAIT) XOM(‘+ WAIT ??? PAYROLL‘") SYSID(SYSA)”,
“MATCHLIM(10) ACTION(‘’) ALTACT(‘EX MWAIT’)”
```

TRAP ADD(EXMWAIT)

Adds a trap named EXMWAIT.

XOM(“+ WAIT ??? PAYROLL “")

Monitors all links with OMEGAMON for MVS.

The pattern in the subfield instructs AF/OPERATOR to look for any exception message made up of these elements:
- + (plus sign)
- one or more blanks
- WAIT
- one or more blanks
- any 3 alphanumeric characters
- one or more blanks
- PAYROLL
- one or more blanks
- any further continuation

The trap will be triggered by OMEGAMON wait exceptions with the jobname PAYROLL.

SYSID(SYSA) AF/OPERATOR will monitor only exceptions from System A.

MATCHLIM(10) The first 9 times the trap triggers, AF/OPERATOR will issue the command in the ACTION subfield. It will issue the command in the ALTACT subfield the tenth time it is triggered. This cycle repeats for subsequent executions of the trap.

ACTION(‘’ ) Since the ACTION subfield is null, AF/OPERATOR will take no action in response to the trap until the tenth triggering.
ALTACT('EX MWAIT')

When the tenth WAIT exception occurs, AF/OPERATOR executes the REXX exec MWAIT.

The REXX exec MWAIT determines whether the payroll job has been swapped out for more than 1 minute and, if it has, marks the job non-swappable. MWAIT is described in “Issuing OMEGAMON Commands” on page 195.
JES3 Traps

Description

You can write traps for JES3 console messages or commands the same way you would for MVS console messages and commands, with the following qualifications:

- If you want a trap to issue a command to JES3, use the JES3OPER command. See the AF/OPERATOR Command Reference Manual for the syntax of this command.
- You can build JES3 command macros to use as follows:
  - The operator at a JES3 or MVS console enters a short mnemonic text that stands for a sequence of commands to JES3.
  - AF/OPERATOR traps and suppresses the mnemonic and executes the sequence of commands using a JES3OPER command.
Pattern Matching in Trap Statements

Definition

*Pattern matching* is the trapping of one or more text strings by matching them to a specified combination of pattern characters. Pattern matching is an important tool for creating and using traps.

*Pattern variables* are the generic text strings created by combining explicit text with *wildcard* characters, such as question marks (?) and asterisks (*).

Purpose

Pattern matching enables you to:

- Reduce the number of traps required for automation. The special characters allow you to represent all possible variations on a string.
- Trap, suppress, or display all of a certain type of message, command, or exception. By using wildcard characters to represent the text that varies, you can manipulate the whole group.
- Capture one or more triggering events for use in the action part of the trap.

Pattern characters

A pattern character may consist of:

- an alphanumeric character (0–9 and A–Z)
- a wildcard character:
  
<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>Question mark, which matches any one alphanumeric character.</td>
</tr>
<tr>
<td>#</td>
<td>Pound sign, which matches any one numeric digit.</td>
</tr>
<tr>
<td>*</td>
<td>Asterisk, which matches any string, including the null string.</td>
</tr>
</tbody>
</table>

*Note:* The asterisk is extremely resource-intensive if used as the first character in a pattern. Use caution!

- single blank Which matches any number of blanks.
- multiple blanks Which match the same number of contiguous blanks in the target string as 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 character.
- **Period (.)** Terminates a pattern variable name.

You can use the question mark, pound sign, and asterisk pattern characters to create pattern variables. See “Pattern Variables” on page 61 for a description of how to do this.
Case Sensitivity

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 are typically not case sensitive because AF/OPERATOR converts them to uppercase letters.

Examples

The following 4 examples show how to use pattern matching to construct a trap:

1. To suppress all messages containing the words $HASP10 and ending with one digit, use the pattern string

   ‘$HASP10#’

   This matches any character string containing exactly 8 characters, beginning with $HASP10, and ending with a single digit. ($HASP100 or $HASP109 but not $HASP10A)

2. The $HASP373 message contains variable fields that contain different information depending on the situation:

   $HASP373 TSOUSER1 STARTED
   $HASP373 PRODJOB1 STARTED - INIT 2 - CLASS M

   The following trap watches for a $HASP373 message that contains any text (indicated by the asterisk wildcard) followed by the word STARTED:

   “TRAP ADD(TSHS373A)”,
   “WTO($HASP373 * STARTED)”,
   “SUPPRESS ENABLE”

3. This trap watches for and suppresses a $HASP373 message that contains any text (indicated by the asterisk wildcard), followed by the word INIT, followed by any text (again indicated by an asterisk wildcard):

   “TRAP ADD(TSHS373B)”,
   “WTO($HASP373 * INIT **)”,
   “SUPPRESS ENABLE”

4. This trap watches for and suppresses any $HASP373 message:

   “TRAP ADD(TSHS373C)”,
   “WTO($HASP373 **)”,
   “SUPPRESS ENABLE”
Using asterisks

*Use the asterisk (*) wildcard character with caution.* Because it can match any pattern, it can be extremely resource-intensive if used as the first character in a pattern. Do not use the asterisk to begin a pattern followed by literal text to match; all messages will be scanned to seek the target pattern.

AF/OPERATOR recognizes the asterisk (*) as a wildcard character in full patterns under the following conditions:

- 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. AF/OPERATOR matches a target string of zero or more characters that begins at the position of the asterisk.

For example, in the trap defined as follows:

```
"TRAP ADD(TRP1) WTO('JOB##### *.ABENDED') ... "
```

AF/OPERATOR traps only WTO messages whose text begins with the word JOB followed by 5 numeric characters, followed by a space then any text (indicated by the asterisk wildcard), followed by the word 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 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

  `ABCW,X,Y,Z @$`

since the asterisk matches

  `W,X,Y,Z @$`

The pattern

`ABC*,*`

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

  `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

  `Y,Z @$`
Using blanks

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

`'IEC210lb214-'`

matches all of the following strings:

- `IEC210lb214-`
- `IEC210lbb214-`
- `IEC210lbbbb214-`

However, the full pattern

`'IEC210lbb214-'`

matches only the string

- `IEC210lbb214-`

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.
Name Patterns

Definition

A name pattern is a short text pattern you can use for matching names in selected name fields of AF/OPERATOR commands. A name is a string of 1 to 8 alphanumeric or national characters. For example, a name pattern consisting of the string S followed by an asterisk matches every target name starting with S. Usually, however, a name pattern is the first part of a name—from 1 to 7 characters—followed by an asterisk, for example, TRAP111* or TRAP11*.

Using wildcards

You can use the asterisk (*), question mark (?), or pound sign (#) as a wildcard character in the name fields of the following commands to match trap, variable, or file names:

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

When you use an asterisk alone, the asterisk wildcard character will match everything. Use caution when using the asterisk alone; it is extremely resource-intensive.

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.
Name Patterns

Cautions

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 DELETE(*)**

would delete every user-defined trap in storage.
Pattern Variables

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

You can use the asterisk (*), question mark (?), or pound sign (#) to define a pattern variable.

The PARSE command
The PARSE command is a powerful tool for writing REXX execs to be executed by traps. It can supplement or replace the use of pattern variables. See the PARSE command in the IBM publication TSO/E REXX/MVS Reference for more information.

Defining pattern variables
To define a pattern variable, follow these steps:

1. Write one of the following pattern character strings:
   ▪ a string of n question marks (?) to match n alphanumeric characters in the text string
   ▪ 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 8 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.
Pattern Variables

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:

- 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 a match exists. AF/OPERATOR does not match a null string.

For information about using the PARSE command, which can supplement or replace pattern variables in a trap REXX exec, see the IBM publication TSO/E REXX/MVS Reference. For more information about AF/OPERATOR variables, see the AF/OPERATOR Command Reference Manual.

Creating patterns using software documentation

You can use the messages and codes manual for your operating system or software product as a guide for creating message patterns. Message manuals usually show the variable fields within a message and the delimiters used between variable fields. For example, if message IEC210I is documented as follows:

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

This pattern for the IEC210I 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
Variable Substitution

Introduction
Variable substitution enables you to pass data to a REXX exec. For example, if shutting down a CICS region is one of the tasks you are automating, you could use the information captured in the REGN variable (shown in the example on page 50) as part of the AF/OPERATOR command to shut down this region.

How it works
Match variables get information (a value) from a message or a command and make this information available to the REXX exec. If the REXX exec contains statements that reference a given variable name, those statements will have access to the value of the variable.

Syntax considerations
In certain situations, variable names must have a character prefix. AF/OPERATOR uses the ampersand (&) prefixed to the variable name to denote a variable inside a quoted string (other than a pattern) or inside a string enclosed in parentheses. In previous examples in this chapter, you saw the use of ampersands in the ACTION keyword of the TRAP ADD command (the ACTION keyword is a string enclosed in quotes).

```
"ACTION('WTO "&REGN IS UP" DESC(2)')"
```

Use ampersand prefixes as follows:

- A variable that AF/OPERATOR has to substitute when it executes the REXX exec and adds the trap must begin with a single ampersand (&).
- Variables that AF/OPERATOR has to substitute when an event triggers the trap must begin with a double ampersand (&&).

Note: You need to supply an ampersand only when you reference a variable inside a quoted string, because AF/OPERATOR treats these strings as literal values. The ampersands tell AF/OPERATOR to substitute a value for the variable. AF/OPERATOR does not substitute values in traps you add at the console, only in traps you add through a REXX exec.
Example of variable substitution

In the example below, a variable named VARA receives a value from a message that it matches.

```
“TRAP ADD(TWXMPL)”,
“WTO($HASP395 "VARA ENDED “)”,
“ENABLE NOLOG”,
“ACTION(’TPUT ‘&&VARA ended at &&AOTIME” TSOID(&&AOTSUID)’)”
```

The variable VARA is a pattern variable that obtains a value from the text of the matching string. The match variables AOTIME and AOTSUID are evaluated by AF/OPERATOR when the match occurs. (See the AF/OPERATOR Command Reference Manual for further explanation of match variables.)

The ACTION keyword of this trap references a variable; the value of this variable will be substituted as follows.

- When the REXX exec executes, all TRAP commands in the file also execute and take effect immediately. For example, when the file terminates, if the REXX exec contained a TRAP ADD command, AF/OPERATOR would have defined the TRAP but would have substituted values for any variables contained in that command.

- When the trap match occurs, AF/OPERATOR interprets its ACTION portion again and substitutes values for the variables one more time.
Specifying Trap Responses

Description
You specify AF/OPERATOR responses to trapped events by placing certain operands in the TRAP ADD command. This section describes some of the basic responses you can designate. See the AF/OPERATOR Command Reference Manual for complete information on TRAP ADD commands.

Specifying trap actions
You can specify two basic actions in the ACTION field of a trap:

- issue a single AF/OPERATOR command
- execute an AF/OPERATOR REXX exec

The AF/OPERATOR command can contain symbolically substituted text from pattern variables created from the message or command pattern, or from any variables available to the trap.

To send a replacement message, use one of the following commands:

- a WTO or TPUT command
- an OPER command that directs the system to issue the MVS SEND command

To issue a replacement command, use one of the following:

- an appropriate AF/OPERATOR command
- an OPER, JES3OPER, or IMSOPER command that directs the system to issue an appropriate MVS, JES3, or IMS command
- a CP command that directs the system to issue a DIAGNOSE command to the VM system

If a single command will not meet your purpose, execute a REXX exec, which allows for more sophisticated logic and command sequences.
Rules for quotes in the ACTION field

In the following discussion, the word *quote* means a single quote mark ('). The phrase *a pair of quotes* refers to ‘’. The four characters ” ” are referred to as *two pairs of quotes*.

If you include a quoted string inside another quoted string, double the number of quotes each time you move inwards. For visual clarity, you can put blanks between sets of quotes. To check that you have the correct number of closing quotes on the right side, follow this rule: the right side always has an odd number of quotes.

For example, the string in the ACTION field of a TRAP command has a single quote at each end, like this:

```
ACTION('_______________________________')
```

If you include a quoted string in the ACTION field, put a pair of quotes at each end of that string, like this:

```
ACTION('_______________________________')
OPER ''___________________''
```

For example, you could write:

```
TRAP ADD(Z) CMD(DA) ACTION('OPER "$DA$"')
```

If you include another quoted string, put two quotes at each end of it, like this:

```
ACTION('_______________________________')
OPER ''___________________''
SEND '''_____'''
```

For visual clarity, you can put blanks between the groups of quotes. In this example, the blanks are represented by b :

```
TRAP ADD(Z) CMD(HI) ACTION('OPER "SEND ""HELLO""b"b")
```

This is another example of the ACTION part of a TRAP command:

```
ACTION('OPER "SEND ""HELLO"",USER=(TSOID)" CONSOLE(1))
```
Using match variables

REXX execs executed by traps can use data available in match variables generated by AF/OPERATOR, especially those that store the contents of various JES2 JCT fields. For example, the following trap:

```
“TRAP ADD(NOTEUSER) WTO(‘*’) JOBTYPE(JOB) JOBNAME(‘TEST*’),
ACTION(‘EX NOTEUSR’)”
```

with the following line in a REXX exec named NOTEUSER:

```
TPUT ‘&AOTEXT’ USERID(&AOTSUID)
```

sends all job-related messages to the TSO user with the ID assigned by the NOTIFY= parameter in the user JCL.

For more information on this use of match variables, see the AF/OPERATOR Command Reference Manual.

Order of REXX exec execution

If AF/OPERATOR sends a REXX exec to a system already executing another REXX exec, that system enqueues the incoming exec. When the environment becomes available, the REXX exec executes. If two or more REXX execs are enqueued, they will execute in the order they were enqueued; first in, first out.

Logging messages

If you use the default LOG option in a TRAP ADD or TRAP CHANGE command, AF/OPERATOR will write a message to its own message log (RKOGLOGM) every time an event triggers the trap. If you use the NOLOG option, AF/OPERATOR will not write any message to the message log and will not log the results of the trap action. Use the default LOG option only for diagnosing trap action, then specify NOLOG, because this option improves AF/OPERATOR performance.

Note: Neither the LOG nor NOLOG option affects the messages written to SYSLOG.
Example: Setting traps with extended actions

Use the ACTION keyword in a trap to specify one extended action in addition to the immediate actions. The following example shows how to develop the components of a trap that suppresses and rewords a $HASP395 message. You specify the alternate WTO in the ACTION field of the TRAP ADD command.

```
“TRAP ADD(TWNEWMSG),
“WTO($HASP395 * ENDED *),
“ENABLE SUPPRESS",
“ACTION('WTO ‘The expected task has ended’ ‘)"
```

The line:

```
“TRAP ADD(TWNEWMSG)"
```

uses the AF/OPERATOR TRAP ADD command to define a trap named TWNEWMSG. You must always supply the name of the trap as part of the TRAP ADD command. This name should conform to your site’s naming conventions.

The line:

```
“WTO($HASP395 * ENDED *)”,
```

specifies WTO as the type of event you want AF/OPERATOR to look for and specifies the text of the message you want AF/OPERATOR to match. Note how the selection criteria of this trap, the text `$HASP395 * ENDED *`, uses asterisks. The asterisk (*) in the message pattern is a wildcard character that allows you to specify a key portion of the message in a single pattern. In this example, the asterisk tells AF/OPERATOR to trap all messages that begin with $HASP395, followed by a space, and then any text (the first *), followed by another space and then the word ENDED, followed by a space and then any text (the second *).

The remaining lines:

```
“ENABLE SUPPRESS",
“ACTION('WTO ‘The expected task has ended’ ‘)"
```

are keywords that control action(s) AF/OPERATOR will take. To activate the trap, you must specify the ENABLE keyword. The SUPPRESS keyword prevents the selection criteria (the text that is matched) from being displayed on MVS consoles. The ACTION keyword allows one extended action. In this case, the extended action returns the WTO message you specified as part of the ACTION keyword. Notice the grouping of single quotes in the ACTION keyword. Enclose the whole string inside the parentheses in single quotes, and enclose each string within this string in two consecutive single quotes. Following are a few pointers that make the use of quotes easier to follow:

- The last group of single quotes is always odd.
- Add one pair of single quotes around each string within the main string.
- Do not use the double-quote character for two single quotes.
Enabling and Disabling Traps

Description
You can set any number of traps at startup and enable only those you want active. Until you enable it, a trap does not trap messages or commands, or respond to system times.

Note: The keyword ENABLE (or ENA) often appears in TRAP ADD commands. This manual generally omits it in TRAP ADD examples.

You can enable or disable traps in either of two ways:
- by using the ENABLE or DISABLE option with a TRAP ADD command
- by issuing a TRAP ENABLE or TRAP DISABLE command

By default, a TRAP ADD command sets a trap but leaves it disabled. If you include the keyword ENABLE in the command, the trap becomes active as soon as AF/OPERATOR compiles the definition.

You can enable a trap you have set previously by issuing a TRAP ENABLE command. Similarly, you can disable an active trap by issuing a TRAP DISABLE command.

For example, for a trap named ABC, the command:

```
"TRAP ADD(ABC) ... ENABLE ..."
```

has the same effect as the following sequence:

```
"TRAP ADD(ABC) ..."

"TRAP ENABLE(ABC)"
```

You can enable or disable one trap by specifying its name in the command. To enable or disable groups of traps, you can specify one or more name patterns. For example:

```
"TRAP ADD(Q*) ... ENABLE ..."
```

enables all traps with names beginning with the letter Q.

Advantages of disabling traps
Disabling traps has the following advantages:
- By using only those traps you need, you reduce system overhead.
- You can build up a library of defined traps that you can enable or disable quickly at appropriate times.
Modifying and Deleting Traps

Description
You can modify a WTO, CMD, or xtype trap definition in one of two ways while AF/OPERATOR is running:

- Use a TRAP CHANGE command.
- Delete the trap with the TRAP DELETE command and issue a new trap definition with the TRAP ADD command.

To change a TOD trap you must delete the old trap with TRAP DELETE and add a new one with TRAP ADD.

Using the TRAP CHANGE command
Use the TRAP CHANGE command to modify a working trap definition temporarily. The change takes effect immediately and lasts until AF/OPERATOR is stopped or the trap is deleted or modified again with a TRAP CHANGE command. The TRAP CHANGE command can be used to modify any of the characteristics that can be specified when a trap is defined. Only the options to be changed need to be specified in the command.

There is no TRAP CHANGE command for TOD traps.

For more information, see the *AF/OPERATOR Command Reference Manual*.

Using the TRAP DELETE command
Use the TRAP DELETE command to deactivate a command and delete its definition. Traps are deleted from working storage. Their definitions remain in the command library stored on disk.

You can delete one trap by specifying its name in the command. Alternatively, you can specify one or more name patterns. In this way you can delete all traps whose names match the pattern.

For more information, see the *AF/OPERATOR Command Reference Manual*.

Replacing traps
To replace an entire trap definition, delete the trap with TRAP DELETE and issue a new TRAP ADD command, as follows:

```
TRAP DELETE(ABC)
TRAP ADD(ABC) ...
```
Getting Information about Traps

Description

AF/OPERATOR provides a number of ways for you to get information about the traps in your system. You can:

- Use the SHOW MATCHES command to display active trap matches.
- Use the SHOW TRAPS and SHOW TRAPS TITLE FORMAT commands to find out what traps are available in storage.
- Use the Cross-Reference Facility to see which traps are available in your trap libraries, and how they relate to each other and to your REXX execs.
- Use the trap TEST option to discover how a particular set of traps would work under various assumed operating conditions.

Displaying active matches: SHOW MATCHES

SHOW MATCHES displays active matches AF/OPERATOR is currently processing. You can specify which matches AF/OPERATOR displays by specifying trap number, trap type, all matches, active matches, or queued matches.

For more information on this command, see the AF/OPERATOR Command Reference Manual.

Displaying trap definitions: SHOW TRAPS

SHOW TRAPS displays trap definitions, including trap name, trap type, triggering pattern, other triggering conditions, immediate and extended trap actions, and trap control parameters. It also displays the number of times the trap has fired. Many traps exist in most systems. Issuing an unqualified SHOW TRAPS command may require some time to execute and may produce a significant amount of output.

SHOW TRAPS TITLE FORMAT enables you to specify up to 8 title parameters and up to 8 format parameters to customize the SHOW TRAPS command display.

For more information on these commands, see the AF/OPERATOR Command Reference Manual.

Obtaining SMF Data

By adding the STATS(COLLECT(opt, ..., opt),SMFREC(nnn) parameter to your startup parameters, it is possible to gather information about the performance of AF/OPERATOR subtasks using SMF data. There are four types of data collected in the SMF dataset when this parameter is invoked. Information can be collected about the MATCH that is created when a TRAP fires, about any EXECs that are run during the life of a match, about any FILE accesses made by the execs that are run, and about any IMMEDIATE changes made by the trap.

SMF data collection can be turned on or off using the AF STATS command as described in the AF/OPERATOR Command Reference Manual. If the startup parameter is specified with
the NONE option and the SMFREC number, the AF STATS command can be used to start or stop SMF collection at will without having to respecify the SMF record ID each time.

A complete description of the STATS parameter is found in the AF/OPERATOR Configuration and Customization Guide in the section “Startup Parameter Descriptions.” Appendix F of this manual provides the SMF record layout.

There are two sample programs (one in COBOL and one in REXX) in the TKANSAM library that demonstrate writing reports from SMF records. These programs are named KOGSTCOB and KOGSTREX, respectively.

Using the Cross-Reference facility

The Cross-Reference Facility provides reports that show the relationships between traps and REXX execs for all traps defined by all REXX execs in the AF/OPERATOR command library. By documenting the internal relationships among traps and REXX execs, the reports help you to see how AF/OPERATOR will respond to any specific event. You can forecast the impact of a change in the AF/OPERATOR command library and eliminate redundancies or conflicts in your traps and REXX execs.

Using the TEST option on traps

You can define and monitor an individual trap in the production system by using the TEST option of the TRAP command, as shown in the following example.

```
"TRAP ADD(testtrap) WTO(‘IEF233’) ENABLE ACT(‘EX LOGTAPE’) TEST"
```

This trap reacts to messages that occur on the live production system. However, no actions actually take place. Instead, AF/OPERATOR logs the actions it would have taken in RKOGLOGM.

The trap TEST option is like a warn mode applied selectively to individual traps.
Journaling Trap Data Using the Subsystem Logging Facility

Introduction

The Subsystem Logging Facility (SLF) enables you to journal trapped or user-specified data to an MVS dataspace and consists of the following new components:

- the backup journal datasets required in the AF/OPERATOR JCL procedure
- the AF/OPERATOR startup parameter, JOURNAL
- the JOURNAL, JOURATTR, and JOURHELP options on the TRAP command
- the Candle-supplied REXX function to journal user-specified data

Using the journal components

You use the journaling components as follows:

Backup journal datasets

These VSAM linear datasets serve as backup datasets for the MVS dataspace containing the journaled data. Their DD statements must be included in the AF/OPERATOR startup JCL. See the Configuration and Customization Guide for information.

JOURNAL startup parameter

At AF/OPERATOR initialization, this parameter specifies whether journaling is enabled and in what manner. Additional information is found in the Configuration and Customization Guide section “AF/OPERATOR Startup Parameters”.

Additional options on TRAP

Three additional options on the trap command specify whether the matched data from this trap should be journaled (JOURNAL); specify optional attributes for the SLF data (JOURATTR); specify the name for an optional help panel for SLF data (JOURHELP). Use of these Subsystem Logging Facility options is described in the Command Language Reference in the section “AF/OPERATOR. Commands”.

REXX JOURNAL function

You can use this Candle-supplied REXX function to write your own data directly to the journal. For information, refer to the Command Reference Manual section “System Functions”.
Cross-Reference of Traps and Command Files

This feature provides reports that show the relationships among all trap definitions and all command files in the AF/OPERATOR command library. This provides a useful overview of your current automation activities.

By documenting the internal relationships among traps and command files, the reports help you to see how AF/OPERATOR will respond to any specific event. You can forecast the impact of a change in the AF/OPERATOR command library and eliminate redundancies or conflicts in your traps and command files.

This helps you use AF/OPERATOR more efficiently by grouping related items.

**Note:** Processing time used by this facility is proportional to the number of items listed.

The printed reports are 132 columns wide. The following figures use short fields to show the reports as 80 columns wide.
Report on Command Files Performing Trap Management

This report, AOPRPT01:

- lists all command files alphabetically
- for each command file, lists any traps that it adds, deletes, enables, or disables
- displays a summary of command files performing trap management

**FIGURE 7. Report AOPRPT01**

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>TRAP</th>
<th>TRAP</th>
<th>TRAP</th>
<th>TRAP</th>
<th>TRAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE</td>
<td>LIB</td>
<td>FUNCTION</td>
<td>NAME</td>
<td>TYPE</td>
<td>ARGUMENT</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>----------</td>
<td>------</td>
<td>------</td>
<td>-----------</td>
</tr>
<tr>
<td>MORNING</td>
<td>1</td>
<td>ADD/ENA</td>
<td>@700</td>
<td>TOD</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DIS</td>
<td></td>
<td>@2300</td>
<td>TOD</td>
<td>23:00:00</td>
</tr>
<tr>
<td>NOON</td>
<td>2</td>
<td>ENA</td>
<td>@1200</td>
<td>WTO</td>
<td>$HASP*</td>
</tr>
</tbody>
</table>

---

Command Files Processed: 99999
Traps Added: 99999
Enabled: 99999
Disabled: 99999
Deleted: 99999

Libraries: 1 = USER.COMMAND.LIB1
2 = USER.COMMAND.LIB2

Command file libraries - 2
Report on Traps Manipulated by Command Files

This report, AOPRPT02:
- lists all traps alphabetically by trap name
- for each trap, lists any command files that add, delete, enable or disable the trap
- displays a summary of traps manipulated by command files

FIGURE 8. Report AOPRPT02

<table>
<thead>
<tr>
<th>TRAP NAME</th>
<th>TRAP TYPE</th>
<th>TRAP ARGUMENT</th>
<th>COMMAND FILE</th>
<th>ACTION</th>
<th>LIBRARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>@1200</td>
<td>TOD</td>
<td>*</td>
<td>MORNING</td>
<td>ADD/ENA</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EVENING</td>
<td>DIS</td>
<td>2</td>
</tr>
<tr>
<td>@700</td>
<td>WTO</td>
<td>$HASP*</td>
<td>NOON</td>
<td>ENA</td>
<td>1</td>
</tr>
</tbody>
</table>

Traps Added: 99999
Enabled: 99999
Disabled: 99999
Deleted: 99999

Command Files Processed: 99999

Libraries: 1 = USER.COMMAND.LIB1
2 = USER.COMMAND.LIB2

Command file libraries: 2
Report on Traps Executing Command Files

This report, AOPRPT03:

- lists all traps alphabetically by trap name
- for each trap, lists any command file it executes directly by its ACTION or ALTACT operand
- displays a summary of traps executing command files

FIGURE 9. Report AOPRPT03

<table>
<thead>
<tr>
<th>TRAP NAME</th>
<th>TRAP TYPE</th>
<th>TRAP ARGUMENT</th>
<th>EXECUTES</th>
<th>LIB</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRAP001</td>
<td>TOD</td>
<td>*</td>
<td>STARTUP</td>
<td>1</td>
</tr>
<tr>
<td>TRAP025</td>
<td>WTO</td>
<td>IEF*</td>
<td>CHECK</td>
<td>3</td>
</tr>
</tbody>
</table>

Command Files Executed: 99999

Libraries:
1 = USER.COMMAND.LIB1
2 = USER.COMMAND.LIB2
3 = USER.COMMAND.LIB3

Command file libraries - 3
Report on Command Files Executed by Traps

This report, AOPRPT04:

- lists alphabetically all command files that are executed directly by trap ACTION or ALTACT operands
- for each command file, lists the trap that executes it
- displays a summary of command files executed by traps

FIGURE 10. Report AOPRPT04

<table>
<thead>
<tr>
<th>COMMAND FILE</th>
<th>LIB</th>
<th>EXECUTED TRAP</th>
<th>TYPE</th>
<th>TRAP ARGUMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECK</td>
<td>3</td>
<td>TRAP025</td>
<td>WTO</td>
<td>IEF*</td>
</tr>
<tr>
<td>STARTUP</td>
<td>1</td>
<td>TRAP001</td>
<td>TOD</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IFLTRAP</td>
<td>CMD</td>
<td>S JES2</td>
</tr>
</tbody>
</table>

Command Files Executed : 99999

Libraries: 1 = USER.COMMAND.LIB1  
           2 = USER.COMMAND.LIB2  
           3 = USER.COMMAND.LIB3

Command file libraries - 3
## Report on Trap Definitions Sorted by Argument

This report, AOPRPT05:

- lists all traps alphabetically by trap argument, (that is, by the text pattern that will trigger the trap)
- for each trap, lists the type of action it takes when triggered
- displays a summary of trap definitions

### FIGURE 11. Report AOPRPT05

<table>
<thead>
<tr>
<th>Trap Name</th>
<th>Trap Type</th>
<th>Trap Argument</th>
<th>Trap Action</th>
<th>Trap Lib</th>
</tr>
</thead>
<tbody>
<tr>
<td>@700</td>
<td>WTO</td>
<td>$HASP*</td>
<td>EX CFILE &amp;DATA</td>
<td>1</td>
</tr>
<tr>
<td>@1200</td>
<td>TOD</td>
<td>*</td>
<td>SH VARS</td>
<td>1</td>
</tr>
</tbody>
</table>

Command Files Processed: 99999

Libraries: 1 = USER.COMMAND.LIB1
            2 = USER.COMMAND.LIB2

Command file libraries - 2
Cross-Reference Facility Error Report

When you generate a report on the cross-reference facility, AF/OPERATOR examines each trap in the library for errors such as syntax errors. It does not list these defective traps in the body of the report, but it lists them in an error report like the one in Figure 12.

FIGURE 12. Cross-Reference Facility Error Report

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>TRAP</th>
<th>TRAP FILE</th>
<th>TRAP LIB</th>
<th>FUNCTION</th>
<th>NAME</th>
<th>REASON OF ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST</td>
<td>1</td>
<td>ADD</td>
<td>TEST</td>
<td>1</td>
<td>AOOCSX</td>
<td>Syntax error in ACTION parm</td>
</tr>
<tr>
<td>ADD</td>
<td>TRAP1</td>
<td>Syntax error in TRAP type</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>TRAP2</td>
<td>Syntax error in ACTION parm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>TRAP3</td>
<td>Syntax error in TRAP type</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>AOOCS6</td>
<td>Syntax error in ACTION parm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>AOOCS8</td>
<td>Syntax error in ACTION parm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>AOOCSX</td>
<td>Syntax error in ACTION parm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Total Libraries Processed: 1
Total Command Files Processed: 5
Total TRAP Commands Processed: 54
TRAP Errors Encountered: 7

Libraries: 1 = TDBB09.AFO.SAMPLIB

To obtain a complete list of traps in the body of the report, first correct the trap errors and then generate the report again.
Running the Cross-Reference Reports

The AOXREF member in the hlq.TKANSAM dataset contains sample JCL to run the cross-reference reports for CFILE command files (see note 2). Verify the correct dataset name for your installation. You must edit this sample JCL as shown in the following figure.

Notes:

1. The line numbers preceding each line of the sample JCL shown below are for reference only; your editor screen will appear slightly different.
2. AOXREF does not support REXX. If the RKANCMD dataset contains any REXX programs, various errors will appear on the SYSPRINT report.

FIGURE 13. JCL to be Edited to Run the Cross-Reference Reports

Make the following changes to the sample JCL:

**Line 01:** Replace cccccccc with a valid job statement for your site.

**Line 03:** CANDLE.TKANMOD is the load library dataset unloaded from tape. Replace CANDLE.TKANMOD with your site’s dataset name.

**Line 04:** You may use one or more input command libraries with your RKANCMD concatenation. Change CANDLE.RKANCMD to your site’s actual dataset name.

**Line 06:** If you want the reports to be sorted in main memory, omit this DD statement. (This is practical only for small reports.)
Running the Cross-Reference Reports
Testing a Simulated Environment

Introduction

OG*TSO is a simulated version of an AF/OPERATOR console that runs under TSO. It is a safe environment, separate from your production system, in which to learn about AF/OPERATOR. It is useful for developing and testing REXX execs you write. You can create traps and execute REXX execs, watch the results display on the screen of your TSO terminal, and test your solutions before putting them into your production system.

Some commands under OG*TSO are simulated while others work just as they do with your AF/OPERATOR started task.

**Important**

Within an OG*TSO environment, any session (such as a POVI, OMEGAMON, or OMEGAVIEW session) issued with the LOGON command is a fully-functional session and all subsequent procedures will behave in a normal fashion. That is, even though OG*TSO is a “simulation” of AF/OPERATOR, the sessions which it initiates with the LOGON are real.

The other ways in which OG*TSO differs from a real AF/OPERATOR session are listed in “Guidelines for Using OG*TSO” on page 92.

Chapter Contents

- The OG*TSO REXX exec. .......................................................... 84
- Configuring OG*TSO. .............................................................. 85
- OG*TSO Console Simulation. .................................................. 88
- Testing REXX execs using OG*TSO ........................................ 90
- Guidelines for Using OG*TSO .................................................. 92
OG*TSO is invoked by executing a REXX exec named KOGTSO.

The KOGTSO REXX exec is generated by the Customizer and is initially located in a dataset named `rhilev.@system.RKANSAM`, where `rhilev` is the high-level qualifier used for the runtime JCL library created during the AF/OPERATOR installation and customization process.

Copy this member to a dataset in the SYSPROC concatenation for your TSO session.

### Accessing OG*TSO from ISPF

You access OG*TSO by executing the KOGTSO REXX exec from an ISPF session. For example,

- To execute the KOGTSO REXX exec from an ISPF input field, type
  
  ```
  TSO %KOGTSO
  ```
  
  and press Enter.

**Result:** The OG*TSO Startup Parameters panel appears.

### Exiting OG*TSO

To exit OG*TSO and terminate the current session, press your PF END key.
Configuring OG*TSO

Introduction

You only need to configure OG*TSO once. You can, however, change any of the configuration information at any time.

If you need to provide configuration data to OG*TSO, follow the steps below, otherwise proceed to “OG*TSO Console Simulation” on page 88.

OG*TSO Startup Parameters panel

The first time you access OG*TSO, before it has been configured, the OG*TSO Startup Parameters panel appears as shown below. OG*TSO requires you to specify a number of datasets on this panel. You must create the datasets yourself, OG*TSO does not perform this task for you. You may need to consult the person who customized AF/OPERATOR at your site about dataset names.

<table>
<thead>
<tr>
<th>!Candle</th>
<th>Version nnn</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMAND ====&gt;</td>
<td>OG*TSO STARTUP PARAMETERS</td>
</tr>
<tr>
<td>MESSAGE LOG ===&gt; SYSOUT(X)</td>
<td></td>
</tr>
<tr>
<td>COMMAND FILES ====&gt;</td>
<td></td>
</tr>
<tr>
<td>======</td>
<td></td>
</tr>
<tr>
<td>====&gt;</td>
<td></td>
</tr>
<tr>
<td>CHECKPOINT ====&gt;</td>
<td></td>
</tr>
<tr>
<td>PARAMETERS ====&gt;</td>
<td></td>
</tr>
</tbody>
</table>

OPTIONAL STARTUP COMMAND:
COMMAND NAME ===>
ARGUMENTS ===>
Configuring OG*TSO

Specifying OG*TSO parameters

To configure OG*TSO, specify the following information on the OG*TSO Startup Parameters panel:

**MESSAGE LOG**

Specify sysout in the form SYSOUT(x) or specify a fully-qualified dataset name to receive messages OG*TSO may produce. If you specify a dataset name, enclose it in single quotes.

*Note:* If your AF/OPERATOR uses the DISKMLOG startup parameter, the OGMSGLnn ddnames must be preallocated to a message log dataset before invoking KOGTSO. They will override the specification you make on the OG*TSO panel.

**COMMAND FILES**

Specify at least one fully-qualified dataset name to contain the REXX execs you want to test. Enclose the dataset names in single quotes.

You can concatenate dataset names. At the top of the concatenation list, you might specify a personal library dataset you have created for your own use. For example:

`'youruserid.OGTEST.REXXLIB'`

Using your own library protects your production system library from inadvertent modification of tested execs.

The dataset at the base of the concatenation list can be an AF/OPERATOR test REXX exec library. You may specify the sample library that accompanies AF/OPERATOR and test directly from sample execs, however Candle recommends that you preserve a “vanilla version” of the sample library and make modifications in another dataset.

**CHECKPOINT**

Specify a fully-qualified checkpoint dataset name that is a copy of the dataset created by AF/OPERATOR during customization. The allocation parameters of the two datasets must be identical.

**Important**

Do not use the production system checkpoint dataset.

You can create a copy of the production system checkpoint dataset by issuing the following TSO command:

```
DEFINE CLUSTER (NAME('usrhilo.RKOGCKP') +
                MODEL('rvhilev.@system##.RKOGCKP'))
```

where `usrhilo` is a high-level qualifier you will use for the dataset you are creating, and `rvhilev.@system##.RKOGCKP` is the name of the checkpoint dataset created during AF/OPERATOR installation and customization for a specific system and address space.
When you have finished configuring the OG*TSO startup parameters, press Enter. There may be a delay while OG*TSO allocates datasets and SYSOUT files to use in the simulation. OG*TSO displays a message indicating that initialization is in process. You may receive explanatory messages if there is a problem with the way the OG*TSO Startup Parameters panel is filled in. For example, this panel expects the names of fully-qualified datasets to appear in quotes. If you receive the message:

```
IEC130I RKANPAR DD STATEMENT MISSING
```

it indicates that you did not specify a dataset for the Startup Parm DSN field. Press Enter, and the simulation will run with default Candle AF/OPERATOR startup parameters.

When OG*TSO is successfully initialized, the OG*TSO Console Simulation panel appears.

---

**rvhilev**

High-level qualifier for the runtime VSAM datasets.

**system**

SMFID of the MVS system where an AF/OPERATOR address space was configured.

**##**

AF/OPERATOR address space identifier assigned by the Customizer.

You may need to consult the person who customized AF/OPERATOR on your MVS system.

**PARAMETERS**

Optional. Specify the fully-qualified dataset name and the member name that contains the AF/OPERATOR startup parameters that were configured for this copy of AF/OPERATOR during customization. Enclose the entire name in quotes.

The copy generated by the Customizer is initially located in a dataset named `rhilev.RKANCMD(OGPRM##)`. If you leave this field blank, default Candle startup parameters will be used.

**COMMAND NAME**

Optional. Specify a REXX exec (in the command files dataset concatenation) to execute at OG*TSO startup, or leave this field blank.

**ARGUMENTS**

Optional. Specify arguments to be passed to the COMMAND NAME exec during OG*TSO startup, or leave this field blank.
OG*TSO Console Simulation

Introduction

This section introduces you to the OG*TSO environment. After you configure the OG*TSO startup parameters, the OG*TSO Console Simulation panel appears.

OG*TSO Console Simulation panel

The OG*TSO Console Simulation panel simulates an operator’s console in a TSO session. The top portion displays messages from AF/OPERATOR that would normally go to the operator’s console. AF/OPERATOR startup messages are included on this display.

Unlike a real console, you can scroll up, down, left, and right with your up, down, left, and right PF keys. Pressing Enter refreshes the panel. The MAX (M) scrolling option works in OG*TSO. Scroll amount field values can be: PAGE, HALF, or CSR.

You can type simulated AF/OPERATOR commands on the Command ===> line at the bottom of the panel and scroll down to see the results immediately displayed. The maximum command length is 100 bytes. Commands longer than 100 bytes are truncated.

You must preface all simulated AF/OPERATOR commands entered from the Command ===> line with a subsystem identifier. For example, enter:

   O120 SHOW STATS

where O120 is the subsystem name of this AF/OPERATOR address space, to display various AF/OPERATOR statistics.

You can issue OG*TSO simulated WTOs from the Command ===> line by entering the WTO command as follows:
WTO 'message text'

If you issue the AF command with no operand, as follows:

O120 AF

information about your OG*TSO session is displayed. In that display, the console ID is automatically set as follows: CID(OG*TSO) to remind you that this is an AF/OPERATOR console simulation under TSO.
Testing REXX execs using OG*TSO

Introduction
To execute a REXX exec in OG*TSO, the REXX exec must reside in one of the command files datasets you specified on the OG*TSO Startup Parameters panel. Also, be sure that the exec you are testing is not preceded, higher in the dataset concatenation, by an exec of the same name.

Testing a trap
Follow this procedure to execute a REXX exec that contains a trap in OG*TSO and issue a simulated WTO message to the panel (to determine if the trap is working).

Assume you have created a short REXX exec to execute a trap. The exec is named TSHSP373, and its contents are as follows.

/* REXX */
ADDRESS AFHOST
''
TRAP ADD(TSHSP373) WTO(''$HASP373'') ENABLE SUPPRESS''

1. To execute the exec in OG*TSO, type on the command line:

   O120 EX TSHSP373

   where O120 is the subsystem name of this AF/OPERATOR address space, and press Enter.

2. Read the AF/OPERATOR messages on your panel that indicate this trap’s status. One message should say:

   !AOP0171 WTO TRAP TSHSP373 ADDED (ENABLED)

3. To verify the contents of the trap, type:

   O120 SHOW TRAP(TSHSP373)

   and press Enter. Look at the COUNT field. It logs the number of times the trap triggered (matched). The count should be (0) because AF/OPERATOR has not yet matched the WTO specified in the trap. (You may have to scroll right to see the COUNT field.)

4. To issue a simulated WTO message to the panel, type:

   WTO ' '$HASP373 ANYONE'

   and press Enter.

   The OG*TSO WTO command simulates a real WTO sent to the console. This message does not display on the panel because you instructed AF/OPERATOR to suppress it in trap TSHSP373.

5. To issue a WTO message to the panel that AF/OPERATOR does not suppress, type:

   WTO ' '$HASP999 ANYONE'

   and press Enter.

   The message displays on your panel because you did not instruct AF/OPERATOR to suppress it.
6. To verify that AF/OPERATOR has performed the specified actions, check the COUNT field of the trap as follows:
   A. Type:
      
      O340 SHOW TRAP(TSHSP373)
      
      and press Enter.
   B. Check the COUNT field. You may have to scroll right to see it. AF/OPERATOR logs the number of times the trap has triggered (matched). The count should be (1).

7. If the count is not (1), do the following:
   A. Check the message text of the WTO in your trap to verify that you typed it correctly.
   B. Make sure you enclosed the message text in single quotes.

Testing complex REXX execs

You can use OG*TSO to test complex REXX execs or to recreate a problem. Sometimes you will need to modify the code slightly for testing in OG*TSO. For example, OPER RESP cannot return actual console data, because OG*TSO is confined to the TSO address space. Therefore, the expected contents of the LINE variables must be explicitly set in your code before any statements that use those variables. If you are issuing an OPER RESP for a D NET,MAJNODES, you need to predefine LINE variables to contain the data VTAM returns from a D NET command. Remember to preset a LINE# value that corresponds to the number of LINE variables you select for testing purposes.

A useful method for setting up the LINE variables with valid message data is to copy the messages from RKOGLOGM into a dataset, then create a REXX exec that declares those variables, using the captured message data. The exec can be called before the LINE variables are used. Remember that OG*TSO messages are sent to the location specified on the OG*TSO Startup Parameters panel, either a dataset or a SYSOUT class.
Guidelines for Using OG*TSO

Introduction

Not all AF/OPERATOR commands, variables, functions, or test environments are available under OG*TSO.

Limitations of OG*TSO

These are the differences between real AF/OPERATOR sessions and OG*TSO console simulation sessions.

- The following AF/OPERATOR components are not supported under OG*TSO.
  - Multi-System Management Facility
  - NetView Interface
  - AOSIM
  - JES3 exits
  - CICS exit
  - IMS exit
- The variable AOTSUID is not available when using OG*TSO.
- ACTIVE may be misleading when used in OG*TSO. The value it returns reflects the tasks running on the MVS production system, not tasks running under OG*TSO.
- REPID used in OG*TSO returns a non-zero return code.
- JOBSTEP may be misleading when used in OG*TSO. The value it returns reflects the tasks running on the AF/OPERATOR production system.
- Certain AF/OPERATOR functions (for example, WTO, WTOR, and OPER) do not behave quite the same under OG*TSO as they do in native-mode AF/OPERATOR because they are being simulated. For example,
  - OG*TSO converts WTORs to WTOs. Automation routines which are dependent on a reply may loop.
  - OG*TSO converts OPER to a WTO. No actual command is issued to the system.
  - WTOs issued from a REXX exec in OG*TSO cannot be trapped.
Introduction

This chapter describes the communication environment and protocols supported by AF/OPERATOR. General information relevant to the environment is presented first, followed by information specific to each of the connection types: APPC, XCF, and TCP/IP.

Chapter Contents

- Overview of the Communications Environment ............................................. 94
- Features of the AF/OPERATOR Communication Services Interface ................. 95
- Session Types .................................................................................................. 98
- Connecting AF/OPERATOR Systems ............................................................... 101
- Connecting AF/OPERATOR to AF/REMOTE for Windows NT ....................... 106
- Using the COMSDRCV Function ................................................................. 109
- REXX Error Recovery .................................................................................. 114
- Configuration Requirements for Establishing APPC Connections .................. 116
- Configuration Requirements for Establishing XCF Connections ..................... 120
- Configuration Requirements for Establishing TCP/IP Connections ................. 122
- Advanced Communications--AF TCP/IP Communications Protocol .............. 124
- Advanced Communications--Creating a Simple Client Application ............... 129
- Advanced Communications--Open TCP/IP communications ........................ 146
- Automated LINK management ...................................................................... 148
Overview of the Communications Environment

In the AF/OPERATOR REXX environment, you can send commands and, optionally, receive responses from a remote partner. Depending on the capabilities of the remote partner, the connection can be made using APPC, XCF, or TCP/IP.

This allows AF/OPERATOR to communicate with other systems that have this feature, including:

- AF/OPERATOR systems
- AF/REMOTE®

A Candle-supplied REXX API, COMSDRCV, gives execs running under AF/OPERATOR access to components which manage inbound and outbound communication tasks. (You can find detailed information about COMSDRCV and review its format and use in the AF/OPERATOR Command Language Reference Manual.)

Peer-to-peer communication between AF/OPERATOR systems can be established using either APPC, XCF, or TCP/IP protocols. AF/OPERATOR systems can send WTOs, MLWTOs, WTORs, AF/OPERATOR commands, and subsystem commands with responses with minimal changes to existing automation routines. Using these sessions for peer-to-peer communication increases system performance by allowing AF/OPERATOR to handle multiple simultaneous communication tasks.

For example, an XCF connection can be established between two AF/OPERATOR systems in a SYSPLEX before VTAM is available, or when VTAM simply is unavailable.

Note to MSMF users

If you are using MSMF to communicate between AF/OPERATOR peers, review the information in Appendix B, “Using MSMF to Communicate between AF/OPERATOR Peers” on page 293. If you define the same linkids using the AF/OPERATOR communications facilities described in this chapter, and these linkids are those currently used by the older Multi-System Management Facility (MSMF), AF/OPERATOR will use the communication facilities in this chapter in preference to using MSMF. For example, if your exec contains the line:

```
WTO 'good morning' SYSID(SYSB)
```

and an APPC session called SYSB is active, AF/OPERATOR sends the WTO over the APPC connection.

Client and server

This chapter uses the terms client and server when discussing communication links between systems.

An AF/OPERATOR server can receive incoming requests for communication services initiated by other AF/OPERATOR systems or applications.

An AF/OPERATOR client can initiate outbound requests to establish communications.
Features of the AF/OPERATOR Communication Services Interface

Introduction

AF/OPERATOR interface support consists of the following features:

- Communication component
  Enables AF/OPERATOR to function as a communication server. The communication component manages requests for communication services from client systems.

- Link component
  Enables AF/OPERATOR to function as a client. The link component defines, requests, and manages communication sessions with server systems.

- REXX API (application programming interface) component
  A Candle-supplied COMSDRCV REXX function running under AF/OPERATOR that can send and receive data using the communication and link components.

- Trace facility
  Provides diagnostic information for the communication and link components.

- Communication match
  This is an AF/OPERATOR match type specific to communications.

The components listed above have associated commands or functions that are explained in detail elsewhere in this manual.

Note: All commands and functions in this chapter are supported only in the AF/OPERATOR environment.

Communication component commands

The AF/OPERATOR communication component responds to the following commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMM START</td>
<td>Initiates an AF/OPERATOR-controlled communication server.</td>
</tr>
<tr>
<td>COMM STOP</td>
<td>Terminates AF/OPERATOR-controlled communication servers.</td>
</tr>
<tr>
<td>COMM DISPLAY</td>
<td>Displays AF/OPERATOR server status information.</td>
</tr>
<tr>
<td>COMM SUSPEND</td>
<td>Temporarily disallows AF/OPERATOR servers from accepting new client sessions.</td>
</tr>
<tr>
<td>COMM RELEASE:</td>
<td>Releases temporarily suspended AF/OPERATOR servers.</td>
</tr>
</tbody>
</table>
Link component commands
The AF/OPERATOR link component responds to the following commands.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINK DEFINE</td>
<td>Defines a client communication session.</td>
</tr>
<tr>
<td>LINK START</td>
<td>Initiates client communication sessions.</td>
</tr>
<tr>
<td>LINK STOP</td>
<td>Terminates client communication sessions.</td>
</tr>
<tr>
<td>LINK DELETE</td>
<td>Stops and deletes client communication sessions and definitions.</td>
</tr>
<tr>
<td>LINK DISPLAY</td>
<td>Displays status of communication sessions.</td>
</tr>
</tbody>
</table>

REXX API functions
The following REXX application programming interface functions enable AF/OPERATOR to send and receive data during a communication session.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMSDRCV</td>
<td>Sends or receives data from a remote system.</td>
</tr>
<tr>
<td>LNKCHECK</td>
<td>Determines whether a specified session is active.</td>
</tr>
</tbody>
</table>

Trace facility
The AF product control command provides a communications trace option.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AF COMTRACE</td>
<td>Activates or deactivates a communication trace and displays communication trace information.</td>
</tr>
</tbody>
</table>
**Communications match**

A communications match is an AF/OPERATOR match that has been scheduled from another system using the COMSDRCV REXX function with a *datatype* specification of EXEC. The communication match (COM) can be displayed using the AF/OPERATOR SHOW MATCHES command.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHOW MATCHES COM</td>
<td>Displays communication matches.</td>
</tr>
</tbody>
</table>

The following is an example of this command embedded in a REXX EXEC:

```rexx
“SHOW MATCHES COM”
```

A typical response to this command follows:

```
!AOP0201 MAT 14    (SYSB) CLASS() ASID(0) AUTH(0) LINKID(SYSB1)
!AOP0201 -------- COM: OPER ‘O999 SHOW MAT’ RESP
!AOP0201 -------- TRAP(OGSUBSYS) CONTINUE
!AOP0201 -------- CLIENT(AFRMT)
```
Session Types

Overview

A session is a connection established between two systems for the purpose of sending data. Underlying the specific protocol, the actual connection can be between VTAM LUs (logical units), XCF members, or TCP/IP sockets. This section describes the types of sessions that can be established using one of the communication services available under AF/OPERATOR. The LINK DEFINE command defines an AF/OPERATOR-initiated communication session suitable for sending commands from a client to a server AF/OPERATOR.

Sendonly sessions

The term send only is used to describe these sessions, since a client cannot process any inbound request or unsolicited response. All XCF and TCP/IP sessions, as well as the default APPC session, are send only. When a client AF/OPERATOR initiates this session using a LINK START command, the communication component on the AF/OPERATOR server automatically initiates an internal session which is used to return responses to the client.

When an APPC communication session is initiated by AF/REMOTE, the internal session is optional.
Internal sessions

An AF/OPERATOR internal session is always associated with a send only session. The internal session is used to return responses from an AF/OPERATOR system to the originating system. It is AF/OPERATOR-controlled. You can not start it, stop it, or use it to send your own data. You can, however, see its linkid displayed when you issue the COMM DISPLAY command.

The following example shows the the partial results of issuing the COMM DISPLAY command:

```
COMM DISPLAY
!AOP3765 SERVER TPNAME(TEST1) LLUNAME(HIGHWAY1) ACTIVE
!AOP3743 ---- PLUNAME - USCACO01.HIGHWAY6
!AOP3743 ---- SERVER - SYSB1 (SYSB)
!AOP3743 ---- CLIENT - SYSG
!AOP3743 ---- INTERNAL SEND ONLY ID - #0000002
```

The last line of this partial display shows the linkid of the internal session associated with the AF/OPERATOR server whose TPNAME is TEST1.

The way the AF/OPERATOR internal session is established varies according to the communications protocol.

APPC internal sessions require that an APPC server has been established on the system from which the client is connecting. This must be done explicitly by the user’s issuing a COMM START command prior to performing the LINK START. The internal session can then be established as a client of this server.

An XCF internal session is established as a client of an XCF internal server, which is automatically started by AF/OPERATOR when the first LINK START is performed. AF/OPERATOR starts this internal server in the same XCF group as the link being started, and identifies it by using the reserved member name, #INTERNL. It is not necessary to perform a COMM START on the client side of an XCF connection before issuing the LINK START.

A TCP/IP internal session is established using the same logical connection as the original send only session, thus exploiting the duplex nature of the connection. It is not necessary to perform a COMM START on the client side of a TCP/IP connection before issuing the LINK START.
Investigating session status

You can further investigate the status of any session by issuing a LINK DISPLAY command, for example:

```
LINK DISPLAY LINKID(#0000002)
```

AF/OPERATOR reserves linkids beginning with a # (pound or hash sign) in the range #0000001 to #9999999 for internal use.

**Note:** Although a TCP/IP internal session does use one of the reserved linkids, that linkid cannot be displayed using the LINK DISPLAY command, since it is not a separate entity from the incoming connection.

SENDRCV sessions (APPC only)

AF/OPERATOR communication links can be defined using the optional SENDRCV parameter which provides an additional session on the link. A SENDRCV session sends data then waits for a response on the same session. This option is provided for internal use between AF/OPERATOR and its application programs. Most operations you may want to perform are more efficiently accomplished using the default, send only session type.

```
+-------------------------+                +-------------------------+
|  AF/OPERATOR System A  |                |  AF/OPERATOR System B  |
|                     |                |                     |
|                Server A |                | Client B             |
|           +-------------|   send only    |-------------+           |
|           |             |<---------------|             |           |
|           |  COMM       |- - - - - - - ->| LINK DEFINE |           |
|           |  Component  |   internal     | LINK START  |           |
|           |             |<-------------->|             |           |
|           +-------------|    SENDRCV     |-------------+           |
|                     |                |                     |
|                     |                |                     |
+-------------------------+                +-------------------------+
```
Connecting AF/OPERATOR Systems

Introduction

This section explains how to set up communication links between two AF/OPERATOR systems so that each can send commands to the other and receive responses. You can link any number of AF/OPERATOR systems in this manner.

The process has 3 basic steps.

1. Initiate the communication component on each system using the COMM START command.
2. Define and start the links between the systems using the LINK DEFINE and LINK START commands.
3. Once an originating AF/OPERATOR system has established an outbound communication link to another AF/OPERATOR, that link can be used to establish an inbound communication link from the other system.

Note: This technique can be used to start communication links to any number of AF/OPERATOR systems, from one location.

Naming convention used in this example

The AF/OPERATOR systems in this example use a naming convention in which the first 4 characters of all system IDs, linkids, transaction program names, and local logical unit names, clearly identify the AF/OPERATOR system. For example, AF/OPERATOR system A is identified by the following names.

SYSID: SYSA
LINKID: SYSAO999
TPNAME: SYSATP
LLUNAME: SYSALLU

AF/OPERATOR system B is identified by the following names.

SYSID: SYSB
LINKID: SYSBO999
TPNAME: SYSBTP
LLUNAME: SYSBLLU
Connecting AF/OPERATOR Systems

**Bringing up the communication servers**

An AF/OPERATOR system must have an active communication component (a server) running in order to receive a command request from another system. So the first task when connecting AF/OPERATOR systems is to define a server on each. This is done using the COMM START command, which uniquely identifies the AF/OPERATOR communication server. Execute a REXX exec, similar to the following, on system A to define and activate its communication server using the APPC communication protocol.

```rexx
/* REXX */
ADDRESS AFHOST
"COMM START TYPE(APPC) TPNAME(SYSATP) LLUNAME(SYSALLU)"
RETURN rc
```

Execute the following exec, on system B, to activate its communication server using the APPC communication protocol.

```rexx
/* REXX */
ADDRESS AFHOST
"COMM START TYPE(APPC) TPNAME(SYSBTP) LLUNAME(SYSBLLU)"
RETURN rc
```

Both execs can be executed manually during the initial development phase. Later, you can incorporate the COMM START command into the exec that is automatically invoked at system startup to establish the initial environment for each AF/OPERATOR. See the COMMAND startup parameter in the *AF/OPERATOR Configuration and Customization Guide*.

**Linking the systems**

The next task is to define the communication links between the systems. This is done using the LINK DEFINE command. This example defines two communication links between AF/OPERATOR systems A and B. One from system A to system B, and a similar link from system B to system A. This allows each system to send commands to, and receive responses from, the other. The links are initiated using the LINK START command.

```rexx
/* REXX */
ADDRESS AFHOST
"COMM START TYPE(APPC) TPNAME(SYSATP) LLUNAME(SYSALLU)"
RETURN rc
```
Connecting AF/OPERATOR Systems

Sample exec LINKDEFB (running on system A) defines and starts an APPC communication session from system A to system B.

```rexx
/* REXX */
ADDRESS AFHOST
  "LINK DEFINE LINKID(SYSBO999) TYPE(APPC) TPNAME(SYSBTP)",  
    "MODENAME(LU62) PLUNAME(SYSBLU)"
IF rc <> 0 THEN
  SAY 'LINK DEFINE UNSUCCESSFUL FOR LINK TO SYSBO999'
ELSE
  "LINK START LINKID(SYSBO999)"
IF rc <> 0 THEN
  DO
    SAY 'LINK START UNSUCCESSFUL'
    "LINK DELETE LINKID(SYSBO999)"
  END
RETURN RC
```

Each LINK DEFINE command identifies the server with which you will communicate, and provides the information AF/OPERATOR needs to establish a communication session. Any number of defined links can be initiated with one LINK START command.

You can manually execute a REXX exec on system B to define the link from system B to system A, or, once the send only session from A to B is started, you can use it to execute a REXX exec on system B.
Starting all links from one location

You can set up the AF/OPERATOR systems in your complex so that one AF/OPERATOR controls the APPC communication links to all the other AF/OPERATOR systems. One REXX exec on a local AF/OPERATOR system can establish outbound communication links to all other AF/OPERATOR systems, then execute REXX execs on the other systems to establish inbound links back to the originating system. To illustrate, system A will be used to remotely control system B.

Sample exec STARTALL, running on system A, calls LINKDEFB (shown in the previous example) to initiate the communication link from A to B. Once this send only session is started, the AF/OPERATOR-supplied REXX API function, COMSDRCV, is used to execute the LINKDEFA exec on system B.

```/* REXX */
ADDRESS AFHOST
"EX LINKDEFB"
DATA = 'EX LINKDEFA'
STATUS = COMSDRCV('SYSB0999',DATA,'EXEC',12)
IF STATUS <> 0 THEN
   SAY 'UNSUCCESSFUL COMSDRCV'
ELSE
   IF COMDATA.1 ^= 'OK' THEN
      DO
         SAY 'LINK FROM SYSTEM B TO SYSTEM A NOT STARTED SUCCESSFULLY'
         "LINK DELETE LINKID(SYSB0999)"
      END
   ELSE
      SAY 'LINK FROM SYSTEM B TO SYSTEM A STARTED SUCCESSFULLY'
.. .. ..
```

Sample exec LINKDEFA, running on system B, defines a communication link from system B to system A and uses the COMSDRCV function to return response data back to the REXX exec on the originating system (the STARTALL exec on system A).
/* REXX */
ADDRESS AFHOST
"LINK DEFINE LINKID(SYSAO999) TYPE(APPC) TPNAME(SYSATP)
MODENAME(LU62),
"PLUNAME(SYSALLU)"
IF rc <> 0 THEN
  DO
    SAY 'LINK DEFINE UNSUCCESSFUL FOR LINK TO SYSAO999'
    DATA = 'NOTOK'
    STATUS = COMSDRCV(,DATA,'DATARPLY')
    IF STATUS <> 0 THEN SAY 'UNSUCCESSFUL DATARPLY'
  END
ELSE
  DO
    "LINK START LINKID(SYSAO999)"
    IF rc <> 0 THEN
      DO
        SAY 'LINK START UNSUCCESSFUL'
        "LINK DELETE LINKID(SYSAO999)"
        DATA = 'START NOT OK'
        STATUS = COMSDRCV(,DATA,'DATARPLY')
        IF STATUS <> 0 THEN SAY 'UNSUCCESSFUL DATARPLY'
      END
    ELSE
      DO
        DATA = 'OK'
        STATUS = COMSDRCV(,DATA,'DATARPLY')
      END
    END
  END
RETURN RC
**Connecting AF/OPERATOR to AF/REMOTE for Windows NT**

**Overview**

Communications between AF/OPERATOR and AF/REMOTE for Windows NT is accomplished by means of a TCP/IP connection. There is actually no real difference between establishing a TCPIP connection between two AF/OPERATOR systems and this scenario involving AF/REMOTE.

You can find additional information about connectivity between AF/REMOTE and AF/OPERATOR using TCP/IP peer-to-peer connectivity in the *AF/REMOTE on Windows NT User’s Guide* in the chapter entitled “Peer-to-Peer Communications.”

**AF/OPERATOR-initiated sessions**

If AF/OPERATOR is to initiate a client session with AF/REMOTE:

1. The AF/REMOTE TCP/IP server must be active and listening for connections. The local hostname from which AF/OPERATOR is connecting must be in the trusted hostnames list for AF/REMOTE. For more information, refer to the Candle Corp. library for AF/REMOTE.

2. The communication session to AF/REMOTE must be defined on the AF/OPERATOR system using a LINK DEFINE and is initiated using a LINK START command.

```rexx
/* REXX */
ADDRESS AFHOST
"LINK DEFINE LINKID(RMT1) TYPE(TCPIP)",
   "HOSTNAME(111.112.113.114) PORTNUMBER(1040)"
IF rc <> 0 THEN
   SAY 'LINK DEFINE UNSUCCESSFUL FOR LINK TO RMT1'
ELSE
   DO
      "LINK START LINK(RMT1)"
      IF rc <> 0 THEN
         DO
            SAY 'LINK START UNSUCCESSFUL'
         END
   END
   .
   .
RETURN rc
```

You can send command requests to AF/REMOTE from a REXX exec using the AF/OPERATOR COMSDRCV REXX function. To send a command to AF/REMOTE using COMSDRCV, specify a *datatype* of EXEC.
AF/REMOTE-initiated sessions to AF/OPERATOR

If AF/REMOTE is to initiate the session with AF/OPERATOR:

1. the AF/OPERATOR system’s communication component must be active to receive the request. The local hostname from which AF/REMOTE is connecting must be in the trusted hostnames list for AF/OPERATOR.

2. The communication link to AF/OPERATOR must then be defined and started on the AF/REMOTE system.

   /* REXX */
   ADDRESS AFHOST
   “COMM START TYPE(TCPIP) PORTNUMBER(1111)”
   RETURN rc

The communication session can then be initiated from AF/REMOTE.
Using the COMSDRCV Function

About the REXX API

After you have established communication between systems, use the COMSDRCV REXX API function (in REXX execs running under AF/OPERATOR) to send commands and, optionally, receive response data. The COMSDRCV function allows AF/OPERATOR to manage communication tasks for you.

You can use COMSDRCV to:

- Execute an AF/OPERATOR command or a REXX exec on another AF/OPERATOR system and receive a response.
- Issue a WTOR, MVS command, or subsystem command and receive a response.
- Execute commands or execs on AF/REMOTE and, optionally, receive responses.

Issuing an AF/OPERATOR command on another AF/OPERATOR

The following exec uses the COMSDRCV function to issue the SHOW MAT command on a remote AF/OPERATOR system identified by linkid SYSG1.

```/* REXX */
DATA = 'SHOW MAT'
STATUS = COMSDRCV('SYSG1', DATA, 'EXEC')
IF STATUS <> 0 THEN
   SAY 'RESULT = ' STATUS
RETURN STATUS```

Receiving a response to an AF/OPERATOR command

The following exec issues the AF/OPERATOR command, SHOW MAT, on a remote AF/OPERATOR system whose subsystem name is O999 and whose linkid is SYSGO999. The AF/OPERATOR OPER command with the RESP option is used to return data (the SHOW MAT command’s display) from the remote system. The number 2000 specifies the length, in bytes, of data to return. Up to 2000 bytes of the SHOW MAT command’s display will be returned to the originating system and stored in REXX variable COMDATA.1.
When response data is obtained from executing an AF/OPERATOR command on another system, each 150 bytes represent 1 line.

Another example of receiving a response uses the REDIRECT() function:

```rexx
/* REXX */
DATA = 'OPER "O999 SHOW MAT" RESP'
NLINES = (14*150)
STATUS = COMSDRCV('SYSG0999',DATA,'EXEC',2000)
IF STATUS > 4 THEN
  SAY 'RESULT = ' STATUS
ELSE
  DO I = 1 TO 14 BY 1
    J = ((I - 1) * 150) + 2
    DATA.I = SUBSTR(COMDATA.1,J,150)
  END
  SAY '===== RESPONSE ====='
  DO I = 1 TO 14 BY 1
    SAY STRIP(DATA.I,'T')
  END
  SAY '=== END RESPONSE ==='
RETURN STATUS

/* REXX */
RED_RC = REDIRECT("DATA."
"OPER "O999 SHOW MAT" RESP SYSID(SYSG0999)
DO I = 0 TO DATA.0
  SAY 'DATA.' I '=>' DATA.I
END
RED_RC = REDIRECT("OFF")
```

**Executing an exec on another AF/OPERATOR and receiving a response**

The following exec sends the AF/OPERATOR command:

```
EX TESTX2
```

to a remote AF/OPERATOR system identified by linkid SYSG1. The exec to be executed (in this case, TESTX2) must already reside in the REXX command library of the remote AF/OPERATOR. The number 10 specifies the length, in bytes, of response data to return to the exec on the originating system.

```rexx
/* REXX */
DATA = 'EX TESTX2'
STATUS = COMSDRCV('SYSG1',DATA,'EXEC',10)
IF STATUS > 4 THEN
  SAY 'RESULT = ' STATUS
ELSE
  SAY 'RESPONSE = ' COMDATA.1
RETURN STATUS
```
The response is returned to the originating exec and stored in REXX variable COMDATA.1.

The following exec, TESTX2 on SYSG1, uses the DATARPLY specification to return data to the REXX exec (shown above) running on the originating system.

```
/* REXX */
DATA = '123456789'
STATUS = COMSDRCV(DATARPLY)
IF STATUS <> 0 THEN
  SAY 'RESULT = ' STATUS
ELSE
  SAY 'DATARPLY EXECUTED OK'
RETURN STATUS
```

When using COMSDRCV with the DATARPLY specification, do not specify the linkid of the originating system or the length of the reply being sent. These tasks are handled by AF/OPERATOR.

**Issuing a WTOR on another AF/OPERATOR**

The following exec issues a WTOR on a remote AF/OPERATOR system identified by linkid SYSG1. A response of 119 bytes is requested. (MVS restricts the maximum response length for a WTOR to 119 bytes.)

```
/* REXX */
DATA = 'DO YOU WANT TO CONTINUE?'
STATUS = COMSDRCV('SYSG1',DATA,'WTOR',119)
IF STATUS <> 0 THEN
  SAY 'RESULT = ' STATUS
ELSE
  SAY 'REPLY = ' COMDATA.1
RETURN STATUS
```

The remote operator’s response is automatically returned to the originating AF/OPERATOR and stored in REXX variable COMDATA.1. The SAY command routs the response to the SYSLOG and to the AF/OPERATOR message log.

**Issuing an MVS command with response on another AF/OPERATOR**

You can use the COMSDRCV function to send any MVS or subsystem command and automatically receive a response.

For example, the following exec issues the MVS command:

```
D A,L
```

on a remote AF/OPERATOR system identified by linkid SYSO999. Up to 2000 bytes of the command’s response will be automatically returned to the exec on the originating AF/OPERATOR.
Using the COMSDRCV Function

The command’s response is returned as a single string and stored in REXX variable COMDATA.1. When a response is obtained from executing an MVS or subsystem command on another AF/OPERATOR system, each 150 bytes represent 1 line.

Issuing a WTORT

You can use the COMSDRCV function to route a WTO through a second AF/OPERATOR system to a third AF/OPERATOR system. This exec sends the WTO text:

**THIS IS A TEST**

to an intermediate AF/OPERATOR system identified by linkid SYSG1. SYSG1 reroutes the WTO to its final destination, the AF/OPERATOR system identified by linkid TESTB.

```rexx
/* REXX */
DATA = 'D A.L'
STATUS = COMSDRCV('SYSG0999',DATA,'CMRSP',2000)
IF STATUS > 4 THEN
  SAY 'RESULT = ' STATUS
ELSE
  OFFSET = 1
  DO WHILE OFFSET < LENGTH(COMDATA.1)
    MYDATA = SUBSTR(COMDATA.1,OFFSET,150)
    SAY MYDATA
    OFFSET = OFFSET + 150
  END
RETURN STATUS
```

When you send a WTORT, the first 8 characters of data must specify the linkid of the WTO’s final destination. Since linkid TESTB is shorter than 8 characters, it is followed by blank characters. The text of the message being sent begins at the ninth character of data.

Communication sessions from the originating system to the intermediate system, and from the intermediate system to the final destination must both be active for rerouting to occur.
Sending data to a distributed application

You can send data from AF/OPERATOR to a distributed application, such as AF/REMOTE, and receive a response. This example sends a telephone number to an application identified by linkid AFRMT. The number 20 is the maximum length, in bytes, of the expected response.

```rexx
/* REXX */
DATA = '3105551212'
STATUS = COMSDRCV('AFRMT',DATA,'USRDATA',20)
IF STATUS <> 0 THEN
    SAY 'RESULT = ' STATUS
ELSE
    IF COMDATA.1 = 'OK' THEN
        SAY 'REPLY = ' COMDATA.1
    ELSE
        SAY 'USER DATA NOT PROCESSED'
RETURN STATUS
```

Another REXX exec, running on the application identified by linkid AFRMT, processes the user-defined data and returns a response. The response is stored in REXX variable COMDATA.1.

Returning data to a distributed application

You can use the DATARPLY specification, in an exec running on AF/OPERATOR, to return data to an originating REXX script running on AF/REMOTE.

```rexx
/* REXX */
DATA = '123456789'
STATUS = COMSDRCV(,DATA,'DATARPLY')
IF STATUS <> 0 THEN
    SAY 'RESULT = ' STATUS
ELSE
    SAY 'DATARPLY EXECUTED OK'
RETURN STATUS
```

When using COMSDRCV with the DATARPLY specification, do not specify the linkid of the originating system or the length of the reply being sent. These tasks are handled by AF/OPERATOR.
**REXX Error Recovery**

**Description**

Execs you write must use proper error-handling techniques to ensure that the partner transaction program is not left hanging if a REXX error occurs during the communication session. The condition to be avoided is sometimes called a dangling conversation.

**Using SIGNAL ON ERROR**

If an exec is the sole user of a link (the link is not shared by other execs), a LINK STOP for that linkid should be included as part of the REXX SIGNAL ON ERROR procedure. The following partial REXX exec defines and starts an APPC communication session with AF/REMOTE. In the event of a REXX error or failure, the AF/OPERATOR communication session with this AF/REMOTE is stopped.

**FIGURE 14. REXX Error Recovery Example**

```rexx
/* REXX */
/*
  . Some statements
  */
ADDRESS AFHOST

link_name = 'RMT'
tp_name = 'SYSTEM1'
partner_lu_name = 'AFRTF'

"LINK DEFINE LINKID("link_name")",
  "MODENAME(LU62)",
  "TPNAME("tp_name") PLUNAME("partner_lu_name")",
  "NOID NOHRTBT"
If rc <> 0 Then
  Say 'LINK DEFINE failed for link' link_name 'RC=' rc
Else
```

114   AF/OPERATOR User’s Guide Version 340
Do
  "LINK START LINKID("link_name")"
If rc <> 0 Then
  Do
    Say 'LINK START failed for link' link_name 'RC= ' rc
  End
Else
  Do
    /* Link has been started successfully. Set SYNTAX */
    /* condition trap, so the link is terminated, */
    /* if any error occurs. */
    Signal On Syntax Name Link_terminate
    /* */
    /* . Remainder of program */
    /* */
  "LINK DISPLAY LINKID("link_name")"
  Call Execute_tp
  "WAIT SEC(5)"
  Signal OFF Syntax
  Call Link_terminate
End
End
Return
/* *------------------------------------------------------------------*/
Execute_tp:
BUFF = COPIES("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz",10)
STATUS = COMSDRCV(Link_name,BUFF,USRDATA)
SAY "STATUS: " STATUS
Return
/* *------------------------------------------------------------------*/
Link_Terminate:
If Condition() <> '' Then
  Do
    Parse source . . exec_name .
    Say 'Error occurred in program' exec_name
    Say 'Error condition is' Condition('C') 'on line' sigl
    Say 'Link' link_name 'will be terminated.'
  End
"LINK STOP LINK("link_name")"
Return
Configuration Requirements for Establishing APPC Connections

Overview

In order to establish APPC connections, you must define resources to APPC and VTAM. To do so, you will need to consult with your site’s VTAM or APPC system programmer or administrator to complete this chapter.

It outlines the steps necessary to define the VTAM logical units that will be used by AF/OPERATOR for APPC communication services.

What you need to know

Using APPC connections requires a knowledge of this communications protocol. If you need to refer to APPC documentation, review the following IBM MVS/ESA Version 4.3 references:

- MVS/ESA V4 Initialization and Tuning Reference
- MVS/ESA V4 Planning: APPC Management

System Requirements

AF/OPERATOR requires the following software to support APPC communications:

- An active APPC/MVS component on the MVS/ESA system on which AF/OPERATOR runs
- VTAM(R) Release 3.2 or later

APPC APPLID definitions

Complete the following configuration steps before using the APPC communication services interface support features of AF/OPERATOR.

1. Define LU 6.2 applids to VTAM.

   LUs are defined to VTAM on MVS by APPL statements in SYS1.VTAMLST.

   Define at least 2 LU 6.2 applids (one for inbound and one for outbound communication requests), or as many as your site requires. To use the same applid for more than one session, Candle recommends you define each LU as a parallel session.

   This is done with a PARSESS=YES instruction. (These lines are highlighted in the example below.) If you do not define the applids to support parallel sessions, you will need to define one applid for each AF/OPERATOR client session and one applid for each AF/OPERATOR server session.

   AF/OPERATOR requires that your APPL statements contain the APPC=YES instruction. (These lines are highlighted in the example below.) Other VTAM specifications may differ from those shown, depending on the conventions in use at your site. The following example shows APPL statements that define VTAM LU 6.2 applids named HIGHWAY1 and HIGHWAY6 using the APPC communication protocol.
2. Vary the node active.

\texttt{V NET,ACT,ID=xxxxxxxx}

where \texttt{xxxxxxxx} is the member name which resides in VTAMLST.

3. Make sure the APPC/MVS address space and APPC/MVS transaction scheduler address space are up. If they are not, issue the following commands to bring them up.

\texttt{S APPC,SUB=MSTR}

\texttt{S ASCH,SUB=MSTR}

4. Define LU 6.2 applids to APPC/MVS.

LUs used for APPC communication, in addition to being defined to VTAM, must also be defined to APPC/MVS LUADD statements in parameter library members named in the form APPCPMxx.

- AF/OPERATOR, to receive inbound communication requests, requires at least one NOSCHED type LU 6.2 applid defined to APPC/MVS.

The LUADD statement should resemble the following example.
Configuration Requirements for Establishing APPC Connections

AF/OPERATOR User’s Guide Version 340

Note: Both TPDATA and SIDEINFO are user-defined VSAM datasets. Refer to MVS/ESA V4 Planning: APPC Management for instructions on defining clusters for the side information and TP profile datasets.

- AF/OPERATOR, to initiate outbound communication requests, requires at least one SCHED type LU 6.2 applid defined to APPC/MVS as the base LU.

The LUADD statement should resemble the following example.

```
LUADD ACBNAME(HIGHWAY1)
  NOSCHED
  TPDATA(xxxx.xxxxxx)
  SIDEINFO DATASET(xxxx.xxxxxx)
```

**Note:** Both TPDATA and SIDEINFO are user-defined VSAM datasets. Refer to MVS/ESA V4 Planning: APPC Management for instructions on defining clusters for the side information and TP profile datasets.

- The APPCPMxx members containing the LUADD statements must reside in your SYS1.PARMLIB library. To activate the APPC LU, use the MVS **SET** command as shown below:

```
T APPC=xx
```

**Dependencies**

The Customizer MSMF INTEGRITY startup option has a direct effect on the performance of APPC communication sessions. If you choose Y for this option, the following lines are added to your AF/OPERATOR RKANPAR member:

```
MSMF_INTEGRITY_LEVEL(CONFIRM)
MSMF_INTEGRITY_SCOPE(ALL)
```

Applications that will interface to AF/OPERATOR must then be configured to support the CONFIRM session. This will increase system overhead each time data is transferred as the session waits for confirmation. The extent of the degradation of your system’s performance depends on the communication traffic at your site. You can increase the performance of your APPC communication sessions by choosing N for the MSMF INTEGRITY startup option. See “startup parameters” in the AF/OPERATOR Configuration and Customization Guide.
If heartbeat checking is in effect for the session, then applications that will interface to AF/OPERATOR must support confirm sessions (see information about the LINK DEFINE command in the AF/OPERATOR Command Reference Manual).
Overview
AF/OPERATOR XCF communications exploits the Cross-System Coupling Facility (XCF) service of MVS Sysplex.

System requirements
AF/OPERATOR XCF communications support is enabled only when XCF is available. This requires that the minimum hardware needed for a system to participate in a Sysplex be available.

XCF group and member name definitions
There is no need to define to the Cross-System Coupling Facility any specific resources used by AF/OPERATOR XCF communications. You should, however, consider establishing a naming convention for the XCF groups and members you use.

An XCF group can have one or more AF/OPERATOR servers active within it.

XCF groups and members determine couple data set size
The number of XCF groups and the largest number of members in an XCF group are used to determine the size of the Sysplex couple data set. A limit on these numbers is set when the systems programmer formats the Sysplex couple data set.

To display the maximum number of XCF groups and members per group as defined on your Sysplex, issue the command:

\[ \text{D XCF,COUPLE,TYPE=SYSPLEX} \]

What you need to know
For further information relating to the Sysplex couple data set, refer to the following IBM manual:

- MVS/ESA SP Vx Setting Up a Sysplex

Rules for naming XCF groups and members
An XCF group name can be up to 8 characters long and can consist of the characters A-Z, 0-9, and national characters ($, #, and @). To avoid conflicting with names used by IBM for its XCF groups, AF/OPERATOR will not allow group names beginning with the letters A through I or the character string SYS. AF/OPERATOR will also reject the special name UNDESIG, which is also reserved by IBM.

In addition to choosing the name for the XCF group, you must also identify an AF/OPERATOR XCF server by assigning it an XCF member name. The name of the server must be unique within the XCF group. AF/OPERATOR restricts member names to 8 characters and can consist of the characters A-Z, 0-9, and national characters ($, #, and @).
Candle suggests you avoid using member names starting with the pound sign (#). AF/OPERATOR reserves a special member name, #INTERN, for its “internal server”. You should also avoid assigning member names that conflict with member names generated by XCF. These generated names start with M and have a numeric suffix, for example, M1234.

**XCF Naming Examples**

If you want to connect two AF/OPERATOR systems, each having a server, you can choose one of the following naming methods:

1. Use two groups and, in both cases, assign the server a member name of SERVER. Thus, on SYSA, the server is the member called SERVER in group KOGGRP01, while on SYSB, the server is the member called SERVER in group KOGGRP02.
2. Use a single group with a group name of KOGGROUP. Give the server on SYSA the member name SERVSYS and the server on SYSB the member name SERVSYSB.

If you are trying to connect several systems in this manner, method 1 is preferable, since each client connecting to a server will make use of internal XCF group members for communications. With a large number of interconnected systems, the limit to the number of members in a group can easily be reached due to the number of clients within the group.
Configuration Requirements for Establishing TCP/IP Connections

System requirements
Using the TCP/IP communications protocol for an AF/OPERATOR session requires the following software:

- TCP/IP Version 3 Release 2 for MVS or later.

The TCP/IP for MVS address space must be active on the system where AF/OPERATOR is running.

Server definitions
There is no need to define to TCP/IP MVS any specific resources used by AF/OPERATOR. A port number, together with the local hostname, are used to uniquely identify a server within the network.

Since the local hostname is a constant value, the only additional requirements when establishing an AF/OPERATOR TCPIP server are to select a free port number within the allowable range and define the trusted hostnames list.

Trusted Hostnames List
In order to control who has access to your system through a TCP/IP server, it is necessary to define a list of authorized hostnames or IP addresses that can connect to a particular AF/OPERATOR. This list is referred to as the trusted hostnames list.

The trusted hostnames list is stored as a member called HOSTNAME in the same PDS used for the startup parameters member. This is the PDS referenced by the RKANPAR DD in the AF/OPERATOR startup procedure.

The trusted hostnames list has the following syntax rules:

- each entry in the list defines one potential client
- each entry is enclosed with [BEGIN] and [END] or <BEGIN> and <END> or +BEGIN+ and +END+ statements
- each entry has a single ADDRESS= statement that specifies a hostname or a dotted decimal IP address
- an individual hostname specification can be defined as STATIC or DYNAMIC. (By default, they are STATIC.)
  - Define a hostname as STATIC if the specified hostname is unlikely to resolve to a different IP address during the life of AF/OPERATOR. For efficiency, AF/OPERATOR will resolve an individual hostname only once rather than every time a logon attempt is received.
  - Define a hostname as DYNAMIC to resolve the individual hostname each time a logon attempt is received.
- comment lines are allowed and must begin with an asterisk
- each statement must begin in column 1
**Trusted hostnames list syntax examples**

The following provides examples of the trusted hostnames list syntax:

```
<BEGIN>
*comment - SYSTEM A
ADDRESS=10.0.0.1
<END>
<BEGIN>
*comment - SYSTEM B
ADDRESS=10.0.0.2
<END>
<BEGIN>
*comment - SYSTEM C
ADDRESS=SYSC
<END>
<BEGIN>
*comment - SYSTEM D
ADDRESS=SYSD
STATIC
<END>
<BEGIN>
*comment - SYSTEM E
ADDRESS=SYSD.XYZ.COM
STATIC
<END>
<BEGIN>
*comment - remote user
ADDRESS=USER1.XYZ.COM
DYNAMIC
<END>
```

**STATIC or DYNAMIC hostname definitions**

The HOSTNAME member is automatically read at AF/OPERATOR startup but can be refreshed dynamically using the AF RELOAD(HOSTNAME) command. Refreshing the trusted hostnames list clears any previously resolved STATIC hostname values. (This is useful if a normally static hostname has had its address modified for some reason.) The current list of trusted hostnames can be displayed using SHOW HOSTNAME.

The IP address of a client is checked against the trusted hostnames list by the server when a connect request is received. If the client is not in the trusted hostnames list, the attempted connection is immediately rejected.

When a connection is rejected for this reason, message !AOP3839 is displayed indicating the IP address from which the connection was attempted. For example,

```
!AOP3839 CONNECTION REFUSED FOR 123.123.123.123 -NOT IN TRUSTED HOSTNAMES LIST
```
Introduction

AF/OPERATOR provides two distinct levels of open TCP/IP communications features that enable AF/OPERATOR to communicate with many other TCP/IP-based applications on a variety of other platforms supporting TCP/IP.

Communications between two Candle AF systems can be achieved quite simply using standard commands and API without having to be aware of the specifics of the protocol underlying the method. In this unit, we describe the AF Communications Protocol, allowing user-written applications to utilize the methods contained in the protocol. In addition, it is possible to disable the AF Communications Protocol, thus providing access to completely open TCP/IP communications, where total control of data transmission is given to the user’s REXX application. This permits you to construct data streams that comply with protocols published for use with other software packages. For details, see “Advanced Communications--Open TCP/IP communications.” on page 146.

The AF packet protocol described in this section can be disabled through the use of the AFPACKET(OFF) keyword on the COMM START and the LINK DEFINE commands. Should you choose to disable the AF packet protocol, the data is passed is simply a continuous stream of bytes without boundaries, and it will therefore be the responsibility of the user application to impose a structure on it.

The simplest method of causing AF/OPERATOR to run an EXEC and pass back a response is through the use of the AF packet protocol described in this section. Thus, Candle recommends using the default, AFPACKET(ON), for this purpose.

Candle AF packet format

Data transmitted between AF/OPERATOR or AF/Remote and any user application must comply with the following internal message format:

<table>
<thead>
<tr>
<th>Linkid or Message Token</th>
<th>8 byte string or integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Type</td>
<td>8 byte string</td>
</tr>
<tr>
<td>Reply Data Length</td>
<td>2 byte integer/binary</td>
</tr>
<tr>
<td>Send Data length</td>
<td>2 byte integer/binary</td>
</tr>
<tr>
<td>Send Data</td>
<td>32K maximum length</td>
</tr>
</tbody>
</table>

If you want to create your own applications such as C/C++, Java, or Visual Basic programs or REXX scripts that use native socket code, you must understand the details of the message packet format and the order of and restrictions on sending different types of records.

Note: The AF/OPERATOR COMSDRCV REXX function and AF/Remote REXX peer API functions already implement this internal message packet format. You do not need to code to this specification if you are simply planning to use the existing AF functions.

All user-written applications that want to communicate with AF/OPERATOR or AF/Remote must adhere to the message packet format described above. By following this format, a REXX script using native REXX socket code or a C/C++, Java, or Visual Basic program...
making socket calls can establish TCP/IP-based communications with AF/OPERATOR or AF/Remote.

**The linkid or message token**

The linkid/message token field in the message packet has dual uses. It is normally the place where the linkid is stored. The linkid is a 1- to 8-character alphanumeric string assigned by the originator of the conversation as a handle. It should be left-justified so that it begins in position 1 and is padded with blanks to fill all 8 bytes. The linkid must be present in the first message sent, which is always an ID message.

Once a conversation has been established, this 8-byte field can be reused as a message token. Tokens are used to associate response data that is received with the originating request. Tokens are helpful if multiple requests for data have been sent and the order of responses received is unpredictable. In this case, the originator of the request should supply a unique value in the first 8 bytes of the message packet. The peer system is then obligated to return the same token in the first 8 bytes of the response, along with the response data itself.

If message tokens are not being used, then each message should begin with the conversation linkid.

**Data types**

The data type field in the message packet must be a known data type. It can be one of the following:

- ID
- EXEC
- DATARPLY
- DATARERR
- USRDATA

The data type field should begin in position 9 of the message packet. It should be left-justified in the field and the field padded with blanks.

The data type field should reflect the contents of the data contained in the fifth field, the send data field of the message packet.

**The ID data type**

An ID record must be the first message packet sent after a connection is made. The send data should contain information about the connecting application, such as its name, version information, and so forth.

When a user application initiates a conversation with an AF server, it must specify `REPLY=YES` in the ID information. If the ID record is accepted, the AF system will return its own ID record as an acknowledgement. This acknowledgement ID record will specify `REPLY=NO`. 
When a user server application receives an ID record that shows \texttt{REPLY=YES} in its data, the user server application should acknowledge the ID by replying with its own ID specifying \texttt{REPLY=NO}.

The exchange of ID records allows certain options for the conversation to be negotiated.

**ID record keywords:** The ID data type’s send data field contains various keywords as shown below, separated by semicolons.

- **NAME=** Product name
- **VERSION=** AF Protocol version. Value must be 1.
- **MAINTENANCE=** Maintenance level
- **DATA=** Values may be EBCDIC or ASCII
- **DATARPLY=** Value \{NORMAL/KEYWORD\}. The default for this parameter is NORMAL. Only the reply is returned; no AOCASE value or any other keyword values are returned. A value of KEYWORD causes the reply to be formatted into keyword/value pairs delimited by semicolons. The ‘\texttt{REPLY=value}’ pair is always the last keyword/value pair without a trailing semicolon.
- **REPLY=** Values may be YES or NO, indicating whether or not an ID is expected to be returned.
- **CLIENTID=** Value is xxxxxxxx, and is the identifier used in COMM DISPLAY or other messages, for example, !AOP3836.

**ASCII or EBCDIC ID record negotiation:** AF/OPERATOR handles either ASCII or EBCDIC data. If you want to receive an ASCII reply when AF/OPERATOR is acting as the server, you must indicate that on an ID record. By sending an ID packet with \texttt{DATA=ASCII} specified as an option, you will have ASCII data returned to you.

Specifying \texttt{DATA=ASCII;REPLY=YES} in your ID record causes AF/OPERATOR to return its ID record, which should include the negotiated \texttt{DATA=ASCII} value as well as any text in ASCII format. (If the returned text is not in ASCII format, the negotiation has failed.)

When AF/OPERATOR connects to a remote server as a client, it automatically sends an EBCDIC ID record unless you have specified the ASCII keyword on the \texttt{LINK DEFINE} command. Even if the first ID record from AF/OPERATOR is sent in EBCDIC, you can still negotiate that subsequent packets are to be translated to ASCII by replying to AF/OPERATOR’s ID packet with an ID packet containing the \texttt{DATA=ASCII} keyword.

**Note:** The \texttt{USRDATA} data type data is never modified or translated, and is not affected by the \texttt{DATA=} negotiation.

**Examples of ID records:** The following is an example ASCII ID packet from a client:

```
XXX ID ....NAME=FRED;VERSION=1;DATA=ASCII;DATARPLY=KEYWORD
55522222442222200204444345444454444444444434444444454445454545445
888000009400000000E0E1D5D6254B65239FED1B4141D13399B414120C9DB597F24
```

The above example also shows little-endian lengths. For example, on the mainframe processor, 36 is represented in hexadecimal as \texttt{X’002E’}; in little-endian, it’s represented as \texttt{X’2E00’}. If you specify a \texttt{DATA=ASCII} negotiation, that will ensure that the length is kept straight-forward.
The EXEC data type — AF/OPERATOR command execution

To run a REXX exec called XYZ on AF/OPERATOR, the client system sends the following:

```
anytokenEXEC ....EX XYZ
```

In EBCDIC, this is

```
anytokenEXEC ....EX XYZ
89AA9989CECC44440000CE4EEE
15836255575300000006570789
```

AF/OPERATOR accepts ASCII, and with ASCII, the length must be little-endian.

If a reply is to be returned to the client, then the reply data length must be non-zero and indicates the maximum size reply to be returned to the client by AF/OPERATOR. In this case, the first 8 characters are a token that is returned unaltered to the client. Thus, if multiple requests are outstanding, the reply can be given to the correct originator on the client system. The client sends the following, with a reply length equal to 16 bytes:

```
........EXEC ....EX XYZ
00000135CECC44440100CE4EEE
00000246575300000006570789
```

When the reply to this EXEC is received, the token field must contain the original token from the EXEC; for example, X'0000000000123456'.

To run a REXX EXEC and pass it a parameter, send:

```
token002EXEC ....EX ABC 'parm data'
A9989FFCECC44440401CE4CCC479899488A87
362550025753000000025701230D719404131D
```

The command executed through an EXEC packet is not limited to the EX command. It can be any AF/OPERATOR command, for example:

```
token001EXEC ....SHOW STATS
A9989FFCECC44440000ECDE4EECEE
362550015753000000025701230D719404131D
```

The response data types — DATARPLY and DATARERR

An AF/OPERATOR EXEC initiated by receipt of an EXEC message packet request can use the Candle-supplied REXX function COMSDRCV with a datatype of DATARPLY to return a reply. Here are some examples of coding the COMSDRCV REXX function:

```
RC = COMSDRCV(,'REPLY','DATARPLY')
```

or

```
text = 'REPLY'
RC = COMSDRCV(text,'DATARPLY')
```

Assuming the example above was initiated on receipt of the following EXEC packet:

```
........EXEC ....EX XYZ
00000135CECC44440100CE4EEE
00000246575300000006570789
```

then these examples cause the following packet to be sent back to the client:
The original token, X’0000000000123456’ is returned as well as is the reply data. The token could have been a character value. If the DATARPLY=KEYWORD was specified on the ID record, the actual reply value will consist of a string of keyword/value pairs separated by semicolons. Currently, the only pairs are ‘AOCASE=value’;REPLY=value’, but program parse routines should be written to allow for the addition of new keywords. As these are keyword pairs, there is no specified order in which they appear, except that the REPLY/VALUE pair will always be present at the end of the reply string. The keywords, other than ‘REPLY=’, may or may not be present depending on the type of command that caused the reply to be created.

If an error occurs, you will receive a DATARERR packet indicating a data reply error. For example, if the named EXEC doesn’t exist or the EXEC ends before sending a DATARPLY, you will receive:

The reply includes the original token plus a numeric return code and character reason code.

**Note:** If you have requested a response and the AF/OPERATOR EXEC ends before you issue a DATARPLY, an automatic DATARERR message packet response is returned. To avoid the client waiting indefinitely, all user-written server applications should ensure that a reply (either DATARPLY or DATARERR) is always given.

**EXEC requests from AF/OPERATOR to a user application:** If you are connecting from AF/OPERATOR to another application, you can use the Candle-supplied REXX function COMSDRCV with a datatype of EXEC to send this data type request.

If a response is requested on the COMSDRCV request, the remote application should return either a DATARPLY or DATARERR packet. The token supplied in the originating EXEC message packet must be returned as the token value in the response message packet.

**Use of allowed message types**

An AF/OPERATOR server accepts all packet types except USRDATA packets. USRDATA packets can only be used on a connection from a client to a non-AF/OPERATOR server. An AF/OPERATOR server is able to return ID, DATARPLY, or DATARERR packets.
Advanced Communications--Creating a Simple Client Application

Overview

Having discussed the AF packet format syntax requirements, this section will show how to put the various request types together in a sample AF TCP/IP client application.

This sample client application will connect with AF/OPERATOR, perform handshake processing then run an EXEC which should pass some data back to the client in response.

The example is written in REXX for OS/390 or z/OS using the REXX Socket Application Programming Interface (API), but the basic principles for socket connection and data transmission apply to most programming languages.

Process flow

The client application can be broken down into four basic sections:

1. Initialize the TCP/IP environment
2. Connect to server
3. Exchange data with server
4. Terminate the environment

Each of these sections will be discussed individually below.
Initialize the TCP/IP environment

Before we can connect to the server, we must first initialize the TCP/IP environment and allocate a socket, which acts as an end-point for communications.

The sample code below shows how we do this using the REXX SOCKET function calls for ‘INITIALIZE’ and ‘SOCKET’.

The REXX variable USER is set to contain a unique identifier to TCP/IP for this execution of the application. This unique identifier is used on the SOCKET ‘INITIALIZE’ call.

In addition, the server is identified by hostname and port number in the variables TCPHost and TCPPort. These variables will be used later in the example when we establish the connection.

/* REXX */
USER = SYSVAR("SYSUID")              /* IDENTIFY OURSELVES TO TCP/IP */
TCPHost = 'SYSA'                      /* HOSTNAME OF SERVER */
TCPPort = '12345'                     /* PORTNUMBER OF SERVER */

/* INITIALIZE TCP/IP ENVIRONMENT */
PARSE VALUE SOCKET('INITIALIZE',USER) WITH RC

/* ALLOCATE SOCKET FOR COMMUNICATIONS */
PARSE VALUE SOCKET('SOCKET') WITH RC SocketID

Assuming successful completion of both calls, the REXX variable SocketID now contains a socket descriptor, which is basically a handle or identifier for our application’s end of the TCP/IP conversation that is about to be established. The SocketID socket descriptor should be used on all further REXX SOCKET function calls for this connection.

Connect to server

The connection to the server is established using the REXX SOCKET function call ‘CONNECT’. This call requires that we provide a dotted decimal IP address. Therefore, prior to connecting, we use the 'GETHOSTBYNAME' request to resolve the supplied hostname into a dotted decimal IP address.

The third argument on CONNECT is referred to as the network address and comprises the literal '2', plus the port number and dotted decimal IP address of the server to which we are connecting.

/* RESOLVE HOSTNAME INTO DOTTED DECIMAL IP ADDRESS */
PARSE VALUE SOCKET('GETHOSTBYNAME',TCPHost) WITH RC TCPAddr

/* CONNECT TO SERVER */
PARSE VALUE SOCKET('CONNECT',SocketID,'2' TCPPort TCPAddr) WITH RC

The connection is made specifying the previously assigned socket descriptor in SocketID. Note that ‘GETHOSTBYNAME’ is one of the few exceptions to the rule that a SocketID must be specified; it does not require a socket descriptor be provided.
Exchange data with server

The exchange of data between the two end-points of the conversation we have established is achieved using the REXX SOCKET function calls 'SEND' and 'RECV'. We must now be concerned with the AF packet syntax as previously described.

Sending data

In our sample client application, we will send two different AF request packets.

- The ID record is used in the handshake
- The EXEC record is used to perform work on AF/OPERATOR.

Since we're going to be using it twice we'll put the REXX SOCKET 'SEND' call in a subroutine.

```rexx
/* ************************************ */
/* SUBROUTINE TO SEND DATA */
/* ************************************* */
DO_SEND: PROCEDURE EXPOSE SocketID
ARG SEND_DATA
PARSE VALUE SOCKET('SEND',SocketID,SEND_DATA) WITH RC LENGTH
IF WORD(RC,1) <> 0 THEN SAY 'SEND RC=' RC
RETURN RC

EXPOSE will give the subroutine access to the socket descriptor of our conversation and the data we want to send is supplied as the only argument to the subroutine.

The ID record is the first data the client transmits and is the more complex of the two types we're sending. We construct our ID record as shown in the following code segment.

```rexx
ID_LINK = LEFT('TSO'||USER,8)             /* 8 CHARACTER LINKID */
ID_TYPE = LEFT('ID',8)                            /* 8 CHARACTER REQUEST TYPE */
ID_REPLY_LEN = D2C(100,2)                 /* 2 BYTE REPLY LENGTH */
ID_DATA = 'NAME=REXX_CON;REPLY=YES;DATA=EBCDIC;CLIENTID='||USER
ID_DATA_LEN = D2C(LENGTH(ID_DATA),2)   /* 2 BYTE DATA LENGTH */
RC = DO_SEND(ID_LINK||ID_TYPE||ID_REPLY_LEN||ID_DATA_LEN||ID_DATA)
```

Assuming a TSO userid of USER1, the following ID record is constructed and sent to AF/OPERATOR.

```
TSUSER1ID      ....NAME=REXX_CON;REPLY=YES;DATA=EBCDIC;CLIENTID=USER1
EDEBECDCCCA44DD5D603DCDCC7DCEB66CDD5DCDDE7ECB5CCEC7CCCCC5CDCCDCECC7EBCDF
326425919400000004025145E9577D365EB95738E852B4131E523493E33955394E42591
```

Assuming we have our EXEC request in a REXX variable called 'EX_DATA', we also use our DO_SEND subroutine to send the EXEC record. This is constructed in a very similar manner as show below.

```rexx
EX_LINK = LEFT('TKN'||USER,8)             /* 8 CHARACTER TOKEN */
EX_TYPE = LEFT('EXEC',8)                            /* 8 CHARACTER REQUEST TYPE */
EX_REPLY_LEN = D2C(3000,2)                 /* 2 BYTE REPLY LENGTH */
EX_DATA = SEND_DATA
EX_DATA_LEN = D2C(LENGTH(EX_DATA),2)   /* 2 BYTE DATA LENGTH */
RC = DO_SEND(EX_LINK||EX_TYPE||EX_REPLY_LEN||EX_DATA_LEN||EX_DATA)
```
In the case of the EXEC record, the first 8 character of the packet comprise the token. If we were to have more than one concurrent request outstanding, we would need to generate unique token values. This would allow us to associate the response with the original request. However, in our simple client application, multiple token values are not needed, as any response we receive from AF/OPERATOR must relate to our single EXEC request.

The reply length field indicates the maximum length of the response our application will accept. If you need more data returned to you, increase this value.

Assuming the EX_DATA variable contains 'EX TESTEXEC', the following stream is sent to AF/OPERATOR.

```
TKNUSER1EXEC ....EX TESTEXEC
EDDEECDFCECC44440B00CE4ECEECCECC
3254259157530000B80B57035235753
```

**Receiving data**

Our previous examples have dealt with constructing and sending both packet types used by our sample application. However we skipped over receiving the ID record we get in response to our ID request.

Since we specified REPL Y=YES and we also coded a non-zero reply length in the ID packet we sent, AF/OPERATOR will send a reply in response.

**Significant characteristics of stream socket receive:** The REXX SOCKET function RECV has the following basic syntax.

```
RC = SOCKET('RECV', socketid, maxlength)
```

Receive processing requires attention to the following details for the reasons shown:

- It’s important to be aware that the third argument, maxlength, indicates the maximum amount of data you want to receive. It is possible to receive a smaller amount of data than you specify in maxlength.
  
  If you expect a certain amount of data on the stream, you may need to perform multiple receives until you get the required amount of data.

- AF uses stream sockets. The data transmitted over a stream socket is just a continuous stream of bytes; there are no delimiters build in by the underlying TCP/IP services. It's at this point that the need for the AF packet protocol becomes apparent.
  
  Each packet that AF sends has a 20 byte header and that header contains the length of the variable portion that immediately follows on the stream. This gives us the ability to extract individual requests from the stream.

A simple way to approach receiving data that complies with the AF packet protocol is to make two passes at receiving it. First, retrieve the fixed 20 bytes header; then retrieve the variable section that follows.

**The receive subroutine:** Again since we're going to be using it several times, we'll put the REXX SOCKET 'RECV' call in a subroutine. If we pass the length of data expected as an argument to the subroutine, we can manage, within the subroutine, the process of doing repeated receives until the expected data is complete.
DO_RECV: PROCEDURE EXPOSE SocketID
ARG RECV_MAX
RECV_DATA = ''
RC = 0
DO WHILE RC = 0 & RECV_MAX > 0
    PARSE VALUE SOCKET('RECV',SocketID,RECV_MAX) WITH RC LEN BUFF
    IF RC = 0 THEN DO
        IF LEN = 0 THEN DO
            SAY 'ZERO BYTES RECEIVED = CONNECTION TERMINATED'
            LEAVE
        END
        RECV_MAX = RECV_MAX - LEN
        RECV_DATA = RECV_DATA||BUFF
    END
    ELSE SAY 'RECV RC=' RC BUFF
END
RETURN RC RECV_DATA

The receive subroutine is passed a single argument containing the length of data to be received from the stream.

The received data will be accumulated in the variable called RECV_DATA before being returned to the caller. The amount of data remaining to be received is maintained in the variable called RECV_MAX. In the event that we only receive part of the required amount of data, the RECV_MAX value is reduced according to how much has actually been received and can therefore be used on the next SOCKET 'RECV' call as the maxlen argument.

If a return code of zero is received from 'RECV' and no data is returned, this indicates that the remote partner has terminated the connection.

Therefore we can extract an individual AF packet from the stream by issuing two calls to the DO_RECV subroutine.

GET 20 BYTE PACKET HEADER
PARSE VALUE DO_RECV(20) WITH RC ID_REPLY
IF RC = 0 THEN DO /* IF HEADER RECEIVE OK */
    REPLY_DATA_LEN = C2D(SUBSTR(ID_REPLY,19,2))
    /* GET VARIABLE LENGTH ID REPLY DATA */
    PARSE VALUE DO_RECV(REPLY_DATA_LEN) WITH RC ID_REPLY
END

The response to the EXEC request will be received in an almost identical manner.
Terminating the environment

The simplest way to terminate the TCP/IP environment is by using the REXX SOCKET function 'TERMINATE' call. This call also closes the connection and releases the socket.

/* TERMINATE TCP/IP ENVIRONMENT
PARSE VALUE SOCKET('TERMINATE') WITH RC
OS/390 REXX sample TCP/IP client application

This example, as well as the example provided in “OS/390 REXX sample TCP/IP client application” on page 135 amplify the information provided in the previous section, “Advanced Communications--AF TCP/IP Communications Protocol” on page 124.

Running the sample client application

Before you execute the sample client application, you must have started a server port on AF/OPERATOR and added the hostname/IP address of the system from where the connection is being made to the trusted hostnames member.

Sample application code

Now that you are familiar with all of the individual components to the sample application, a few return code checks can be added to complete the application as follows:

```rexx
/* REXX */
/* ******************************************************* */
/* SAMPLE TCP/IP CLIENT CONNECTION TO AF/OPERATOR */
/* ********************************************************* */
USER = SYSVAR("SYSUID") /* IDENTIFY OURSELVES TO TCP/IP */
TCPHost = 'SYSA' /* HOSTNAME OF SERVER */
TCPPort = '12345' /* PORTNUMBER OF SERVER */

/* INITIALIZE TCP/IP ENVIRONMENT */
PARSE VALUE SOCKET('INITIALIZE',USER) WITH RC
IF WORD(RC,1) = 0 THEN DO /* IF INITIALIZED SUCCESSFULLY */

/* RESOLVE HOSTNAME INTO DOTTED DECIMAL IP ADDRESS */
PARSE VALUE SOCKET('GETHOSTBYNAME',TCPHost) WITH RC TCPAddr
IF WORD(RC,1) = 0 THEN DO /* IF GETHOSTBYNAME SUCCESSFUL */
SAY 'HOSTNAME' TCPHost 'RESOLVED TO' TCPAddr

/* ALLOCATE SOCKET FOR COMMUNICATIONS */
PARSE VALUE SOCKET('SOCKET') WITH RC SocketID
IF WORD(RC,1) = 0 THEN DO /* IF SOCKET OBTAINED */
SAY 'SOCKET NUMBER= ' SocketID

/* CONNECT TO SERVER */
PARSE VALUE SOCKET('CONNECT',SocketID,'2' TCPPort TCPAddr) WITH RC
IF WORD(RC,1) = 0 THEN DO /* IF CONNECTED OK */
SAY 'CONNECTED TO' TCPHost 'PORT' TCPPort

/* PERFORM HANDSHAKE */
RC = SEND_ID_REC()
IF WORD(RC,1) = 0 THEN DO /* IF HANDSHAKE SUCCESS */

/* PROMPT USER FOR 'EXEC' COMMAND */
SAY 'ENTER COMMAND FOR AF =>';
PARSE UPPER PULL CMD;

/* PERFORM EXEC */
RC = SEND_EXEC_REC(CMD)
END
```
END
ELSE SAY 'CONNECT RC=' RC
END
ELSE SAY 'SOCKET RC=' RC
END
ELSE SAY 'GETHOSTBYNAME RC=' RC

/* TERMINATE TCP/IP ENVIRONMENT */
PARSE VALUE SOCKET('TERMINATE') WITH RC
IF WORD(RC,1) <> 0 THEN SAY 'TERMINATE RC=' RC
END
ELSE SAY 'INITIALIZE RC=' RC
RETURN 0

/* ********************************************************** */
/* SUBROUTINE TO SEND ID RECORD TO SERVER */
/* ********************************************************** */
SEND_ID_REC: PROCEDURE EXPOSE USER SocketID
ID_LINK = LEFT('TSO'||USER,8)          /* 8 CHARACTER LINKID               */
ID_TYPE = LEFT('ID',8)                        /* 8 CHARACTER REQUEST TYPE */
ID_REPLY_LEN = D2C(100,2)               /* 2 BYTE REPLY LENGTH            */
ID_DATA = 'NAME=REXX_CON;REPLY=YES;DATA=EBCDIC;CLIENTID='||USER
ID_DATA_LEN  = D2C(LENGTH(ID_DATA),2)   /* 2 BYTE DATA  LENGTH      */
RC = DO_SEND(ID_LINK||ID_TYPE||ID_REPLY_LEN||ID_DATA_LEN||ID_DATA)
IF WORD(RC,1) = 0 THEN DO               /* IF SEND SUCCESSFUL       */
/* GET 20 BYTE PACKET HEADER                                                      */
PARSE VALUE DO_RECV(20) WITH RC ID_REPLY
IF RC = 0 THEN DO                     /* IF HEADER RECEIVE OK          */
IF SUBSTR(ID_REPLY,9,8) == LEFT('ID',8) THEN DO
  REPLY_DATA_LEN = C2D(SUBSTR(ID_REPLY,19,2))
  SAY 'ID HEADER=> SUBSTR(ID_REPLY,1,16) 'LEN' REPLY_DATA_LEN
/* GET VARIABLE LENGTH ID REPL Y DATA                                 */
PARSE VALUE DO_RECV(REPLY_DATA_LEN) WITH RC ID_REPLY
IF RC = 0 THEN SAY 'ID REPL Y =>' ID_REPLY
ELSE SAY 'RECV ID RC=' RC
END
ELSE SAY 'RECV ID RC=' RC
ENDIF
ELSE SAY 'SEND RC=' RC
RETURN RC
Advanced Communications--Creating a Simple Client Application

/* ********************************************************************* */
/* SUBROUTINE TO SEND EXEC RECORD TO SERVER */
/* ********************************************************************* */
SEND_EXEC_REC: PROCEDURE EXPOSE USER SocketID
ARG SEND_DATA

EX_LINK = LEFT('TKN'|USER,8) /* 8 CHARACTER TOKEN */
EX_TYPE = LEFT('EXEC',8) /* 8 CHARACTER REQUEST TYPE */
EX_REPLY_LEN = D2C(3000,2) /* 2 BYTE REPLY LENGTH */
EX_DATA = SEND_DATA
EX_DATA_LEN = D2C(LENGTH(EX_DATA),2) /* 2 BYTE DATA LENGTH */

RC = DO_SEND(EX_LINK||EX_TYPE||EX_REPLY_LEN||EX_DATA_LEN||EX_DATA)
IF WORD(RC,1) = 0 THEN DO /* IF SEND SUCCESSFUL */
/* GET 20 BYTE PACKET HEADER */
PARSE VALUE DO_RECV(20) WITH RC EXEC_REPLY
IF RC = 0 THEN DO /* IF HEADER RECEIVE OK */
IF SUBSTR(EXEC_REPLY,9,8) == 'DATA RPLY' THEN DO
REPLY_DATA_LEN = C2D(SUBSTR(EXEC_REPLY,19,2))
SAY 'EXEC HEADER=>' SUBSTR(EXEC_REPLY,1,16) LEN REPLY_DATA_LEN
/* GET VARIABLE LENGTH EXEC REPL Y DATA */
PARSE VALUE DO_RECV(REPLY_DATA_LEN) WITH RC EXEC_REPLY
IF RC = 0 THEN SAY 'EXEC REPLY => EXEC_REPLY'
ELSE SAY 'RECV EXEC RC=' RC
END
ELSE SAY 'RECV EXEC RC=' RC
END
ELSE SAY 'SEND RC=' RC

RETURN RC

/* ********************************************************************* */
/* SUBROUTINE TO SEND DATA */
/* ********************************************************************* */
DO_SEND: PROCEDURE EXPOSE SocketID
ARG SEND_DATA

PARSE VALUE SOCKET('SEND',SocketID,SEND_DATA) WITH RC LENGTH
IF WORD(RC,1) <> 0 THEN SAY 'SEND RC=' RC
RETURN RC

/* ********************************************************************* */
/* SUBROUTINE TO RECEIVE DATA */
/* ********************************************************************* */
DO_RECV: PROCEDURE EXPOSE SocketID
ARG RECV_MAX
RECV_DATA = ""

Understanding the Communications Environment 137
RC = 0
DO WHILE RC = 0 & RECV_MAX > 0
PARSE VALUE SOCKET(‘RECV’,SocketID,RECV_MAX) WITH RC LEN BUFF
IF RC = 0 THEN DO
  IF LEN = 0 THEN DO
    SAY ‘ZERO BYTES RECEIVED = CONNECTION TERMINATED’
    LEAVE
  END
  RECV_MAX = RECV_MAX - LEN
  RECV_DATA = RECV_DATA||BUFF
END
ELSE SAY ‘RECV RC=’ RC BUFF
END
RETURN RC RECV_DATA

Expected results
The sample code has a number of SAY statements that indicate the current stage within the process. You should see something like:

Example 1:

HOSTNAME SYSA RESOLVED TO 192.168.0.1
SOCKET NUMBER= 1
CONNECTED TO SYSA (192.168.0.1) PORT 12345
ID HEADER=> TSOUSER1ID LEN 89
ID REPLY =>
NAME=OMCGATEWAY/MVS;VERSION=1;MAINTENANCE=INITIAL;REPLY=NO;
DATA=EBCDIC
ENTER COMMAND FOR AF => ex testexec
EXEC HEADER=> TKNUSER1DATARPLY LEN 32
EXEC REPLY => EXEC WAS MATCH 93 (VERSION 0340)

In the above example, the EXEC called TESTEXEC has used the Candle-supplied COMSDRCV REXX function with the DATARPLY request type.

Here is the example TESTEXEC REXX procedure that AF/OPERATOR used to produce the above response:

/* REXX */
X = ‘GLBVGET’(“AOMATNUM”)
X = ‘SYSVGET’(“AOVSN”, “AORLS”)
REPLY = ‘EXEC WAS MATCH’ AOMATNUM ‘(VERSION’ AOVSN||AORLS’)’
RC = COMSDRCV(REPLY,’DATARPLY’)

Note: Any AF/OPERATOR command can be issued by the delivery of an AF EXEC packet. You are not limited to the EX or EXEC command. You should also be aware that OPER RESP and WTOR commands are able to route the response back to the originator.
Example 2:

HOSTNAME SYSA RESOLVED TO 192.168.0.1
SOCKET NUMBER= 1
CONNECTED TO SYSA (192.168.0.1) PORT 12345
ID HEADER=> TSouser1ID  LEN 89
ID REPLY =>
NAME=OMCGATEWAY/MVS;VERSION=1;MAINTENANCE=INITIAL;REPLY=NO;
DATA=EBCDIC
ENTER COMMAND FOR AF => oper 'd xcf' resp
EXEC HEADER=> TKUSER1DATARPLY LEN 450
EXEC REPLY =>  D XCF
   IXC334I  07.24.50  DISPLAY XCF 057
   SYSPLEX LPAR400J:   SYSA     SYSB     SYSC

The sample REXX application shows how to connect to AF/OPERATOR, perform a command, and receive a response. Rather than just issuing a SAY of the response, you may wish to perform additional output formatting within the REXX application to make response data more readable.

Client information in the AF environment

If you issue a COMM DISPLAY while your EXEC is running, you should see something like the example below showing details of our connection, where the LINKID value (the first 8 characters of the ID record) and the CLIENTID= value from the ID record keywords are substituted.

!AOP3743 ---- CLIENT HOSTNAME - 192.168.0.1
!AOP3743 ---- SERVER - SYSA999(TSOUSER1)
!AOP3743 ---- CLIENT - USER1
!AOP3743 ---- INTERNAL LINK ID - #0000029
!AOP3743 ---- CONNECTION ID - #A000042
!AOP3743 ---- TOTAL RECEIVE COUNT - 2
!AOP3743 ---- ID COUNT - 1
!AOP3743 ---- EXEC COUNT - 1

Additionally the CLIENTID= value is available in the COM match global variable called AOCLNTID.

!AOP0201 GLOBAL AOCLNTID='USER1'

Common errors

If you haven’t issued the COMM START command on AF/OPERATOR, the port to which you are being connected to on the host isn’t open. This condition will cause you to receive:

HOSTNAME SYSA RESOLVED TO 192.168.0.1
SOCKET NUMBER= 1
CONNECT RC= 61 ECONNREFUSED Connection refused

If you’ve not put the IP address of the client system in the AF/OPERATOR trusted hostnames list, the connection will be terminated before the handshake is completed. Depending on the network speed, this may be before or after the ID packet is sent.

HOSTNAME SYSA RESOLVED TO 192.168.0.1
SOCKET NUMBER= 1
CONNECTED TO SYSA (192.168.0.1) PORT 12345
RECV RC= 54 Connection reset by peer

If a REXX procedure is to be run but the member is unable to be found by AF/OPERATOR, the response is a DATARERR packet with a return code 2824:

ENTER COMMAND FOR AF => ex unknown
EXEC HEADER=> TKNUSER1DATARERR LEN 23
EXEC REPLY => RETURN=2824;REASON=NONE

If the command is invalid to AF/OPERATOR, the DATARERR packet contains return code 2020:

ENTER COMMAND FOR AF => doh
EXEC HEADER=> TKNUSER1DATARERR LEN 23
EXEC REPLY => RETURN=2020;REASON=NONE
ASCII connections

When connecting to AF/OPERATOR from an ASCII-based platform, it is not necessary to worry about ASCII to EBCDIC conversion of text. Simply include DATA=ASCII in the ID record handshake and AF/OPERATOR will translate the textual portion of the packet. However, from an ASCII platform, AF/OPERATOR expects the two length fields in the AF packet header to be little-endian (or LSB 'least significant byte' first).

For example, the previously sent EXEC packet (which has big-endian lengths)

```
TKNUSER1EXEC     .. .. EX TESTEXEC
EDDEECDFCECC4444 0B 00 CE4ECECECC
3254259157530000 B8 0B 57035235753
```

would be as follows in ASCII format with little-endian lengths

```
TKNUSER1EXEC     .. .. EX TESTEXEC
5445545345442222 B0 00 45254555454
4BE5352158530000 8B B0 58045345853
```

Notice how the reply-length field value of 3,000 is represented as 0BB8 in the EBCDIC or big-endian stream, but is B80B in the ASCII or little-endian stream.

Additionally the data_length field of 11 is represented as 000B and 0B00 respectively.

The simplest method to convert the lengths in REXX is to use the REVERSE function; for example, when constructing a length to be sent, the statement

```
EX_REPLY_LEN = D2C(3000,2)  /* 2 BYTE REPLY LENGTH */
```

for little-endian becomes

```
EX_REPLY_LEN = D2C(REVERSE(3000,2))  /* 2 BYTE REPLY LENGTH */
```

When extracting a little-endian length from the stream, the statement

```
REPLY_DATA_LEN = C2D(SUBSTR(EXEC_REPLY,19,2))
```

for little-endian becomes

```
REPLY_DATA_LEN = C2D(REVERSE(SUBSTR(EXEC_REPLY,19,2)))
```
Object REXX for Windows Sample TCP/IP Client Application

This example, as well as the example provided in “Advanced Communications--Creating a Simple Client Application” on page 129, amplify the information provided in the section “Advanced Communications--AF TCP/IP Communications Protocol” on page 124.

With a few small adjustments, our OS/390 REXX client application can be converted to run under Object REXX for Windows.

The main change is the use of "REXX TCP/IP Socket Library Functions (RxSock)"; the code is modified to allow for the different arguments and return code values. Additionally "DATA=ASCII" has been added to the ID record used in the handshake process and the update for little-endian length processing has been made.

The same basic socket functions are used:

- Init
- GetHostByName
- Socket
- Connect
- Send andRecv

/* REXX */
/* ********************************************************************** */
/* OBJECT REXX FOR WINDOWS 95/ WINDOWS NT                       */
/* SAMPLE TCP/IP CLIENT CONNECTION TO AF/OPERATOR */
/* ************************************************************************ */
if RxFuncQuery("SockLoadFuncs")
then rc = RxFuncAdd("SockLoadFuncs","RxSock","SockLoadFuncs")
rc = SockLoadFuncs()
if rc = 0 then return

USER = 'TEST'
TCPHost = 'SYSA' /* HOSTNAME OF SERVER */
TCPAddr.!family = 'AF_INET'
TCPAddr.!port = '12345' /* PORTNUMBER OF SERVER */
/* INITIALIZE TCP/IP ENVIRONMENT */
PARSE VALUE SockInit() WITH RC
IF WORD(RC,1) = 0 THEN DO /* IF INITIALIZED SUCCESSFULLY */

/* RESOLVE HOSTNAME INTO DOTTED DECIMAL IP ADDRESS */
PARSE VALUE SockGetHostByName(TCPHost, TCPAddr.!) WITH RC
IF WORD(RC,1) THEN DO /* IF GETHOSTBYNAME SUCCESSFUL */
SAY 'HOSTNAME' TCPHost 'RESOLVED TO' TCPAddr.!addr

/* ALLOCATE SOCKET FOR COMMUNICATIONS */
PARSE VALUE SockSocket('AF_INET', 'SOCK_STREAM', '0') WITH SocketID
IF WORD(SocketID,1) <> -1 THEN DO /* IF SOCKET OBTAINED */
SAY 'SOCKET NUMBER=' SocketID


/* CONNECT TO SERVER */
PARSE VALUE SockConnect(SocketID, TCPAddr.!) WITH RC
IF WORD(RC,1) = 0 THEN DO /* IF CONNECTED OK */
  SAY 'CONNECTED TO' TCPHost '('TCPAddr.!:addr') PORT' TCPAddr.!:port
/* PERFORM HANDSHAKE */
RC = SEND_ID_REC()
IF WORD(RC,1) <> -1 THEN DO /* IF HANDSHAKE SUCCESS */
/* PROMPT USER FOR 'EXEC' COMMAND */
SAY 'ENTER COMMAND FOR AF =>';
PARSE UPPER PULL CMD;
/* PERFORM EXEC */
RC = SEND_EXEC_REC(CMD)
msg = RxMessageBox('REXX processing complete', 'End of REXX')
END
ELSE SAY 'CONNECT RC=' RC
ELSE SAY 'SOCKET RC=' RC
ELSE SAY 'GETHOSTBYNAME RC=' RC
ELSE SAY 'INITIALIZE RC=' RC
RETURN 0
/* *************************************************************/
/* SUBROUTINE TO SEND ID RECORD TO SERVER */
/* *************************************************************/
SEND_ID_REC: PROCEDURE EXPOSE USER SocketID
ID_LINK = LEFT('WIN'||USER,8) /* 8 CHARACTER LINKID */
ID_TYPE = LEFT('ID',8) /* 8 CHARACTER REQUEST TYPE */
ID_REPLY_LEN = reverse(D2C(100,2)) /* 2 BYTE REPLY LENGTH */
ID_DATA = 'NAME=REXX_CON;REPLY=YES;DATA=ASCII;CLIENTID='||USER
ID_DATA_LEN = reverse(D2C(LENGTH(ID_DATA),2)) /* 2 BYTE DATA LENGTH */
RC = DO_SEND(ID_LINK||ID_TYPE||ID_REPLY_LEN||ID_DATA_LEN||ID_DATA)
IF WORD(RC,1) <> -1 THEN DO /* IF SEND SUCCESSFUL */
/* GET 20 BYTE PACKET HEADER */
PARSE VALUE DO_RECV(20) WITH RC ID_REPLY
IF RC > 0 THEN DO /* IF HEADER RECEIVE OK */
  IF SUBSTR(ID_REPLY,9,8) == LEFT('ID',8) THEN DO
    REPLY_DATA_LEN = C2D(reverse(SUBSTR(ID_REPLY,19,2)))
    SAY 'ID HEADER=>' SUBSTR(ID_REPLY,1,16) 'LEN' REPLY_DATA_LEN
    /* GET VARIABLE LENGTH ID REPLY DATA */
    PARSE VALUE DO_RECV(REPLY_DATA_LEN) WITH RC ID_REPLY
    IF RC > 0 THEN SAY 'ID Reply =>' ID_REPLY
ELSE SAY 'RECV ID RC=' RC
END
ELSE SAY 'RECV ID RC=' RC
END
ELSE SAY 'SEND RC=' RC

RETURN RC

/* ************************************************** */
/* SUBROUTINE TO SEND DATA                           */
/* ************************************************** */
DO_SEND: PROCEDURE EXPOSE SocketID
ARG SEND_DATA
PARSE VALUE SockSend(SocketID,SEND_DATA) WITH RC LENGTH
IF WORD(RC,1) = -1 THEN SAY
  'SEND RC=' RC
ELSE SAY 'RECV EXEC RC=' RC
END
ELSE SAY 'SEND RC=' RC

RETURN RC

/* **************************************************** */
/* SUBROUTINE TO SEND EXEC RECORD TO SERVER          */
/* **************************************************** */
SEND_EXEC_REC: PROCEDURE EXPOSE USER SocketID
ARG SEND_DATA
EX_LINK = LEFT(TKN||USER,8)  /* 8 CHARACTER TOKEN */
EX_TYPE = LEFT('EXEC',8)      /* 8 CHARACTER REQUEST TYPE */
EX_REPLY_LEN = reverse(D2C(3000,2)) /* 2 BYTE REPLY LENGTH */
EX_DATA = SEND_DATA
EX_DATA_LEN = reverse(D2C(LENGTH(EX_DATA),2)) /* 2 BYTE DATA LENGTH */
RC = DO_SEND(EX_LINK||EX_TYPE||EX_REPLY_LEN||EX_DATA_LEN||EX_DATA)
IF WORD(RC,1) <> -1 THEN DO   /* IF SEND SUCCESSFUL */
  /* GET 20 BYTE PACKET HEADER */
  PARSE VALUE DO_RECV(20) WITH RC EXEC_REPLY
  IF RC > 0 THEN SAY
    'EXEC REPLY ='
    EXEC_REPLY
  ELSE SAY
    'RECV EXEC RC=' RC
  END
ELSE SAY 'RECV EXEC RC=' RC
END
ELSE SAY 'SEND RC=' RC

RETURN RC
RETURN RC
/* ************************************************* */
/* SUBROUTINE TO RECEIVE DATA */
/* ************************************************* */
DO_RECV: PROCEDURE EXPOSE SocketID
ARG RECV_MAX
RECV_DATA = "
RC = 0
DO WHILE RC <> -1 & RECV_MAX > 0
PARSE VALUE SockRecv(SocketID, BUFF, RECV_MAX) WITH RC
IF RC = 0 THEN DO
   SAY 'ZERO BYTES RECEIVED = CONNECTION TERMINATED'
   LEAVE
END
IF RC <> -1 THEN DO
   RECV_MAX = RECV_MAX - RC
   RECV_DATA = RECV_DATA||BUFF
END
ELSE SAY 'RECV RC=' RC BUFF
END
RETURN RC RECV_DATA
Introduction

AF/OPERATOR provides two distinct levels of open TCP/IP communications features that enable AF/OPERATOR to communicate with many other TCP/IP-based applications on a variety of other platforms supporting TCP/IP.

In the case where you need to make a stream socket connection from AF/OPERATOR to a server application or to connect a client application to AF/OPERATOR, you may need to obey the protocol imposed by that application. In these circumstances, you will need to disable the protocol that AF uses for inter-communications with other AF systems by using the AF_PACKET(OFF) option on either the LINK DEFINE or COMM START command.

**Note:** The simplest method of delivering a standard request to an AF system is through the AF communications protocol. This is the default and you should be sure you understand the ramifications of disabling it by means of the AF_PACKET(OFF) options.

Any communications between two Candle AF systems is generally going to be simpler using the built-in communications protocol accessed using the higher level Candle API function, COMSDRCV, or, in AF/Remote, using the AFR_PEER functions. If you are writing your own TCP/IP application, you should consider using the AF TCP/IP communications protocol which may prove the best method of delivering an event to AF/OPERATOR as opposed to transmitting a data stream using AF_PACKET(OFF).

For information see “Advanced Communications--AF TCP/IP Communications Protocol” on page 124.

Stream sockets

Using AF_PACKET(OFF) on either the LINK DEFINE or COMM START command for TYPE(TCPIP) communications removes the AF packet header from TCPIP connections.

This gives the AF TCPIP communications a more socket-like character, causing them to take on the stream nature of the underlying TCP/IP protocol.

A fundamental characteristic of a stream socket is that data is a continuous stream of bytes, without boundaries. It is the responsibility of the user's REXX application to impose a structure on it.

Connection ownership

As mentioned above, when using the AF_PACKET(OFF) option for a connection, data is transmitted as a continuous stream of bytes.

This means that AF only permits one match to access the connection at any one time. Failure to limit access means that multiple matches could receive different segments of the same stream. For this reason we assign “ownership” of the connection to an individual match.

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.

It is possible to transfer ownership of an active connection through use of the COMADMIN REXX function with its GIVE and TAKE parameters. For information about COMADMIN, see the appropriate section in the *AF/OPERATOR Command Reference Manual*.

If a match ends while it owns an active connection, the connection is automatically terminated.
Automated LINK management

Introduction
This section describes how to set up AF/OPERATOR so that it will automatically establish communications links at product startup and keep them active.

Automatically establishing links at product startup
Link definitions can be stored in a member in the same PDS used for the startup parameters member. This is the PDS referenced by the RLANPAR DD in the AF/OPERATOR startup procedure.

The default member name for the link definitions member is LINKDEFS. However, you can override this default name by means of the LINKDEFS(xxxxxxxx) startup parameter.

Syntax rules
Each line can contain all of the keywords normally used on the LINK DEFINE command.

Automatic LINK activation
Including the ACTIVATE keyword on the definition will cause the product to automatically attempt to start the connection.

Examples
In the example that follows, three links are defined at product startup. Use of the ACTIVATE keyword means an attempt is made to activate the links named LINKSYSB and LINKSYSC.

```
LINK(LINKSYSA) HOST(SYSA) PORT(12345)
LINK(LINKSYSB) HOST(192.168.0.1) PORT(12345) ACTIVATE
LINK(LINKSYSC) GROUP(KOGXCF01) MEMBER(SYSCSRVR) ACTIVATE
```

Timing
Any activation requests for the links defined in LINKDEFS are initiated before the initial command is issued at product startup.

By default, AF/OPERATOR will allow up to 30 seconds for any links to become active before issuing the initial command. However you can adjust this value if necessary by using the TIMEOUT option of the LINKDEFS startup parameter.

Automatically keeping links active
Including the RECOVERY keyword (in addition to the ACTIVATE keyword) on the definition causes AF/OPERATOR to attempt to reestablish a connection should the remote application not be available at product startup or in the event that the remote application becomes unavailable at some later time.

Polling
With the exception of XCF links, OG/MVS will check inactive links every minute to determine if an automatic restart is required.

**Note:** The ACTIVATE and RECOVERY keywords are standard keywords of the LINK DEFINE command and are not restricted to being used in the LINKDEFS member. See the AF/OPERATOR Command Reference Manual for the syntax of the LINK DEFINE command.
Automated LINK management
Introduction

POVI (Programmerless Open VTAM Interface) is an AF/OPERATOR automation interface with VTAM 3270 applications. This chapter describes POVI, and gives examples of POVI implementations that simplify common data center tasks and solve common data center problems.

Chapter Contents

What is POVI? ................................. 152
The POVI Programming Environment ...................... 153
Logging on to POVI ................................ 155
POVI REXX Functions ............................ 158
Example: POVI Logs Onto TSO and Submits Batch Job ....... 161
Recording a POVI script ............................ 164
Using Variables in a Script ......................... 169
POVI Utilities .................................... 172
Setting POVI Options ............................ 174
Operator Facility .................................. 175
Handling Errors .................................. 179
POVI Security ................................... 182
POVI Environment Commands ....................... 184
What is POVI?

POVI lets you automate interactions with VTAM 3270 applications and provides two REXX based methods to accomplish this task.

- You can use the POVI REXX functions to create REXX execs that pass information between AF/OPERATOR and VTAM application sessions. POVI can be instructed to capture data by parsing an application’s screen and to insert data and variables into specific screen fields or cursor locations. After you create a REXX exec, you can instruct AF/OPERATOR to execute it automatically when triggered by an AF/OPERATOR trap. AF/OPERATOR uses the POVI Playback Manager during REXX exec execution to accomplish your task.

- You can use POVI’s Keystroke Recorder to create scripts, sequences of keystrokes and variables. POVI can record everything you type as you “run through” a task and save this information for playback. After you create a script and test it, you create a REXX exec to execute it. When triggered by a trap, AF/OPERATOR executes the REXX exec that plays back your script and uses the Playback Manager to take the desired actions.

Comparison of automation methods

If your task is to build production applications that are easily debugged and maintained, the superior error checking offered by POVI REXX functions makes them the preferred automation method.

Users new to automation may find the Keystroke Recorder to be a more intuitive method; it can be used to automate 3270 applications quickly. Scripts developed with the Keystroke Recorder can be included in REXX execs that use the POVI functions.

The PEEK facility (accessed from the POVI Recording Manager pulldown menu) can be used to debug REXX execs developed with either automation method. Access to the PEEK facility may be restricted to a list of authorized personnel.

POVI Online Help

You can get Help for any panel in POVI by pressing F1. Online Help includes detailed information about all POVI options you can adjust from the pull-down menus and many other POVI topics. A POVI Help Index is available from the Help pull-down of the Session Selection panel. A Glossary and Search capability are also included.

Take advantage of POVI Online Help. Since Help information is included as part of POVI, it can be updated when your site applies Candle cumulative maintenance. When you press F1, you have access to the latest information available.

Migration from previous versions

Most POVI environment commands are replaceable with POVI REXX functions which provide superior error-checking. The “see also” entries in “POVI Environment Commands” on page 184 point you to the appropriate POVI REXX function to look up in the AF/OPERATOR Command Reference Manual.
The POVI Programming Environment

The following figure illustrates the relationship between POVI and other parts of the system. This complex environment should only be set up or modified using the Customizer. Do not attempt to add sessions to the POVI Recording Manager without using the Customizer. The Customizer ensures that any session defined to the Recording Manager is also defined to the Playback Manager, enabling you to successfully playback the REXX execs you create.

The POVI programming environment encompasses the address spaces of AF/OPERATOR, CT/Engine, and your VTAM 3270 application sessions. It is connected by 2 VTAM sessions.

POVI uses CT/Engine services to establish VTAM 3270 application sessions. So in addition to examining messages in the AF/OPERATOR log when you debug REXX execs, also examine the messages output to the CT/Engine log.
POVI applids

AF/OPERATOR uses the default applids in the following table to support POVI. These applids are defined during AF/OPERATOR customization. They function as POVI applications which you can use to accomplish various tasks. The Recording Manager and Playback Manager are the most-often-used.

<table>
<thead>
<tr>
<th>Default APPLID</th>
<th>Component</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVIUSER</td>
<td>Recording Manager</td>
<td>Lets you (the interactive user) record and test scripts.</td>
</tr>
<tr>
<td>OVIAO</td>
<td>Playback Manager</td>
<td>Lets REXX execs running on AF/OPERATOR start and control VTAM applications.</td>
</tr>
<tr>
<td>OVICUA</td>
<td>Operator Facility, CUA Operator</td>
<td>Lets the POVI administrator issue commands to trace POVI internal functions or check the status of various resources and facilities. OVICUA offers a menu-driven CUA interface.</td>
</tr>
<tr>
<td>OVIOPER</td>
<td>Operator Facility, Basic Operator</td>
<td>Lets the POVI administrator issue commands to trace POVI internal functions or check the status of various resources and facilities. OVIOPER offers a “classic” console-like interface for the convenience of customers migrating from previous versions.</td>
</tr>
</tbody>
</table>
Logging on to POVI

Types of logons

POVI logons are user initiated or AF/OPERATOR initiated. User initiated logons take place at a physical terminal, while AF/OPERATOR initiated logons take place during the execution of a REXX exec (usually in response to an AF/OPERATOR trap whose ACTION is to execute your REXX exec).

<table>
<thead>
<tr>
<th>Type of logon</th>
<th>Default APPLID</th>
<th>POVI component</th>
</tr>
</thead>
<tbody>
<tr>
<td>User initiated</td>
<td>OVIUSER</td>
<td>Recording Manager</td>
</tr>
<tr>
<td>AF/OPERATOR initiated</td>
<td>OVIAO</td>
<td>Playback Manager</td>
</tr>
<tr>
<td>User initiated</td>
<td>OVICUA</td>
<td>CUA Operator</td>
</tr>
<tr>
<td>User initiated</td>
<td>OVIOPER</td>
<td>Basic Operator</td>
</tr>
</tbody>
</table>

Logging on to POVI from a VTAM terminal

To log on to the POVI Recording Manager from a VTAM network terminal, enter the AF/OPERATOR LOGON command:

```
LOGON APPLID(OVIUSER)
```

where OVIUSER is the Candle default applid of the POVI Recording Manager. Use the Recording Manager applid that was defined at your site during customization.

**Result:** The Recording Manager Session Selection panel lists the VTAM applications you can access. Selections can be added to this menu by using the Customizer to define more accessible VTAM 3270 applications.

You can use the same method to log on to POVI's Operator facility. See “Operator Facility” on page 175 for an example.
Recording Manager Session Selection panel

From the Recording Manager Session Selection panel, you can:

- start a session with a VTAM application
- further customize POVI
- invoke utilities to
  - import and export Keystroke Recorder scripts to a partitioned data set (PDS)
  - debug applications by viewing the 3270 application screen of the application being controlled by a REXX exec executing on AF/OPERATOR
- issue POVI environment commands such as \RC, the command that starts the Keystroke Recorder

For basic information about POVI environment commands, see “POVI Environment Commands” on page 184. For detailed command syntax information, see the AF/OPERATOR Command Reference Manual.
How AF/OPERATOR logs on to POVI

AF/OPERATOR logs on to the POVI Playback Manager and starts a session with a VTAM application during execution of a REXX exec you create.

To successfully instruct AF/OPERATOR to log on to (and off of) POVI, include the following statements in your REXX exec:

1. Call the Candle-supplied REXX function POVILGON. For example:

   ```ruby
toolsrc = POVILGON(OVIAO, sessid)
   IF (toolsrc <> 0) THEN CALL ERROR_ROUTINE
   .
   .
   .
   
   where `sessid` is the name of a VTAM application you can access (a name on the POVI Session Selection panel).

   In your POVILGON command, be sure to specify the APPLID of the Playback Manager, *not* the Recording Manager.

   **Result:** AF/OPERATOR logs on to POVI and starts the VTAM session you specified.

2. At the end of your REXX exec, call the Candle-supplied REXX function POVILGOF:

   ```ruby
toolsrc = POVILGOF()
   
   The POVILGOF function accepts no parameters. POVI keeps track of the application you logged on to and logs you off.

   **Result:** AF/OPERATOR ends the session and logs off POVI.
POVI REXX Functions

Introduction
Candle supplies a set of external REXX functions you can call from your execs. These functions perform common actions such as logging on to and off POVI, playing back scripts, positioning the cursor, and finding a specified string.

Why use POVI REXX functions?
The advantages of using the POVI REXX functions for automating your application are:

**Performance**
The benefits of compiled REXX are exploited.

**Environment Management**
The POVI functions manage the tasks of logging on and addressing various VTAM applications for you. Using the POVI functions can reduce the amount of coding required to accomplish these tasks.

**Enhanced Error Checking**
The functions provide a set of return codes which, should an error occur, allow you to easily trace and correct the problem. Without this type of error-checking, any unusual application behavior could cause a terminal input error.
**Summary of POVI functions**

The POVI REXX functions are summarized in the following table. For more detailed information about these functions, see the *AF/OPERATOR Command Reference Manual*.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POVIFIND</td>
<td>Finds a string in the terminal buffer, and returns the row and column of either the start of the string or an offset from the start.</td>
</tr>
<tr>
<td>POVILGOF</td>
<td>Logs off POVI and the associated VTAM application.</td>
</tr>
<tr>
<td>POVILGON</td>
<td>Logs on to and addresses POVI, and starts a session with a VTAM application.</td>
</tr>
<tr>
<td></td>
<td><em>Note:</em> For a given match, no more than one session, started using POVILGON, can be concurrently active.</td>
</tr>
<tr>
<td>POVILOC</td>
<td>Positions the cursor on a simulated 3270 screen.</td>
</tr>
<tr>
<td>POVIPBS</td>
<td>Plays back a script.</td>
</tr>
<tr>
<td>POVISEND</td>
<td>Writes data to the current cursor location.</td>
</tr>
<tr>
<td>POVIDLAY</td>
<td>Waits a specified number of seconds, then refreshes the terminal buffer.</td>
</tr>
<tr>
<td>POVIDATA</td>
<td>Retrieves data from a specified screen location for a specified length.</td>
</tr>
<tr>
<td>POVISAVE</td>
<td>Returns a handle for either the current POVI session that was started using POVILGON or the specified session which was started using the native LOGON command.</td>
</tr>
<tr>
<td>POVISEND</td>
<td></td>
</tr>
<tr>
<td>POVISAVE</td>
<td></td>
</tr>
<tr>
<td>POVIVSET</td>
<td>Sets a CT/Engine variable to a specified value.</td>
</tr>
<tr>
<td>POVIRSTR</td>
<td>Restores a session environment for use by POVI REXX tools from a handle created using POVISAVE.</td>
</tr>
</tbody>
</table>
Using POVI REXX functions

Creating a successful REXX exec using the POVI REXX functions may seem time consuming at first. The following suggestions are offered to speed the development process.

- Plan out in advance every keystroke you make when interacting with your VTAM 3270 application. The best way to do this is to perform the operation manually and take note of your every move. Your notes should include:
  - the data you input
  - the row and column coordinates where you input this data
  - any key you press
  - the expected response
  - the row and column coordinates where you expect this response

- The exec you create to interact with a VTAM 3270 session should:
  - logon using the POVILGON function. Calls to other POVI REXX functions operate within the environment created by the POVILGON function. You can optionally specify the recording terminal’s model number to avoid logmode inconsistencies during playback of scripts.
  - check the value returned by every POVI REXX function you call. A return value of 0 (zero) indicates successful completion.
  - logoff using the POVILGOF function. This function handles de-addressing the environment in which your POVI session runs and terminates the session.
Example: POVI Logs Onto TSO and Submits Batch Job

Sample REXX exec

This sample REXX exec can be called by AF/OPERATOR to automatically log onto TSO, submit a batch job, write a message to the console, and log off.

FIGURE 15. Programmerless Open VTAM Interface Sample REXX Exec

/* REXX */ /* pbmappl - POVI Playback manager VTAM applid */ application - VTAM 3270 application name (as appears on the Recording Manager Session Selection panel */ model - 3270 model number address MVS PARSE UPPER ARG pbmappl, application, environment, model NEWSTACK toolsrc = POVILOGON(pbmappl, application, environment, model) IF (toolsrc <> 0) THEN CALL ERROR_ROUTINE toolsrc = POVISET(USERID, USER01) IF (toolsrc <> 0) THEN CALL ERROR_ROUTINE toolsrc = POVISET(PASSWORD, MYPASS) IF (toolsrc <> 0) THEN CALL ERROR_ROUTINE toolsrc = POVIPBS(TSOLOGON) IF (toolsrc <> 0) THEN CALL ERROR_ROUTINE toolsrc = POVIDATA(1, 27, 8) IF (toolsrc <> 0) THEN CALL ERROR_ROUTINE PULL ISPFPDF IF (ISPFPDF <> "ISPF/PDF") THEN CALL ERROR_ROUTINE toolsrc = POVISEND("", CLEAR) IF (toolsrc <> 0) THEN CALL ERROR_ROUTINE toolsrc = POVISEND("SUBMIT 'PROD.CNTL(MYJOB)'", ENTER, "SUBMITTED") IF (toolsrc <> 0) THEN CALL ERROR_ROUTINE toolsrc = POVIDATA(2, 1, 80) IF (toolsrc <> 0) THEN CALL ERROR_ROUTINE PULL submit_message
Example: POVI Logs Onto TSO and Submits Batch Job

In the example above:

- **NEWSTACK** is used to create a stack on which POVI functions may store data. **DELSTACK** is used to delete the stack when it is no longer needed.

- **POVILGON** instructs AF/OPERATOR (using the Playback Manager) to log on to an accessible VTAM application (one listed on POVI's Session Selection panel).

- **POVISET** initializes values for the variables USERID and PASSWORD. POVI will use these values when it plays back the TSOLOGON script.

- **POVIPBS** plays back the prerecorded script TSOLOGON to logon to TSO.

- **POVIDATA** returns data from the screen. After the call to POVIPBS, the POVIDATA function is used to check that the expected screen is present. The code fragment:

```plaintext
  toolsrc = POVIDATA(1, 27, 8)
  IF (toolsrc <> 0) THEN CALL ERROR_ROUTINE
  PULL ISPFPDF
  IF (ISPFPDF <> "ISPF/PDF") THEN CALL ERROR_ROUTINE
```

captures the data at row 1, column 27, for a length of 8 characters. The POVIDATA function stores data on the REXX data stack. The data is retrieved using the PULL instruction and stored in a variable named ISPFPDF. If this variable does not contain the expected data (in this case, the 8-character string “ISPF/PDF” from line 1, column 27 of the ISPF/PDF Primary Option Menu) then the error routine is invoked.

- **POVISEND** is invoked twice in the example. The first call clears the screen.

```plaintext
  toolsrc = POVISEND("", CLEAR)
```

The second call submits a batch job, simulates pressing the Enter key, and verifies that the job is successfully submitted.

```plaintext
  toolsrc = POVISEND("SUBMIT 'PROD.CNTL(MYJOB)'",ENTER, "SUBMITTED")
```
Example: POVI Logs Onto TSO and Submits Batch Job

If, after the simulated ENTER key is pressed, the word SUBMITTED does not appear on the automated application’s next screen, the POVISEND function returns a function code 22.

- POVIDATA is used again to return data from the screen. The line:
  \[ \text{toolsrc = POVIDATA(2, 1, 80)} \]
captures the data at row 2, column 1, for a length of 80 characters and stores the data on the REXX data stack.

- throughout the example, the return code value for each function is stored in the variable \text{toolsrc}. This value is checked at every step. The line:
  \[ \text{IF (toolsrc} \neq \text{0) THEN CALL ERROR\_ROUTINE} \]
transfers program control to an error handling routine if a non-zero return code (indicating the REXX function failed) is detected.

- The ERROR\_ROUTINE part of this example gathers diagnostic information using the PARSE command and stores it in variables named INVOKE\_TYPE, EXEC\_NAME, and THE\_REST. It uses the AF/OPERATOR function GLBVGET to retrieve the value of AOPVFNC. AOPVFNC is an automatically-created variable that is set to each POVI function’s name when the POVI function is invoked. So, if an error has just occurred, the value of AOPVFNC contains the name of the POVI function that was being used at the time.

- The line:
  \[ \text{PARSE PULL feedback} \]
retrieves diagnostic information from the stack and stores it in a variable named FEEDBACK.

- The line:
  \[ \text{CALL ERRHNDLR sessid, exec\_name, sigl, aopvfnc, toolsrc, feedback} \]
invokes an external error-handling function called ERRHNDLR and passes the diagnostic information as parameters to the function.

Look in the sample library that accompanies AF/OPERATOR for other REXX exec samples and for a sample error handling routine. Error routines can:

- identify the line on which an error occurs
- perform error-analysis and recovery/retry
- print (or write to a screen) diagnostic information such as:
  - function return codes
  - data that the POVI functions store on the stack
  - the current screen of the application you are controlling
Recording a POVI script

Introduction

This section explains how to use the POVI Keystroke Recorder to create and test a simple script.

To create a simple script, you will follow a three-step process:

1. Start a session with a VTAM application using the Recording Manager.
2. Record the script.
3. Test the script by playing it back.

If you need help while using POVI or if you receive a POVI error message, and you want more explanation of the problem and suggestions for solving it, press F1.

Starting a Recording Manager session

To start a VTAM session:

1. From a network terminal, log on to POVI’s Recording Manager by issuing this command:

   LOGON APPLID(OVIUSER)

   In your LOGON command, be sure to specify the APPLID of the Recording Manager, not the Playback Manager.

   Result: The Session Selection panel is displayed.

2. On the Session Selection panel, type S (Start) or type a / beside the session ID of the application you want to access. Press Enter.

   Result: The initial panel for the application you selected is displayed.
### Recording the script

To record your script:

1. **Type** `RC` **in any unprotected input field of the application**, and press Enter.

   **Result:** The POVI Keystroke Recorder’s Script Selection panel is displayed. This panel allows you to select an existing script or to begin recording a new script.

   ![POVI Keystroke Recorder Script Selection Menu](image)

2. **At the Name prompt**, type the name you want to give the new script, and press Enter. The name can be up to 8 characters long.

   **Result:** The Break Key Selection panel is displayed. This panel lets you select a key that tells the Keystroke Recorder to stop recording.

   ![POVI Keystroke Recorder Break Key Selection Panel](image)
Recording a POVI script

3. (Optional) At the **Description** prompt, type a longer description of the script. Press Enter.

4. Follow the instructions on the panel to select a Break key.

   The Break key can be the Clear key, any function key, or any PA key. When you press the Break key while recording a script, the Keystroke Recorder stops recording and displays the Option Menu.

   Select your Break key with care. Once you are recording a script, you cannot use the Break key in the application because the Recording Manager will stop.

   **Result:** The Break Key Verification panel is displayed.
5. Confirm or change your Break key selection.
   - To confirm your selection, press the Break key you selected.
     **Result:** The Keystroke Recorder returns you to your application to begin recording.
   - To change the Break key, press any key other than the one you selected.
     **Result:** The Keystroke Recorder returns you to the Break Key Selection panel, where you can select a different Break key.

6. As you interact with the application, all of your keystrokes are recorded in the script.

7. When you finish the script, press the Break key.
   **Result:** The Option Menu is displayed.
8. To save the script, type S (Select) to the left of **Save recording**, and press Enter.
   
   **Result:** POVI saves the script and returns you to the application.

**Playing back the script**

To make sure your script runs properly, play it back. Use the same model terminal you used for recording the script.

To play back the script follow these steps:

1. Select the same application from the Session Selection panel.
2. At the same input field where you began recording the script, type \RC and press Enter.
   
   **Result:** The Script Selection panel is displayed.

3. Find your script in the list, type S (Select) to the left of the script’s name, and press Enter.

   **Result:** The script runs immediately.

   **Tip:** If the size of the input field permits, you can type \RC **scriptname** directly into the input field and press Enter. This allows you to bypass the Script Selection panel.
Using Variables in a Script

Introduction

To make your scripts more useful and versatile, you can define variables and insert them into application fields while the script is being recorded. When the script is called by a REXX exec, POVI captures the value of each variable and substitutes that value for the variable’s name while the script is running.

Once you have defined the variables and recorded the script, you can build a REXX exec that will run the script automatically when triggered by an AF/OPERATOR trap.

Defining and initializing variables for recording

Plan the variables you will use in your script. Consider the task’s requirements and its relationship to other tasks.

Now define and initialize each variable:

1. At the Command ===> line of the POVI Session Selection panel, type this command:
   \SET varname
   where varname is the name of a variable you will use in your script.

   Tips:
   A. Do not use an ampersand (&) in the variable name when you are setting the variable. You will add an ampersand to the beginning of each variable name when you record the script.
   B. You can use AF/OPERATOR-supplied system and match variables as values in place of literal values. They are listed in the AF/OPERATOR Command Reference Manual.

2. Press Enter.
   Result: A blank panel is displayed.

3. Type the initial value you want to assign to the variable, and press Enter. You can assign any value that is acceptable to the VTAM application in which you will record the script. A real value will be assigned during execution of the REXX exec.
   Result: The cursor is repositioned at the Command ===> line of the Session Selection panel.

4. Repeat the procedure until you have defined and initialized all the variables you want to use.
Recording the script

To record a script that uses variables:

1. From the Session Selection panel, start the VTAM application.
2. Use the \RC command to begin recording.
3. While you are recording the script, you can type variable names into the application’s input fields. Use the variable names you set, prefixed by an ampersand (&).
4. When the script is played back, POVI substitutes the value of the variable for the variable name.
5. Use the Break key to stop recording.
6. Save the script.

Testing the script manually

Now play back the script to make sure it runs properly.

1. At the Command ===> line of the Session Selection panel, reissue the same \SET commands you issued before recording the script.
   Before you play back a script interactively (rather than playing it back automatically during execution of a REXX exec), you must reinitialize the script’s variables. Otherwise, the script substitutes null values for uninitialized variables, and it does not run properly.
2. Select the same VTAM application.
3. Use the \RC command to play back the script.
   Tip: You can bypass the Script Selection panel by issuing this command from any unprotected input field of the application:

   \RC scriptname

   The script will begin to run immediately.

Modifying the script

If your script does not work the way you want it to, you can modify it rather than re-recording it.

1. Issue the \RC command either from the Session Selection panel or from the VTAM application.
   **Result:** The Script Selection panel is displayed.
2. Type C (Change) to the left of the name of the script, and press Enter.
   **Result:** The Recorder Change Facility panel displays the first 10 lines of the script. You can use the function keys to scroll through the script.
3. To edit the script, follow these instructions.

<table>
<thead>
<tr>
<th>If you want to ...</th>
<th>Do this ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repeat a line</td>
<td>Type R (Repeat) to the left of the line, and press Enter.</td>
</tr>
</tbody>
</table>
When you have finished editing, press F3 to save the script and exit.

**Writing an exec and a trap**

Once you are satisfied with your script, you need to write:

- a trap that causes execution of the REXX exec in response to a specified system event.
  
  For detailed information on traps, see “Trapping System Events” on page 35.

- a REXX exec that logs on to POVI, sets variables, plays back the script, and handles errors.
  
  In writing your REXX exec, use the Candle-supplied REXX functions listed in “POVI REXX Functions” on page 158.

Before you play back a script automatically during execution of a REXX exec, you must reset the script’s variables. Otherwise, the script substitutes null values for uninitialized variables, and it does not run properly. The REXX exec example shown in “Programmerless Open VTAM Interface Sample REXX Exec” on page 161 shows how to set variables using the POVIVSET function and how to play back the script using the POVIPBS function.

Be sure to include some error checking in your REXX exec. At a minimum, your exec should check for return codes. You can also set the options that govern POVI’s operating characteristics for graceful error handling (see “Setting POVI Options” on page 174).

**Note:** Always play back scripts on the same model terminal on which they were recorded. Otherwise, screen coordinates will be incorrect and the script will not run properly.
POVI Utilities

Introduction

POVI provides the following utilities:

**Export/Import**

Allows you to make scripts created on one POVI available to another POVI. When you export a script, it is copied to a member of a partitioned dataset (PDS). The member has the same name as the script. Then you can import the script from the PDS to a different POVI.

**PEEK**

Allows you to look at the screens of other users logged on to POVI.

You can access these utilities by entering a command or by selecting the Utilities pull-down menu from the Recording Manager Session Selection panel.

---

**Export/Import**

**PEEK**

---

Exporting and Importing POVI scripts

To export or import a POVI script follow this procedure:

1. Allocate a PDS with the following characteristics:
   - record length of at least 512
   - record format of either V or VB

2. At the **Command== >** line of the Session Selection panel of the POVI in which you created the script, type **EXPORT** and press Enter.
   Alternately, you can select Export a Script from the Utilities pull-down.

   **Result:** The Script Export panel lists the scripts you can export.

3. On the Script Export panel:
   - At the **PDS Name** prompt, type the name of the PDS to which you want to export the scripts.
   - Type **S** or / to the left of the name of each script you want to export.
   - Press Enter.

   **Result:** The scripts are copied to the PDS you specified.

4. Log off the POVI from which you exported the scripts.

5. Log on to the POVI to which you want to import the scripts.
6. At the **Command===>** line of the Session Selection panel, type **IMPORT** and press Enter. Alternatively, you can select **Import a Script** from the Utilities pull-down.

**Result:** The Import PDS Name pop-up appears.

7. At the **Import PDS Name** prompt, type the name of the PDS from which you want to import scripts, and press Enter.

**Result:** The Script Import panel lists the scripts you can import.

8. On the Script Import panel, type **S** or **/** to the left of the name of each script you want to import, and press Enter.

**Result:** The scripts are imported to the POVI you are logged on to. You can now play back the imported scripts from both POVIS: the one you exported from, and the one you imported to.

### The PEEK function

The PEEK function lets you look at the screens of other users logged on to the Recording Manager or an Operator facility, or look at the Virtual Terminal screens controlled by REXX execs playing back on the Playback Manager. This can be a useful tool for debugging REXX execs you create. If extended security processing for POVI is being used at your data center, you must be authorized as a PEEK user before you can execute this function. For more information, refer to “POVI Security” on page 182.

### Using the PEEK function

To use the PEEK Function:

At the **Command===>** line of the Session Selection panel, type **PEEK** and press Enter. Alternatively, you can select **PEEK at a User’s Session** from the Utilities pull-down menu.

**Result:** You can view the Virtual Terminal screen of an interactive user by entering their user ID, or you can view the Virtual Terminal screen for an AF/OPERATOR-initiated automation session by entering the LU name of the AF/OPERATOR virtual terminal.
Setting POVI Options

Introduction

This section describes the various customizable options that control the default appearance and operating characteristics of POVI sessions.

POVI Admin pull-down

POVI provides the following types of administrative options:

Menu Characteristics

Allows you to adjust Main Menu type if you are migrating from a previous version of POVI.

Miscellaneous

Allows you to adjust the characteristics of all your POVI sessions, such as the POVI Trigger character, the level of diagnostics output to the CT/Engine log, and other aspects of session behavior. For adjustments to individual sessions see “POVI session-specific variables” on page 180.

Extended Security

You can restrict access to various resources and facilities within POVI. For example, access to the Playback Manager can be restricted by LU name prefix.

Timeout

Allows you to adjust default session timeouts at the request of Candle Support Services personnel. The default timeouts are suitable for most installations. See “Operator Facility” on page 175.

Note: Even though a POVI session times out, it may not be cancelled. To ensure cancellation, set the RETURN_TO_MAIN_MENU switch to NO.

POVI Options pull-down

The POVI Options pull-down, which you can access from the Session Selection panel, allows you to adjust display options on POVI panels.
Operator Facility

Introduction
POVI provides an Operator Facility that lets you issue commands that control the POVI address space and set POVI’s operating characteristics. Since POVI default operating characteristics are adequate for most sites, it is not necessary to use this facility. However, Candle Support Services personnel may ask you to log on to the Operator facility should you experience a problem. If necessary, commands to trace POVI internal functions or check the status of various resources can be issued from the Operator facility.

Note: Access to the Operator facility may be restricted to the POVI administrator at your site.

Operator facility applids
AF/OPERATOR provides two separate applids for the Operator facility. The two are identical in function but have different user interfaces. We expect most users will prefer the newer CUA Operator. The “classic” Operator is provided for customers who implemented automation routines using previous versions of Candle products.

<table>
<thead>
<tr>
<th>Applid</th>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVICUA</td>
<td>CUA Operator</td>
<td>Menu-driven panels</td>
</tr>
<tr>
<td>OVIOPER</td>
<td>Basic Operator (“classic”)</td>
<td>Console-like panels</td>
</tr>
</tbody>
</table>

Notes:
1. Operator facility applids are defined during customization and may differ from those listed above.
2. Depending on how POVI is customized at your site, the Operator facility may be listed as a session on your POVI Recording Manager Session Selection panel.
Starting and stopping operator facility sessions

If it is listed, you can log on to the Operator facility from the POVI Recording Manager Session Selection panel. This depends on how POVI was customized at your site.

To log on to the Operator facility from a VTAM network terminal:

- Enter the AF/OPERATOR LOGON command:

  \[ \text{LOGON APPLID(OVICUA)} \]
  
  where OVICUA is the Candle default applid of the POVI CUA Operator. Use the CUA Operator applid that was defined at your site during customization. (An alternative is to use OVIOPER, the Candle default applid of the POVI Basic Operator.)

  \textbf{Result:} The Operator panel appears.

To stop an Operator facility session:

- From the CUA Operator panel, press F3. You will be prompted to confirm that you want to exit the session. (Alternatively, from the Basic Operator panel, issue the LOGOFF command without specifying a session ID.)

  \textbf{Result:} POVI terminates the Operator session and returns to the panel you logged on from.

Issuing operator commands

POVI operator commands are used to control the POVI address space. Unless otherwise noted, you can issue these commands from:

- CUA Operator
- Basic Operator
- MVS console

To issue an operator command from the CUA Operator facility, select it from a pull-down menu, or type the command on the command line and press Enter.

To issue an operator command from the Basic (“classic”) Operator facility, type the command on the command line and press Enter.

To issue a POVI operator command from the MVS console:

Enter the command as follows:

  \[ \text{F poviproc,command} \]

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.
**Summary of operator commands**

The POVI operator commands are summarized below. For more detailed information, see the *AF/OPERATOR Command Reference Manual*.

**Note:** The following commands are available as pull-down items from the CUA Operator facility, so mastering the syntax is only necessary if you want to issue the commands from an MVS console or from the Basic Operator.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECHO</td>
<td>Writes a string to the terminal and empties the message buffer. You can use the ECHO command to verify that the Operator facility is working properly and to force all buffered messages to the log. The syntax is:</td>
</tr>
<tr>
<td></td>
<td>ECHO string</td>
</tr>
<tr>
<td>LOGOFF</td>
<td>Logs a session off POVI. The syntax is:</td>
</tr>
<tr>
<td></td>
<td>LOGOFF session</td>
</tr>
<tr>
<td>MONITOR</td>
<td>Determines which message types are displayed on the Operator panel. The syntax is:</td>
</tr>
<tr>
<td></td>
<td>MONITOR pccccc</td>
</tr>
<tr>
<td></td>
<td>where p specifies enabling (+) or disabling (-) of a message class (cccccc). The available message classes are: ALERT, ALL (all message types), INFO, LOG, REPLY, VIEW, WARN</td>
</tr>
<tr>
<td></td>
<td>Under normal conditions, you probably want to monitor only the LOG messages, but for diagnostic activities issue the command MONITOR + ALL. <strong>Note:</strong> The MONITOR command can be issued only from a POVI Operator facility, not from the MVS console.</td>
</tr>
<tr>
<td>REFRESH</td>
<td>Loads a new copy of a member in your panel library. The syntax is:</td>
</tr>
<tr>
<td></td>
<td>REFRESH PANEL name</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>VSHOW</td>
<td>Displays the status of POVI users, devices, or sessions. To display the status of a user, issue:</td>
</tr>
<tr>
<td></td>
<td><strong>VSHOW userid</strong></td>
</tr>
<tr>
<td></td>
<td>To display the status of all active users, terminals, and sessions, issue:</td>
</tr>
<tr>
<td></td>
<td><strong>VSHOW /</strong></td>
</tr>
</tbody>
</table>
Handling Errors

Introduction
This section discusses the most common types of errors you may encounter when automating 3270 VTAM applications and some adjustments to error-detection you can make at run time.

Are your panels stable?
Consider the panels that you use to interact with your VTAM 3270 application. If an ISPF-based panel should change in some way, perhaps because a new version of the application has been installed, or because a panel is customized for your site, a POVI script or function-based REXX exec may cease to function properly. If a previously working POVI REXX exec suddenly is not working, this is one of the first things you might want to investigate (after making sure your VTAM nodes are varied active).

Detecting a terminal input error
The ability to detect a terminal input error is important since these types of errors may hang your automation routine until it eventually times out. If you are using the POVI REXX functions, this error will be indicated by return code 12. You can use the PEEK facility to determine what step in a sequence of automated tasks did not successfully complete. The PEEK facility will display the panel on which a terminal input error occurred.

A terminal input error occurs when data is sent to a protected screen location. There are many ways that a terminal input error can be generated, some common ways are:

- when your REXX exec types in a value that is too long for a particular input field.
- when the VTAM application you are controlling generates an unexpected asynchronous message, such as an HSM archive message or a TSO send message. The controlling REXX exec is programmed to type data to an input field on a particular panel but instead types it to an unexpected panel that may not contain an input field.

You should include code in your automation routine to terminate, or to retry by logging off then on again, if a terminal input error is detected.
Detecting CT/Engine errors

The POVI REXX functions detect CT/Engine errors when the **Return to Main Menu on Error** option is set. This is one of the default settings on the Admin pull-down, Menu Characteristics option. Automatic notification (by function return codes) of a return to the Playback Manager Main Menu is provided only if you are using POVI REXX functions.

If you are not using REXX functions, and your session encounters a CT/Engine error, you are not automatically notified of this event. Instead, the session hangs and eventually times out. You should include error recovery code in your automation routine to handle this type of error.

POVI session-specific variables

The choices accessible on the POVI Recording Manager Admin pull-down set default values for all POVI sessions. Some of the same Admin choices are also accessible as CT/Engine variables. This gives you the flexibility of adjusting the values of these variables for a particular automation session. For example, if you are testing or debugging a new automation routine, you can set a much higher level of diagnostics than usual for that particular session.

The variables you may adjust for a particular session are described in the table below.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>KABLOG</td>
<td>Output diagnostics level. The value of this variable determines the detail and number of output messages to the CT/Engine log. Each ascending range of numbers adds messages to be output. The ranges are ordered from most often to least often needed message types.</td>
<td>100</td>
</tr>
<tr>
<td>Range</td>
<td>Message Type</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Error messages</td>
<td></td>
</tr>
<tr>
<td>100-199</td>
<td>Successful logon to POVI</td>
<td></td>
</tr>
<tr>
<td>200-299</td>
<td>Logmode related information</td>
<td></td>
</tr>
<tr>
<td>300-399</td>
<td>Session start information</td>
<td></td>
</tr>
<tr>
<td>400-499</td>
<td>Trace of data flow and commands</td>
<td></td>
</tr>
<tr>
<td>500-599</td>
<td>Trace of POVI environment command execution</td>
<td></td>
</tr>
<tr>
<td>600-699</td>
<td>Trace of the POVI Keystroke Recorder</td>
<td></td>
</tr>
</tbody>
</table>
### Handling Errors

**KABRMENU**  
Return to Main Menu on error. A null value for this variable turns off the return to Main Menu on error function. Any other value causes a return to the Playback Manager Session Selection panel in the event of a CT/Engine error and an error message. The user can then press F1 (or code automation instructions to that effect) to obtain an explanation of the error and a recommendation for correcting it.  

Default value: Non-Null

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>KABRMENU</td>
<td>Return to Main Menu on error. A null value for this variable turns off the return to Main Menu on error function. Any other value causes a return to the Playback Manager Session Selection panel in the event of a CT/Engine error and an error message. The user can then press F1 (or code automation instructions to that effect) to obtain an explanation of the error and a recommendation for correcting it.</td>
<td>Non-Null</td>
</tr>
</tbody>
</table>

**KABMTYPE**  
Main Menu type. A null value results in a Playback Manager Session Selection panel that looks like the panel in previous versions of POVI. Any other value results in a Playback Manager Session Selection panel that reflects functions available in AF/OPERATOR.  

Default value: Non-Null
Introduction

External security for POVI (you may see the term basic security in POVI online Help), for example, ACF2 security, is defined using the Customizer. People who are given external security access can log on to POVI. See the AF/OPERATOR Configuration and Customization Guide and the Customizer’s online Help for instructions on configuring POVI external security.

POVI also offers extended security options which allow you to define more levels of access. For example, Admin authority can be restricted to a subset of POVI users. The way you implement extended security for POVI depends on how you customized external security. Extended security can use either POVI’s internal security package (NAM) or the same external security package you used for Customization. You may see the terms internal extended security and external extended security in POVI online help.

Extended security

Extended security allows your site to limit access (by user ID) to any or all of the following POVI resources and features.

- Admin pulldown menu
- PEEK function
- Script Export/Import functions
- Playback Manager
- Recording Manager

The default extended security settings for POVI give everyone who can log on to POVI access to all features.

An effective minimal implementation of extended security is to restrict access to the Admin pulldown, leaving only a few people with Admin authority. People who have access to this menu gain system-wide administrative control of various functions. For example, Extended Security is a selection on the POVI Recording Manager Admin pulldown menu.

The very latest information about how to implement POVI security is available through POVI online Help. This information may be periodically updated by the application of Candle cumulative maintenance.
Protecting IDs and passwords

Although some of the examples in this chapter show user IDs and passwords embedded in REXX execs or scripts, you will probably want to avoid security exposures by storing the IDs and passwords in a protected dataset and retrieving them during execution.

Also, the POVISEND function has a CONFIDENTIAL keyword that can be used to prevent the tracing of user IDs and passwords. See the AF/OPERATOR Command Reference Manual for a description of the POVISEND function.

If you are migrating from a previous version of POVI and are using environment commands to set (\SET) and type (\type, @type, #type) passwords or other sensitive information in a REXX exec, you can use a variable named PASSWORD to prevent the tracing of the data in the CT/Engine log.
POVI Environment Commands

Introduction

Environment Commands are low-level, primitive POVI functions. They are included for customers migrating from previous versions of Candle products. These commands are not recommended for new development; instead, use the functions described in “POVI REXX Functions” on page 158.

The commands in this section are valid only in the POVI environment. When used in a REXX exec, they must be preceded by a logon to the AFGATE environment. They can also be issued interactively from the command line on the POVI Session Selection panel or (unless otherwise noted) from any input field in a VTAM application accessed through POVI.

There are two kinds of POVI environment commands:

- Support commands that invoke the POVI facilities or communicate status.
  The \RC and \SET commands that you have already used are examples of support commands.

- Script simulation commands that can be used for executing keystrokes.
  You can include script simulation commands (such as \LOC and \TYPE) in REXX execs to position the cursor and type in variables.

POVI trigger character

The POVI trigger character prefixes all POVI environment commands and must match the trigger character specified for the AF/OPERATOR POVITRGC parameter. The POVITRGC startup parameter is defined during AF/OPERATOR customization. The backslash character (\) is the default. Some Candle documentation may refer to this character as the environment command prefix character.
Summary of POVI environment commands

The POVI environment commands are summarized below. For more detailed information about these commands and about POVI REXX functions which perform similar tasks (marked “see also” below), see the AF/OPERATOR Command Reference Manual.

\K Simulates pressing a specified function key when the procedure executes. (See also, the POVISEND function.) Any of the following keys may be designated a function key though the \K command:
  - F1 through F24
  - PA1 through PA3
  - Home
  - Clear
  - Enter
  - Up
  - Down
  - Left
  - Right
  - Tab
  - Backtab
  - EraseEOF
  - EraseINP

\LOC Positions the cursor at a specified row (rr) and column (cc) when the REXX exec runs. (See also, the POVILOC function.) The syntax is:

\LOC (rrcc)

\RC When issued from the command line of the POVI Session Selection panel or from an input field of a VTAM application accessed through POVI, \RC opens the Script Selection panel, where you can begin recording a new script or access an existing script.

When issued from a REXX exec, \RC invokes a prerecorded script during execution. (See also, the POVIPBS function.) The syntax is:

\RC scriptname

\S Starts a session with a VTAM application. (See also, the POVILGON function.)

The syntax is:

\S applid
POVI Environment Commands

\SEE Displays the variables set with the `SET command. You can use this command interactively on the POVI command line to check variable values, or you can use it in a REXX exec. The syntax is:

\SEE varname
.
.
.
'anything'

The last line signals the end of the sequence. It can be any string of a maximum of 8 characters before the blank (if you include a blank). Be sure to include this line, because your exec may hang without it.

The following sequence

\SEE USERID
WTO "The POVI variable USERID='&LINE1'
'\SEE END'

writes the current value of USERID to the console.

\SET Defines a variable and allows you to assign it an initial value. (See also, the POVISET function.) \SET can be used both on the POVI command line and in a REXX exec.

\SET varname
'varvalue'

When you are initializing the variable to record a script, you can omit the quotes (unless the initial value you set is a substituted variable).

\STATUS Allows you to check the status of a VTAM application before executing a recorded script or a REXX exec. The syntax is:

\STATUS sessid
.
.
.
'anything'

where sessid is the session ID of the VTAM application. As with the \SEE command, be sure to include a throwaway line to end the sequence, or your REXX exec may hang during execution.

\TYPE Types the value of a variable at the current cursor location on the screen. The syntax is:

\TYPE varname
\@TYPE

Types the value of a variable at the specified row and column on the screen. The syntax is:

\@TYPE (rrcc)varname

where rr is the row number and cc is the column number.

\#TYPE

Types a variable value at a specified unprotected input field. To determine the relative location of the desired field, start with the Home field as field 1 and count forward by pressing the Tab key.

You may use negative as well as positive numbers. To determine the negative number of a field, start with the Home field and count backward by pressing the Backtab key. The last input field on a screen can be specified as (-1).

The syntax is:

\#TYPE (nnn)varname

WAIT

Waits for a specified number of seconds, then exits to the VTAM application and refreshes the screen. (See also, the POVIDLAY function.) The purpose of this command is to allow the application to complete screen processing and updating before more data is passed.

The syntax is:

\WAIT seconds
Communicating with OMEGAMON

Introduction

The AF/OPERATOR OMEGAMON interface lets you gather a wide range of performance data on your system and send it to other applications or automation products. You can gather data from the following Candle performance monitoring products:

- OMEGAMON for MVS
- OMEGAMON for CICS
- OMEGAMON for IMS
- OMEGAMON II for DB2

This chapter describes how to log on and off the OMEGAMON interface, as well as the two principal methods of using the OMEGAMON interface: trapping OMEGAMON exceptions and issuing OMEGAMON commands.

Chapter Contents

Why Use the OMEGAMON Interface? ............................................ 190
Logging On and Off the OMEGAMON Interface ............................... 191
Exception Traps ........................................................................ 194
Issuing OMEGAMON Commands .................................................. 195
Why Use the OMEGAMON Interface?

Exception analysis

Exception analysis is an OMEGAMON feature that monitors predefined thresholds in your system. When a threshold is exceeded, an event called an exception is created. OMEGAMON displays the exception on the OMEGAMON console.

Through AF/OPERATOR, you can write a trap that, at regular intervals, queries an OMEGAMON for an exception and then takes actions when the trap executes. For example, you could build automation routines that reduce the system workload, and when you trap an OMEGAMON for CICS PAGE exception (page-in rate too high), execute the routine when the trap executes.

Communication between OMEGAMONs

The following figure shows how the interface facilitates communication between OMEGAMONs on different systems.

AF/OPERATOR, running on system C, gathers performance data through the OMEGAMON interface from various OMEGAMONs running on systems A, B, and D, as well as C. Exception traps, set up within AF/OPERATOR, fire when specific events occur on those systems, triggering routines set up with an automation product. Two traps are illustrated in the figure: one which sends a WTO when an event is picked up by the OMEGAMONs running on systems A and B, and another which notifies a systems programmer when an event is picked up by an OMEGAMON running on systems C or D.
Logging On and Off the OMEGAMON Interface

Overview
You must log on to connect to the OMEGAMON interface and log off to disconnect. This section lists the procedures for logging on and off.

Before you begin
Before logging on to the OMEGAMON interface, make sure that you followed the installation procedures outlined in the *AF/OPERATOR Configuration and Customization Guide* to define all VTAM connections and LUs.

LOGON command
The LOGON command establishes communication between AF/OPERATOR and a particular OMEGAMON session. It also instructs AF/OPERATOR to poll OMEGAMON for exceptions at regular intervals. Here is an example of the LOGON command using OMEGAMON for MVS:

```
“LOGON OMMVS APPLID(OMVTAMA) NAME(OMMVS1)”,
“USERID(ABC99/AOPASS) PASSWORD(AOAUTH) INTERVAL(00:01:00)”,
“DATA(‘MVS=JOB01’) LTERM(AOVTM101) SSIZE(2)”
```
Logon parameter definitions

**LOGON**
Creates a VTAM link.

**OMMVS**
Indicates the link is with OMEGAMON for MVS.

LOGON commands can also link AF/OPERATOR with:

- **OMCICS**
  OMEGAMON for CICS

- **OMIMS**
  OMEGAMON for IMS

- **OMDB2**
  OMEGAMON II for DB2

Note that these are reserved names supplied by the product. For examples of the LOGON command for OMEGAMON for CICS, OMEGAMON for IMS, and OMEGAMON II for DB2, see the AF/OPERATOR Command Reference Manual.

**APPLID(OMVTAMA)**
OMVTAMA is the defined term for VTAM. It is the name by which the link is known to your data center. This is the applid of the OMEGAMON.

**NAME(OMMVS1)**
OMMVS1 is a user-defined name by which the link is known to AF/OPERATOR. When using the AFADDR command, this name allows AF/OPERATOR to access the addressed environment.

**USERID(ABC99/AOPASS)**
AF/OPERATOR logs on with this user ID, which replaces the user ID field on the OMEGAMON logon screen for this session.

**PASSWORD(AOAUTH)**
AF/OPERATOR handles OMEGAMON internal security with this password, which replaces the password field on the OMEGAMON logon screen for this session.

**INTERVAL(00:01:00)**
AF/OPERATOR will poll this link once a minute for exceptions. This parameter is required for exception traps to function.

**DATA(text)**
Logon information required by the application. For example, when logging onto OMEGAMON for MVS, this operand is required in the form:

```
DATA('MVS=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.
**Communicating with OMEGAMON 193**

**Logging On and Off the OMEGAMON Interface**

For more about the LOGON command, see the AF/OPERATOR Command Reference Manual.

**Logging off**

The LOGOFF command breaks communication between AF/OPERATOR and a particular OMEGAMON session. For example, the following command terminates the session initiated by the LOGON command described in the preceding section:

```plaintext
LOGOFF OMMVS1
```

For more about the LOGOFF command, see the AF/OPERATOR Command Reference Manual.

---

**LTERM(luid)**

A parameter that can be used to force the same virtual terminal ID to be used each time. The `luid` does not have to be in any of the $VTAPPL statements of the AOVTPOOL (so it will not be chosen by any other LOGON), but it needs to be defined to VTAM just as though it were in the pool.

**SSIZE(2/3/4/5)**

A single-digit numeric argument used to select the virtual screen size. Optional screen requests are given in the table below:

<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>

For more about the LOGON command, see the AF/OPERATOR Command Reference Manual.
Exception Traps

Overview

You create exception traps, or xtype traps, with the TRAP ADD command. These commands trap particular OMEGAMON exception messages. The text of the exception message that causes the trap to fire is placed into the AF/OPERATOR-supplied variable AOTEXT, where it can be retrieved by your REXX exec.

Further information

For more information about exception traps, review the examples shown in “OMEGAMON Exception (xtype) Traps” on page 51 and see TRAP ADD xtype in the AF/OPERATOR Command Reference Manual.
Issuing OMEGAMON Commands

Introduction
You can use the interface to send OMEGAMON commands and receive the responses in an automation product. For example, you might have trapped an exception that indicates a job is in a wait condition. You can then use OMEGAMON commands to provide more information on the exception and the job, or take actions such as canceling the job.

Sample REXX exec
The following sample exec demonstrates an important feature of the interface: the ability to issue commands directly to and get responses from an OMEGAMON. This is accomplished using the AFADDR command.

When the tenth WAIT exception occurs, the trap EXMWAIT (described in “OMEGAMON Exception (xtype) Traps” on page 51) executes the following REXX exec, MWAIT:
The first active command in MWAIT is:

**AFADDR OMMVS1**

AFADDR directs AF/OPERATOR commands to a particular OMEGAMON session. For example, AFADDR OMMVS1 directs any subsequent text that is not an AF/OPERATOR command to the session that was named OMMVS1 by the earlier LOGON command.
Further information

For more about the AFADDR command, see the AF/OPERATOR Command Reference Manual.
Communicating with NetView

Introduction

You can consolidate the network management features of NetView and the communication features of AF/OPERATOR into a single, more effective interface.

If you are currently using NetView to manage your network, you can enhance your system by adding AF/OPERATOR’s communication features without sacrificing or reproducing many customizations you have already coded into NetView.

Similarly, if you are already using AF/OPERATOR to monitor and respond to consoles on your system, you can enhance your system by adding NetView’s network management interface without sacrificing existing AF/OPERATOR traps.

AF/OPERATOR communicates with the IBM NetView™ product in several ways, with different parts of AF/OPERATOR performing distinct communication functions. The AF/OPERATOR-to-NetView communication options are:

- NetView Interface
- NetView exits and AF/OPERATOR traps
- AF/OPERATOR NVALERT and OPER commands

The NetView function or task you want to perform will determine which AF/OPERATOR communication option to use. There are different prerequisites for each.

Chapter Contents

- What You Need to Know .................................................. 200
- Using Communication Options ........................................... 201
- Controlling the NetView Interface ..................................... 203
- Starting and Stopping the AF/OPERATOR NetView Interface .... 205
- Displaying AF/OPERATOR NetView Interface Status and Matches ... 207
- Invoking AF/OPERATOR Commands from NetView ............... 209
- Retrieving Data from AF/OPERATOR or Other Candle Products ... 211
- Trapping on a NetView Alert ............................................. 218
- Trapping NetView Console Commands ............................... 220
- Trapping NetView Console VTAM Messages ......................... 222
- Generating NetView Alerts from AF/OPERATOR ................. 224
- Invoking NetView REXX Execs or Commands from AF/OPERATOR 226
What You Need to Know

Consolidating the two products requires a working knowledge of REXX and NetView, plus familiarity with AF/OPERATOR commands.

If you need to brush up on REXX, review the following IBM references:

- *TSO/E REXX/MVS Reference.*
- *TSO/E REXX/MVS User’s Guide*

If you are not already familiar with NetView, review the following IBM references:

- *NetView Resource Alerts Reference*
- *NetView Customization Guide*
- *NetView Customization Guide: Using Assembler*
- *NetView Installation and Administration Guide*
- *SNA Management Services Alert Implementation Guide.*
Using Communication Options

Introduction
You may use any or all of the AF/OPERATOR communication options described in this chapter to consolidate your NetView and AF/OPERATOR systems.

Tasks and communication options
The table below describes various tasks and the AF/OPERATOR communication options that you can use to accomplish them.

<table>
<thead>
<tr>
<th>IF you want to . . .</th>
<th>THEN use the AF/OPERATOR . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>trigger an AF/OPERATOR REXX exec or command from NetView</td>
<td>NetView Interface</td>
</tr>
<tr>
<td>pass NetView information to AF/OPERATOR</td>
<td>NetView Interface</td>
</tr>
<tr>
<td>retrieve data from AF/OPERATOR or from other Candle products and send it to NetView</td>
<td>NetView Interface</td>
</tr>
<tr>
<td>trigger a NetView REXX exec or command from AF/OPERATOR</td>
<td>OPER command</td>
</tr>
<tr>
<td>generate a NetView Alert from AF/OPERATOR</td>
<td>NVALERT command</td>
</tr>
<tr>
<td>trap NetView NCCF console commands from AF/OPERATOR</td>
<td>CMD type TRAP in conjunction with exit driver DSIEX01</td>
</tr>
<tr>
<td>trap NetView console VTAM messages from AF/OPERATOR</td>
<td>WTO type TRAP in conjunction with exit drivers DSIEX06 and DSIEX11</td>
</tr>
</tbody>
</table>

What the NetView Interface does
The AF/OPERATOR bi-directional NetView Interface allows direct communication between the address spaces of the two products using a NetView feature called the Program-to-Program Interface (PPI). NetView Version 2.0 or above is required and the PPI must be enabled.

After you establish a communication link between AF/OPERATOR and NetView, you can use the AOCMD() function in your NetView REXX execs to pass commands directly to AF/OPERATOR or to invoke AF/OPERATOR REXX execs. You can also use the NVIDIA() function in your AF/OPERATOR REXX execs to send data back to NetView.
What the AF/OPERATOR NetView exits do

Both AF/OPERATOR and NetView enable you to filter messages and then either log them or program them to trigger automated system action. By linking AF/OPERATOR and NetView, the AF/OPERATOR NetView exits enable you to control the VTAM console input from both systems from a single point of control.

VTAM messages destined for an MVS console or the system log normally pass through the Subsystem Interface (SSI). AF/OPERATOR routinely examines traffic passing through the SSI to determine whether SSI events match defined traps. When NetView is active, traffic between NetView and VTAM uses a different route, making it impossible for AF/OPERATOR to trap.

The AF/OPERATOR NetView exits overcome this problem by intercepting NetView console VTAM messages and NetView console commands (even ones that NetView suppresses from display) and passing them to AF/OPERATOR where they can be intercepted by traps you write.
Controlling the NetView Interface

AF/OPERATOR-supplied NetView Interface REXX functions

The NetView Interface REXX functions are:

**AOCMD()**
Always executed in the NetView address space (see item 1 in Figure 16 on page 203). It sends a command string from NetView to AF/OPERATOR. You can use it to invoke AF/OPERATOR commands or REXX execs (described in “Invoking AF/OPERATOR Commands from NetView” on page 209) or to initiate data retrieval from OG/MVS or other Candle products (described in “Retrieving Data from AF/OPERATOR or Other Candle Products” on page 211).

**NVIDATA()**
Always executed in the AF/OPERATOR address space (see items 2 and 3 in Figure 16 on page 203). It passes the names and values of AF/OPERATOR REXX variables to the NetView REXX exec that called AF/OPERATOR with the AOCMD() function. You can use it to retrieve data from AF/OPERATOR or from other Candle system-monitoring software products (described in “Retrieving Data from AF/OPERATOR or Other Candle Products” on page 211) and to supply data back to a running REXX program in NetView.

For a complete syntax of each function, see the *AF/OPERATOR Command Reference Manual*.

**FIGURE 16. NetView Interface Communications Scenario**

![NetView Interface Communications Diagram](image-url)
Summary of AF/OPERATOR NetView Interface commands

The AF/OPERATOR NetView Interface responds to the following AF/OPERATOR commands:

- **NVISTART**
  Starts NetView Interface communications as described in “Starting and Stopping the AF/OPERATOR NetView Interface” on page 205.

- **NVISTOP**
  Stops NetView Interface communications as described in “Starting and Stopping the AF/OPERATOR NetView Interface” on page 205.

- **NVISHOW**
  Displays NetView Interface communication status as described in “Displaying AF/OPERATOR NetView Interface Status and Matches” on page 207.

- **SHOW MATCHES NVI**
  Displays active NetView Interface matches as described in “Displaying AF/OPERATOR NetView Interface Status and Matches” on page 207.

For complete syntax descriptions of AF/OPERATOR commands, see the *AF/OPERATOR Command Reference Manual*.

AF/OPERATOR NetView Interface variables

AF/OPERATOR automatically supplies a set of system and global match variables. Those directly related to the NetView Interface feature include:

- **AONVICMD**
  The number of NetView Interface commands received since the last NVISTART command.

- **AONVICTM**
  The total number of NetView Interface commands received since AF/OPERATOR was started.

- **AONVCLST**
  The name of the NetView exec that issued the AOCMD REXX function.

- **AOSSNAME**
  Contains the 1 to 4 byte subsystem name specified using the SUBSYS startup parameter. You can use this value to direct console commands you have defined to specific AF/OPERATOR address spaces.

For a complete list of system and global match variables created by AF/OPERATOR, see the *AF/OPERATOR Command Reference Manual*. 
Starting and Stopping the AF/OPERATOR NetView Interface

Introduction
After installing and customizing the AF/OPERATOR NetView Interface feature, you must bring up AF/OPERATOR and then issue the AF/OPERATOR NVISTART command to activate the NetView Interface on each AF/OPERATOR. You can issue the NVISTART command from a console or from within a REXX exec. If NetView is not available when you issue the command, AF/OPERATOR will attempt to initiate communication at regular intervals (every 30 seconds) until either NetView becomes available or until the NVISTOP command is issued.

You can issue the NVISTART command any time AF/OPERATOR is running. You can also add the NVISTART command to the AF/OPERATOR startup procedure to automatically initiate communication on the AF/OPERATOR side. The NetView side is available as soon as you bring up NetView.

PPI control and the NVISTART command
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 reinitialize the receiver ID.

Prerequisites for the NetView Interface
Before starting the AF/OPERATOR NetView Interface you must have:
- included the AF/OPERATOR load library in the NetView startup procedure and the NetView subsystem address space procedure during AF/OPERATOR NetView Interface customization
- made the NetView module CNMNETV accessible to AF/OPERATOR
- enabled the NetView PPI
Starting the AF/OPERATOR NetView Interface

Issue an NVISTART command for the AF/OPERATOR NetView Interface you want to start. For example, from a console type:

```
O120 NVISTART
```

to start the NetView Interface on the AF/OPERATOR whose subsystem name is O120. The NVISTART command defines a NetView PPI receiver ID (O120, in this case) for use with the AOCMD() function. The SUBSYS startup parameter of the AF/OPERATOR which will communicate with NetView is used as the NetView receiver ID.

When you issue the NVISTART command from within a REXX exec, do not specify the subsystem name. Refer to the AF/OPERATOR Command Reference Manual for a complete description and syntax of the NVISTART command.

Stopping the NetView Interface

You can stop AF/OPERATOR to NetView communication at any time by issuing the NVISTOP command from a console or from within a REXX exec.

**Note:** NVISTOP’s effect on currently running REXX execs initiated by AOCMD() is unpredictable; however, the AOCMD function will return a non-zero function code indicating an error occurred.

Issue an NVISTOP command for the AF/OPERATOR NetView Interface you want to stop. For example, from a console type:

```
O120 NVISTOP
```

to stop the NetView Interface on the AF/OPERATOR whose subsystem name is O120.

Refer to the AF/OPERATOR Command Reference Manual for a complete description and syntax of the NVISTOP command.
Displaying AF/OPERATOR NetView Interface Status and Matches

Introduction

You can display the status of an AF/OPERATOR NetView Interface by issuing the AF/OPERATOR NVISHOW command from the console or from within a REXX exec. NVISHOW indicates:

- whether NetView is active, inactive, or undefined
- whether the NetView Interface receiver ID is active, inactive, or not defined
- whether the NetView Interface subtask is active, inactive, or terminating
- the number of command requests received by the NetView Interface
- the number of currently active NetView Interface matches

You can also display all active NetView Interface matches by issuing the AF/OPERATOR SHOW MATCHES NVI command from the console or from within a REXX exec.

Refer to the AF/OPERATOR Command Reference Manual for a complete description and syntax of the NVISHOW and SHOW MATCHES NVI commands.

Prerequisites

To display the NetView Interface status AF/OPERATOR must be up and active.

Procedure

From within an AF/OPERATOR REXX exec, issue an NVISHOW command as follows:

```
"NVISHOW"
```

When the NVISHOW command is issued from within a REXX exec, do not specify a subsystem name.

From a console, issue an NVISHOW command for a specific AF/OPERATOR NetView Interface. For example,

```
O190 NVISHOW
```

where O190 is the subsystem name of an AF/OPERATOR system.

The NVISHOW command’s status display appears in the AF/OPERATOR log. The following display indicates the NetView Interface has not been started.

```
!AOP3407  NETVIEW INTERFACE (INACTIVE)
!AOP3407  RECEIVER ID (O190)  STATUS (INACTIVE)  SUBTASK (INACTIVE)
!AOP3407  COUNT (0,0)  ACTIVE MATCHES (0)
```
Displaying active NetView Interface matches

Issue a SHOW MATCHES NVI command for a specific AF/OPERATOR NetView Interface. For example, from a console, Enter:

```
O190 SHOW MATCHES NVI
```

where O190 is the subsystem name of an AF/OPERATOR.

When you issue this command from an AF/OPERATOR REXX exec, do not specify the subsystem name, but remember to enclose the command in quotes as follows:

```
“SHOW MATCHES NVI”
```
Invoking AF/OPERATOR Commands from NetView

Introduction

You can use the NetView Interface to execute an AF/OPERATOR command from NetView by placing the command into the Candle-supplied REXX function AOCMD().

Prerequisites

Before using the NetView Interface to execute AF/OPERATOR commands from NetView you must have:

- included the AF/OPERATOR load library in the NetView startup procedure and in the NetView subsystem address space procedure during AF/OPERATOR NetView Interface customization
- made the NetView module CNMNETV accessible to AF/OPERATOR
- enabled the NetView PPI
- issued the NVISTART command to activate the AF/OPERATOR NetView Interface

The AOCMD() function

To send a command from NetView to AF/OPERATOR, the syntax is as follows:

```
variable=AOCMD("command string", "subsysname")
```

where

- **command string** A character string containing the command to be passed to AF/OPERATOR.
- **subsysname** The Receiver ID that was assigned (automatically) by the NVISTART command.
- **variable** Stores a function code returned by AOCMD.

The AOCMD() function process

The following process occurs when the AOCMD function is invoked:

1. NetView sends the command string to AF/OPERATOR.
2. A function code, indicating the status of the AOCMD function call, is returned to NetView by the AOCMD function.
3. A separate return code, indicating the outcome of the AF/OPERATOR command you sent, is stored in the variable AORC, which is automatically created when the AOCMD function completes successfully.
Procedure

This procedure outlines the steps to create and invoke a NetView REXX exec to invoke AF/OPERATOR commands from NetView:

1. Code the NetView REXX exec.
   A. Use the AOCMD() REXX function to invoke the AF/OPERATOR commands. For a complete description and syntax of the AOCMD() function, see the AF/OPERATOR Command Reference Manual.
   B. Include code to check the AORC variable to determine if the AF/OPERATOR commands executed successfully.
   C. Include code to handle errors that may occur.

2. Start the NetView Interface as described in “Starting and Stopping the AF/OPERATOR NetView Interface” on page 205.

3. Invoke the REXX exec you have written to execute the AF/OPERATOR commands.

Example

In this example, the function makes AF/OPERATOR add a trap that suppresses $HASP messages routed to any console. This command is sent to an AF/OPERATOR, whose subsystem name is O120, where it is executed.

/* NetView REXX exec */

x=AOCMD("TRAP ADD(NOJES) WTO("$HASP") SUPPRESS", "O120")

IF (x=0) & (aorc=0) THEN
   END
ELSE
   DO
      SAY ‘AOCMD Function code=’ x
      SAY ‘AORC=’ aorc
   END

NetView sends the TRAP ADD command to AF/OPERATOR. Then, AF/OPERATOR sends a function code back to NetView, informing you that the command was sent. Function codes and their explanations appear in the AF/OPERATOR Command Reference Manual.

In this example, the AOCMD function code is stored in variable x. When AOCMD is successful, the AORC variable is created, and a return code, indicating the status of the command sent, is stored there. For both function code and return code, a value of 0 (zero) indicates success.
Retrieving Data from AF/OPERATOR or Other Candle Products

Introduction

NetView can use the NetView Interface and Candle-supplied REXX functions AOCMD() and NVIDATA() to retrieve data from AF/OPERATOR or from Candle’s other system-monitoring software to which AF/OPERATOR has access. For example, NetView can use the interface to retrieve data from:

- OMEGAMON
- OMEGAVIEW

Prerequisites

Before using the AF/OPERATOR NetView Interface to retrieve data from AF/OPERATOR or other Candle products, you must have:

- included the AF/OPERATOR load library in the NetView startup procedure and the NetView subsystem address space procedure during AF/OPERATOR NetView Interface customization
- made the NetView module CNMNETV accessible to AF/OPERATOR
- issued the NVISTART command to activate the AF/OPERATOR NetView Interface

Process

The following sequence of events describes a typical NetView Interface data retrieval scenario:

1. An event, such as a line going down, triggers an alert to NetView, which prompts NetView to execute a REXX exec.
2. An AOCMD() function in the NetView REXX exec sends a command string to AF/OPERATOR, requesting AF/OPERATOR to execute a REXX exec.
3. AF/OPERATOR retrieves the data (from itself or from other sources).
4. An NVIDATA() function in the AF/OPERATOR exec returns variables to the originating NetView REXX exec. Both types of data retrieval (from AF/OPERATOR itself or through AF/OPERATOR, from other sources) require that the AF/OPERATOR exec use the NetView function, NVIDATA().
Procedure

Follow these steps to create and invoke NetView and AF/OPERATOR REXX execs to retrieve data:

1. Code the NetView REXX exec.
   A. Use the AOCMD() REXX function to invoke the AF/OPERATOR REXX exec. For a complete description and syntax of the AOCMD() function, see the AF/OPERATOR Command Reference Manual.
   B. Include code to check the AORC variable to determine if the AF/OPERATOR commands executed successfully.
   C. Include code to handle errors that may occur.
   D. Include code to retrieve the data returned from AF/OPERATOR.

2. Code the AF/OPERATOR REXX exec you want to use to retrieve and return data to NetView.
   A. Include code to collect data from AF/OPERATOR or from other Candle products.
   B. Use the NVIDATA() REXX function to return data to the NetView REXX exec. For a description and syntax of the NVIDATA() function, see the AF/OPERATOR Command Reference Manual.

3. If you are collecting data from other Candle products, code the REXX execs you want to use to collect and return data from these products to AF/OPERATOR.

4. Start the NetView Interface as described in “Starting and Stopping the AF/OPERATOR NetView Interface” on page 205.

5. Invoke the REXX execs you have written to retrieve data.

Example 1: Retrieving data from AF/OPERATOR

In this example, a NetView exec requests information that AF/OPERATOR can furnish itself.

1. NetView sends the request to AF/OPERATOR, asking it to invoke an AF/OPERATOR exec. An example of a calling NetView exec appears in Figure 17 on page 212.

FIGURE 17. NetView to AF/OPERATOR: Executing a NetView Exec

```/* NetView REXX exec */
x=AOCMD("EX AOMVSCMD 'D SMF'") /* command with response */
IF (x=0) & (AORC=0) THEN
   DO
      SAY 'command=' resp.1 /* resp.1 is first line */
      DO i = 2 TO resp.0 /* resp.0 is # of lines */
         SAY 'response=' resp.i
      END
   END
ELSE
   DO
      SAY 'AOCMD Function code=' x
      SAY 'AORC=' aorc
   END
.*```
2. AF/OPERATOR executes AOMVSCMD (see Figure 17 on page 212), which issues a console command on the local MVS system and returns the response. The lines of text in the response will return in the resp. stem variables. The total number of stem variables is determined by resp.0, which specifies the number of lines returned.

**Note:** This routine calls sample REXX execs (names AOE*) that are distributed with AF/OPERATOR. Look in the sample library that accompanies AF/OPERATOR for additional information.

**FIGURE 18. AOMVSCMD: Using NVIDATA() to Extract AF/OPERATOR Data**

```rexx
/* AF/OPERATOR REXX exec named AOMVSCMD */
IF ARG() = 1 THEN DO
  ARG cmd
END
ELSE DO
  cmd=""
  DO i=1 to ARG()
    cmd=cmd||" "|ARG(i)
  END
END
status='AOEAOPER'(cmd)
IF status = 0 THEN DO
  resp.0='QUEUED'()
  status='NVIDATA'("RESP.0")
  DO i=1 TO resp.0 WHILE(status = 0)
    PARSE PULL resp.i
    status='NVIDATA'("RESP."i)
  END
  IF status < 0 THEN DO
    status=-1*status
  END
END
ELSE DO
  IF status > 0 THEN status=-3
END
RETURN status
```

The AOMVSCMD exec will:

A. get the console command to be issued
B. call AOEAOPER, passing the console command and taking default values for the optional arguments
C. collect the response text in the stack into stem variables and transmit them to the NetView caller (if the call to AOEAOPER was successful)
3. The status of the AOMVSCMD exec will return in the AORC variable in the NetView exec. In this example, AORC represents the status of the console command request. The integer values are defined below:

- **0** Indicates a successful operation.
- **-1** Indicates that the console command was not specified.
- **-2** Indicates that the maximum number of retries to access a subsystem console was exceeded.
- **-3** Indicates that the call to the AOEAOPER function failed.
- **>0** Indicates that an NVIDATA() function call failed. The value is returned by the NVIDATA() function.
Example 2: Retrieving data from OMEGAMON for MVS

In this example, a NetView REXX exec triggers the execution of an AF/OPERATOR REXX exec which logs onto OMEGAMON for MVS and issues an OMEGAMON command. The OMEGAMON data is returned to the calling NetView REXX exec in the form of line variables. The resulting NetView display resembles a “classic” OMEGAMON screen.

1. The calling REXX exec, named MONENQ, runs in the NetView address space.

```rexx
/* REXX ************************************************************************* */
/*                                                                            */
/* PROGRAM NAME: MONENQ                                                      */
/* ************************************************************************** */
/* TRACE I */
'MSG LOG ENTERING MONENQ CLIST' DATE() TIME(E)
ADDRESS NETVASIS
PARSE ARG DOMAIN_NAME
CMD = 'X=AOCMD("EX ISSCENQ" "DOMAIN_NAME")'
INTERPRET CMD
SAY 'OMEGAMON/MVS 'DOMAIN_NAME
IF (X=0) & (AORC=0) THEN
   DO I = 1 TO RESP.0
      RESPONSE.I = OMEGAMON_LINE.I
      SAY RESPONSE.I
   END
ELSE
   DO
      SAY 'AOCMD Function Code=' x
      SAY 'AORC=' aorc
   END
'MSG LOG EXITING MONENQ CLIST' DATE() TIME(E)
EXIT
```

2. The called REXX exec, named ISSCENQ, runs in the AF/OPERATOR address space.
/* REXX ***************************************************************************/ /* */ /* PROGRAM NAME: ISSCENQ */ /* ***************************************************************************/ /* TRACE R */ SAY 'ENTERING NETVIEW INITIATED PROBE - ISSCENQ' strt_TIME = DATE('S')||'','','||TIME('L') SIGNAL ON ERROR NAME ERROR PARSE ARG ENVIRONMENT_NAME /* ***************************************************************************/ /* PROGRAM VARIABLES */ /* ***************************************************************************/ LINE. = '' NVLINE. = '' /* ***************************************************************************/ /* ESTABLISH ADDRESSABILITY TO OMEGAMON */ /* ***************************************************************************/ "ADDRESS" ENVIRONMENT_NAME IF RC <> 0 THEN DO WTO 'UNABLE TO ACCESS OMEGAMON' MSGID(@ISSCENQ)" SIGNAL ERROR END /* ***************************************************************************/ /* ISSUE OMEGAMON COMMANDS FOR ENQUEUES */ /* ***************************************************************************/ "XQCB" "ADDRESS" /* RESET ADDRESS */ /* ***************************************************************************/ /* LOAD SYSTEM VARIABLES & CREATE LOG ENTRIES */ /* ***************************************************************************/ /* HOW MANY LINES CAME BACK? */ LINE# = 0 RET = 'GLBVGET' ('LINE#')
IF LINE# = 0 | LINE# = '' THEN
DO
   "WTO 'COMMAND FAILED IN OMVTAMA' MSGID(@ISSCENQ)"
   SIGNAL ERROR
END

/* ------------------------ PROCESS THE RETURNED LINES */

RESP.0 = LINE#
STATUS = 'NVIDATA'("RESP.0")
DO I = 1 TO LINE# /* BEGIN DO */
   RET=GLBVGET('LINE'||I)
   OMEGAMON_LINE.I=VALUE('LINE'||I)
   STATUS = 'NVIDATA'("OMEGAMON_LINE."I)
END /* END DO */

/* ------------ UPDATE TIME-STAMP SYS VARIABLE */
T_STAMP = DATE('S')'||_'||TIME('S')

/* ****************************************************** */
/* DONE */
/* ****************************************************** */
end_TIME = DATE('S')'||_,||TIME('L')
say 'ISSCENQ starts at --'strt_TIME' and ended at --'end_time'

SAY 'EXITING NETVIEW INITIATED PROBE - ISSCENQ'
EXIT

/* ****************************************************** */
/* TROUBLE PROCESSING */
/* ****************************************************** */
ERROR:
WTO_MSG = '*** NETVIEW CALLED PROBE FAILED'
"WTO '"WTO_MSG"' MSGID(@ISSCENQ)"
EXIT /* RETURN -1 */
Trapping on a NetView Alert

Introduction
AF/OPERATOR does not trap NetView alerts. However, NetView can be customized internally, using SRF filters, to trap on its own alerts.

What you need to know
You will need to refer to NetView documentation for technical information on your version of NetView.

Configuring NetView to trap NetView hardware alerts
Hardware alerts (for example, the message generated when a Network Controller fails) come into the NetView NPDA, NetView’s hardware monitor component. To be trapped by NetView automation, these alert messages must be re-routed to the NetView NCCF, NetView’s automation component. NetView allows you to re-route messages using SRF filters.

For example, executing the following commands, from the Alerts - Static display in the NPDA component,

```
SRF OPER PASS DEFAULT
SRF ESREC PASS DEFAULT
SRF AREC PASS DEFAULT
```

causes all alerts received by the hardware monitor to be passed to the MVS console in the form of BNJxxxx messages. You will probably want to be much more selective in what messages you route to the MVS console at your site. Refer to your NetView documentation for keywords on the SRF command that you can use to filter out messages you do not want forwarded to the NCCF component.

The NCCF component can then be set up to monitor for alerts. The NetView automation programmer can implement an automation routine in the NCCF component to scan a command list for specific messages. A message-driven command list would contain an IF-THEN automation statement for each alert you want to trap. For example:

```
IF MSGID = 'BNJ030I' & TOKEN(12) = 'S10A5411'
THEN EXEC(CMD('AS400RD '));
```

When an alert message matches one in the command list, the NetView automation executes a REXX exec which issues a command to AF/OPERATOR (using the NetView Interface and the AOCMD function).
Possible responses to NetView alerts

There are many possible actions you might want AF/OPERATOR to take in response to a NetView alert. For example, if a Network Controller fails and triggers a NetView alert, you can forward this information through AF/OPERATOR to other Candle products. If your site uses OMEGAVIEW (Candle’s status display product), and your AF/OPERATOR to OMEGAVIEW interface is installed and configured, you can post the status of the failed Network Controller to an OMEGAVIEW status item. Using this method, one site, or an entire enterprise, can monitor both host and network information on the same OMEGAVIEW status panel. If you use AF/REMOTE, you can initiate a beeper page so a technician on call can be immediately notified of a problem.
Introduction

The AF/OPERATOR NetView exit driver for console commands, DSIEX01, routes commands from the NetView console through AF/OPERATOR. This enables command traps in AF/OPERATOR to trap NetView commands issued in NCCF (operator input on the NetView console). See the IBM NetView Customization Guide: Using Assembler for information on the exit point Candle uses.

AF/OPERATOR uses the AOSIM facility to intercept commands entered by the NetView console operator, before NetView executes or logs them. You can trap a NetView console command at this point by using the AF/OPERATOR TRAP command (CMD type). After AF/OPERATOR returns control to NetView, the AOSIM results area indicates whether AF/OPERATOR found a matching trap and whether the trap requested AF/OPERATOR to suppress the command. For information about the AOSIM facility, see “Defining WTO and Command Events with AOSIM” on page 269.

Although the only immediate action you can specify on a trap for NetView console commands is suppression, you can specify extended actions such as invoking an AF/OPERATOR REXX exec. This gives you the ability to implement automation routines in AF/OPERATOR from NetView.

Note: Invoking a REXX exec in AF/OPERATOR using the NetView Interface is a better, more direct method of communicating with AF/OPERATOR but requires NetView Version 2 or higher.

Prerequisites

Before using AF/OPERATOR to trap a NetView console command you must have:

- Installed Candle’s DSIEX01 NetView exit driver during AF/OPERATOR customization. DSIEX01 is an IBM-supplied exit point in the NetView product.
- Coded an AF/OPERATOR trap for the command.
Trapping NetView Console Commands

Procedure

Follow these steps to create and invoke an AF/OPERATOR REXX exec to trap commands entered from the NetView console:

1. Code the AF/OPERATOR REXX exec.
   A. Use the TRAP CMD command to trap commands. For information on coding traps, see “Trapping System Events” on page 35. For a description and syntax of the TRAP command, see the AF/OPERATOR Command Reference Manual.
   B. Include the SUPPRESS immediate action option to suppress commands. Commands typed on the NetView console will be suppressed in AF/OPERATOR and on the NetView console.
   C. Include code to take any extended action you may need based on the command that was typed on the NetView console.

2. Invoke the REXX exec you have written to trap the NetView console commands.

Example

In a combined NetView and AF/OPERATOR system, you want to prevent commands from being issued from a NetView NCCF console that could be considered security exposures to the rest of the system. The following AF/OPERATOR trap suppresses all commands that begin with the pattern ZNET, thus preventing operators at NCCF consoles from successfully issuing commands that can take down VTAM.

```
/***************************************************************************/
* NETVIEW CONSOLE COMMAND TRAPS                                       *
/***************************************************************************/

"TRAP ADD(ZNETNOT)",
"CMD('Z NET,*)",
"JOBNAME('NETVG')",
"ENABLE",
"SUPPRESS",
"LOG"

The JOBNAME of the NetView system on which you want to prevent the command from being issued is specified. Without the JOBNAME keyword, the command will be suppressed on all MVS consoles.
Trapping NetView Console VTAM Messages

Introduction

NetView uses the AF/OPERATOR NetView exit drivers DSIEX06 and DSIEX11 to pass VTAM messages to AF/OPERATOR with the Primary Program Operator application program (PPO). This occurs after retransmitting the messages to the master console but before logging them. (See the IBM NetView Customization Guide: Using Assembler for information on the exit points Candle uses.) You can use the AF/OPERATOR TRAP command (WTO type) to trap the VTAM message at this point as you would any other console message.

NetView uses this process rather than the AOSIM facility so that it can handle multi-line VTAM messages. As a result, the SUPPRESS option of any relevant trap will not suppress the message at the NetView console, even though it does suppress the message at the master console. You must continue to use NetView facilities to suppress VTAM messages at the NetView console.

Prerequisites

Before using AF/OPERATOR to trap NetView VTAM messages you must have:

- Installed Candle’s DSIEX06 and DSIEX11 NetView exit drivers during AF/OPERATOR customization. DSIEX06 and DSIEX11 are IBM-supplied exit points in the NetView product.
- Coded an AF/OPERATOR trap for the WTO.

Procedure

Follow these steps to create and invoke an AF/OPERATOR REXX exec to trap NetView VTAM messages:

1. Code the AF/OPERATOR REXX exec.
   A. Use the TRAP WTO command. For information on coding traps, see “Trapping System Events” on page 35. For a description and syntax of the TRAP command, see the AF/OPERATOR Command Reference Manual.
   B. Include the SUPPRESS immediate action option to suppress the VTAM messages in AF/OPERATOR.
      Note: AF/OPERATOR will not suppress VTAM messages on the NetView console.
   C. Include code to take any extended action you may need based on the NetView VTAM messages.

2. Invoke the REXX exec you have written to trap the NetView VTAM messages.
Example

Traps like the following are typically set during AF/OPERATOR initialization.

```c
/************************************
 NETVIEW MESSAGES
 ************************************/

 "TRAP ADD(TSBNJ001)",
 "WTO('BNJ*')",
 "ENABLE",
 "SUPPRESS",
 "NOLOG"

 "TRAP ADD(TSCNM001)",
 "WTO('CNM*')",
 "ENABLE",
 "SUPPRESS",
 "NOLOG"
```
Introduction

A NetView alert is a record, created by a system or network resource, of an event requiring immediate attention. NetView alerts notify the NetView hardware monitor about the existence and nature of network problems. The hardware monitor displays data from the alert on information panels, informing the operator of status and suggesting possible corrective action. Alerts can also trigger automated system actions.

You can generate NetView alerts by using the AF/OPERATOR NVALERT command within an AF/OPERATOR REXX exec or by specifying the command as the action of a trap. For demonstration purposes, the NVALERT command can also be issued from an MVS console. Alerts generated by using the AF/OPERATOR NVALERT command are sent through VTAM and the CNM routing table, unless the RECEIVER keyword is specified. Specifying RECEIVER causes the alert to be sent to the specified receiver using the NetView Program-to-Program Interface (PPI).

By issuing a generic alert, AF/OPERATOR can notify NetView of the existence and nature of a system network problem. You can then use NetView’s problem determination and diagnostic tools to monitor, identify, and fix the problems.

The NVALERT command requires you to specify an alert type, ID number, and NetView code points (values that index text strings at the alert receiver which display information such as description, probable cause, and required action). You may also specify a hierarchy of system resources pertinent to the alert data, as well as an arbitrary text string for NetView to use should an alert be triggered.

For more information on the NVALERT command, see the AF/OPERATOR Command Reference Manual.

Prerequisites

Before using the AF/OPERATOR NVALERT command to generate NetView alerts you must have:

- defined a VTAM application LU to your site’s local network during NetView Customization
- varied the NetView node active to enable AF/OPERATOR to process the NVALERT command

To send alerts via the NetView PPI, the NetView alert receiver task CNMCALRT must be defined and active.
Generating NetView Alerts from AF/OPERATOR

Procedure

Follow these steps to create and invoke an AF/OPERATOR REXX exec to issue a NetView alert:

1. Code the AF/OPERATOR REXX exec.
   Use the NVALERT command to generate the alert. For a description and syntax of the NVALERT command, see the AF/OPERATOR Command Reference Manual.

   To be effective, the alerts you generate with the NVALERT command must provide NetView with meaningful data. You must identify the various code points defining problems and potential solutions to NetView beforehand, using the Hardware Monitor Code Point Tables. For information on NetView alert table definitions, refer to the IBM publication NetView Resource Alerts Reference.

2. VARY the NetView node active to enable AF/OPERATOR to process the NVALERT command.

3. Invoke the REXX exec you have written to generate the NetView alert.

Example

The following trap command, imbedded in a REXX exec, causes AF/OPERATOR to forward an alert to NetView when a CICS recursive abend occurs.

```
"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")")"
```
Invoking NetView REXX Execs or Commands from AF/OPERATOR

Introduction
You can use the AF/OPERATOR OPER command to issue NetView commands or invoke NetView REXX execs. The OPER command sends the instruction over the SSI.

The OPER command
The OPER command uses the following format:

```
OPER ‘% command’ CONSOLE(id)
```

The text enclosed in quotes must begin with a NetView identifying character, in this case, % (percent sign). NetView must be preconfigured to recognize the identifying character. The character is followed by the command you are sending to NetView and any other parameters required by NetView. The command can be `EX execname` where `execname` is the name of a NetView REXX exec. The `id` is the console number defined to NetView with the AUTOTASK command statement during customization.

Prerequisites
Before using the AF/OPERATOR OPER command to issue NetView commands or invoke NetView REXX execs you must have:

- Enabled console communication with NetView as described in the `AF/OPERATOR Configuration and Customization Guide` or the Customizer Help. This step defines the NetView identifying character and console ID.
- AF/OPERATOR and NetView must be up and active.

Procedure
Follow these steps to create and invoke an AF/OPERATOR REXX exec which uses the OPER command to send an instruction from AF/OPERATOR to NetView over the SSI:

1. Code the AF/OPERATOR REXX exec.
   Use the OPER command to send the instruction to NetView. For a description and syntax of the OPER command, see the `AF/OPERATOR Command Reference Manual`.
2. Invoke the REXX exec you have written to send the instruction to NetView.
Introduction

The OMEGAVIEW® Interface provides bi-directional communication between AF/OPERATOR and one or more OMEGAVIEW status databases. The OMEGAVIEW status 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. The OMEGAVIEW status database is referred to hereafter as “OMEGAVIEW”. This interface allows AF/OPERATOR to create status items in OMEGAVIEW and receive status indication information from OMEGAVIEW. AF/OPERATOR monitors these status items and can use automation scenarios to update them.

The interface provides several new AF/OPERATOR REXX functions that allow AF/OPERATOR and OMEGAVIEW to communicate in the following ways:

- AF/OPERATOR can create and update OMEGAVIEW status items
- AF/OPERATOR can inquire about the state of OMEGAVIEW status items
- OMEGAVIEW can send status change information to AF/OPERATOR by an internal WTO (through the AF/OPERATOR AOSIM Facility)

Chapter Contents

What You Need to Know .................................................. 229
AF/OPERATOR REXX Functions ........................................ 230
Trapping on Status Item Changes ............................... 231
Trapping on Problem Management Alerts ...................... 232
REXX Line Variables ....................................................... 234
OMEGAVIEW Return Codes ............................................ 235
AF/OPERATOR Return Codes ......................................... 236
Establishing a Session ..................................................... 237
Creating Status Items .................................................. 238
Dynamic Rollup Status Items ........................................ 240
Opening a Status Item .................................................. 242
Updating Status Item Information ............................. 243
Assigning a Status Value ............................................... 245
Returning OMEGAVIEW Column Information ........................................... 248
Automated Session Control ..................................................................... 249
What You Need to Know

Using the OMEGAVIEW Interface requires a working knowledge of REXX, AF/OPERATOR, and OMEGAVIEW. If you are not already familiar with OMEGAVIEW, review the following references:

- *OMEGAVIEW User’s Guide*
- *OMEGAVIEW Customization Guide*

If you are not familiar with AF/OPERATOR, review the materials in the AF/OPERATOR documentation set.

For more information about REXX, see the IBM® REXX library.
AF/OPERATOR REXX Functions

Introduction

Information about the following REXX functions can be found in the AF/OPERATOR Command Reference Manual in the chapter entitled “Candle-Supplied REXX Functions.” These functions can be invoked only from AF/OPERATOR REXX execs, and are the basis for communication between AF/OPERATOR and OMEGAVIEW.

Summary of OMEGAVIEW functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STCREATE()</td>
<td>Create OMEGAVIEW status items</td>
</tr>
<tr>
<td>STOPEN()</td>
<td>Create an access path to an OMEGAVIEW status item</td>
</tr>
<tr>
<td>STGET()</td>
<td>Retrieve OMEGAVIEW status item data from items previously opened with STOPEN()</td>
</tr>
<tr>
<td>STSIGNAL()</td>
<td>Update the status color of a status item by changing its value</td>
</tr>
<tr>
<td>STUPDATE()</td>
<td>Modify OMEGAVIEW status item field information, such as action and</td>
</tr>
<tr>
<td></td>
<td>problem management fields</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>STCLOSE()</td>
<td>Delete an access path to an OMEGAVIEW status item</td>
</tr>
<tr>
<td>STDSTRY()</td>
<td>Destroy OMEGAVIEW status items</td>
</tr>
</tbody>
</table>
Trapping on Status Item Changes

Introduction
You can trap status item changes and use the OMEGAVIEW interface functions to update status items when problems are resolved. When a status item changes, OMEGAVIEW sends an internal WTO that is transmitted by the AF/OPERATOR AOSIM facility.

Note: If you want to trap on changes to a specific status item, be sure that you or your OMEGAVIEW administrator have enabled that status item for generating alerts. See the OMEGAVIEW documentation for details.

WTO format
The actual content of the internal WTO issued using AOSIM varies, but its format is as follows:

OMEGAVIEW status_name status_description LEVEL: status_level

Possible status levels are:
- CRITICAL
- WARNING
- OK
- NONE

Example trap definition
This trap fires when an OMEGAVIEW status item changes:

```
"TRAP ADD(trapname)",
"WTO('OMEGAVIEW')",
"USER(KSD)",
"ENABLE",
"LOG",
"ACTION('EX execname')"
```
Trapping on Problem Management Alerts

Introduction

You can trap changes to problem management data and use OMEGAVIEW interface functions to trigger automation scripts when problem management information is added, updated, or deleted. Data emitted with the internal WTO includes enough information about the alert to allow subsequent OMEGAVIEW interface functions to retrieve and manipulate status item data and problem management data in OMEGAVIEW.

WTO Format

Each problem management event is emitted in the following format:

```
KMVPBMGT X statusitemname username starttod endtod actiondescrip
```

where:

- **X**: 1 byte indicator; A to indicate a new problem management entry, U for update, and D for delete.
- **statusitemname**: name of the status item for which problem management data has been entered/modified. The name is the standard `session.itemname` format where the session name is a maximum of 18 bytes and the item name is a maximum of 20 bytes. This value will range between 3 to 39 bytes.
- **userid**: 1 to 8 byte ID of the OMEGAVIEW user issuing the problem management entry/update.
- **starttod**: Start date and time in the format CYYMDDHHMMSS where C represents the number of centuries from 1900, YY represents the year of the century (0-99), MM represents the month of the year (1-12), DD represents the day of the month (01-31), HH represents the hour of the day (00-23), MM represents the minute of the hour (00-59), and SS represents the second of the minute (00-59).
- **endtod**: End date and time in the format CYYMDDHHMMSS format where C represents the number of centuries from 1900, YY represents the year of the century (00-99), MM represents the month of the year (01-12), DD represents the day of the month (01-31), HH represents the hour of the day (00-23), MM represents the minute of the hour (00-59), and SS represents the second of the minute (00-59).
- **actiondescrip**: 0 to 20 byte value for the updated action description field (SDMAGENT).

A TRAP ADD command may be used to detect all problem management events emitted by OMEGAVIEW. For example,

```
TRAP ADD(KOGMVPMO) WTO(‘KMVPBMGT *‘) ENABLE ACTION (‘EX KOGMVPM1‘)
```

The **AOTEXT** match variable can be parsed by the action routine specified by the ACT() on the TRAP ADD command which detected the internal WTO. For example,

```
rc=GLBVGET(“AOTEXT”)”
```
PARSE VAR aotext "KMVPBMGT" action itemname,
   "U="userid,
   "S="stod,
   "E="etod,
   "A="actdescp

The status item name parsed from the WTO text may be used to query data and manipulate
the state of the status item. For example,

/* Fetch the problem management comments and problem description */
rc=STOPEN(mvhd1, itemname, "INPUT")
rc=STGET(mvhd1, itemname)
/* Display information about status item */
DO i=1 to SDMLINE.0
   SAY sdmline.i="VALUE(sdmline.i)
END
/* Reset the state of the status item after 60 seconds */
green=10
IF action="A" & sdmvalue > green THEN DO
   ADDRESS AFHOST "WAIT SECONDS(60)"
   transtxt="Reset by OG/MVS=mvhd1"
   sdmadesc=""
   sdmagent=""
   sdmcomnt=""
   sdmsxfdt=""
   sdmxfxt=""
   rc=STSIGNAL(mvhd1, itemname, green, "transtxt")
   rc=STUPDATE(mvhd1, itemname, "sdmadesc", "sdmagent", "sdmcomnt", "sdmsxfdt", "sdmxfxt")
END
rc=STCLOSE(mvhd1, itemname, "NOCLEAR")

For further examples of trapping problem management alerts and manipulating the status
items, see the sample library members KOGMVPMx in the hilev.rte.TKANSAM dataset
delivered with OG340 and AO340.
REXX Line Variables

Overview

The OMEGAVIEW Interface returns REXX line variables after functions have been executed on the AF/OPERATOR side of the interface. The REXX line variables contain three types of data:

- total number of output lines returned by an SDM function call
- internal application ID
- column names from the OMEGAVIEW database

Process

After a function that returns SDMLINE variables has been executed:

- SDMLINE.0 contains the total number of output lines returned from the last SDM function call. If SDMLINE.0 = 0, then no SDMLINE.n variables were created from that function call.

  **Note:** Even if the last SDM function call returned no output lines, SDMLINE.n variables may still exist from an SDM function call earlier in the match.

- SDMLINE.1 returns an internal application ID. This value is used for debugging purposes and should be ignored by users of the ST functions.

- SDMLINE.2 through SDMLINE.n return the OMEGAVIEW table column names.

See Status Column Definitions in the *AF/OPERATOR Command Reference Manual* for the most current column name/type, field length, and description.
OMEGAVIEW Return Codes

Information about the return codes that OMEGAVIEW may send to AF/OPERATOR through the interface after the AF/OPERATOR REXX exec has executed can be found in the AF/OPERATOR Command Reference Manual in the chapter entitled “Candle-Supplied REXX Functions.” See the OMEGAVIEW documentation for a complete listing of OMEGAVIEW return codes.
AF/OPERATOR Return Codes

Information about the return codes that AF/OPERATOR may return after REXX exec execution using the OMEGAVIEW Interface functions can be found in the AF/OPERATOR Command Reference Manual in the chapter entitled “Candle-Supplied REXX Functions.”
Establishing a Session

Introduction
Before you can use the interface, you must establish a session between AF/OPERATOR and OMEGAVIEW using the LOGON command. For a complete description of the LOGON command see the AF/OPERATOR Command Reference Manual.

Procedure
Issue the LOGON command from the AF/OPERATOR side of the interface:

```
LOGON omview applid(xxxxxSDM) name(OMV1)
```

where xxxxx is the VTAM applid prefix chosen when you installed OMEGAVIEW. Check the applid defined in `hilev.RLVPARM(KSDINSRV)` to obtain the applid of the OMEGAVIEW Interface.

This command may be issued from the console or from within a REXX exec.

- If you are issuing commands to the interface from more than one AF/OPERATOR, you must issue a logon command from each and indicate the OMEGAVIEW you want to contact.
- If you are issuing commands from one AF/OPERATOR to multiple OMEGAVIEWs, you must issue a separate logon for each, indicating the OMEGAVIEW you want to contact.
- If you use the LOGOFF statement, the colors of all status lights will revert back to turquoise. The lights will also revert back to turquoise if the OMEGAVIEW session is terminated by a VTAM error recovery. The color of a status light may be extended temporarily by using the STSSTO and STSSTOD functions.
Creating Status Items

Introduction

Before you can access new status items, you must define them. This section explains how to define the status rule and how to create a status item with the null or threshold rule.

**Important**

Once you have defined a status item, the item remains defined over an OMEGAVIEW restart. However, since the restart terminates the OMEGAVIEW link to AF/OPERATOR, AF/OPERATOR access to the status item terminates as well. Candle recommends that you set up AF/OPERATOR to detect when OMEGAVIEW comes down and restarts. When AF/OPERATOR detects an OMEGAVIEW restart, have it terminate the current logon environment to OMEGAVIEW, re-establish the link to OMEGAVIEW, and issue the necessary STOPEN() calls to regain access to the status items.

Naming status items

Candle recommends that you establish a naming convention for OMEGAVIEW status items prior to using the interface. Consult the OMEGAVIEW User’s Guide for more information on the conventions OMEGAVIEW uses.

Status rules

The STCREATE() function defines the status rule, which determines the values at which a change in status will occur. STCREATE() performs the same function as the OMEGAVIEW Add Status Definition panel. 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.

There are three types of status rules:

- **Null**
  Default. Status values are pre-determined by OMEGAVIEW.

- **Threshold**
  Defines at what value status levels should change.

- **Roll-up**
  Derived by applying a rule to a set of subordinate status items.

For the syntax of the STCREATE function, see the AF/OPERATOR Command Reference Manual.
Creating Status Items

Using a null rule

Candle recommends using the null status rule. If you prefer to use the threshold rule, see “Using a threshold rule” on page 239. A blank value in the rule parameter within the STCREATE() function signals a request for the null rule. The null rule reacts to input values as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Status Keyword</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NONE</td>
<td>TURQUOISE</td>
</tr>
<tr>
<td>10</td>
<td>OK</td>
<td>GREEN</td>
</tr>
<tr>
<td>20</td>
<td>WARNING</td>
<td>YELLOW</td>
</tr>
<tr>
<td>30</td>
<td>CRITICAL</td>
<td>RED</td>
</tr>
</tbody>
</table>

The following exec illustrates the creation of status item TESTITEM with the default status rule.

```rexx
/* REXX             */
/* OMV1 is the LOGON session’s name */
RET=STCREATE(OMV1,‘OMV1.TESTITEM’,‘this is a test item’,‘’)
```

Using a threshold rule

The threshold rule lets you define the values at which status levels should change. This rule has one or more clauses, each of which has three parts:

- status keyword
- comparison operator (GT, GE, EQ, LE, LT, NE)
- value

In the following example, the status item SYSA.IMS is green when the CPU usage is between 0–79.999%. When the CPU usage is between 80–94.999%, the status item is yellow. When usage is at or beyond 95%, the item is red.

```rexx
RET=STCREATE(name,‘SYSA.IMS’,‘SYSTEM A PRODUCTION IMS’,
‘CRITICAL,GE,95000,WARNING,GE,80000,OK,GE,0’)
```

This rule multiplies the status value by 1000 to represent floating point values (integers containing a decimal point) as integers.
Dynamic Rollup Status Items

Introduction

A dynamic rollup status item is an item that represents one of the three basic status item types:

- a list of status items
- a pattern that matches the value of the SDMSTNAM table column
- a combination of both

Rule patterns

When you specify a pattern in the rule for a dynamic rollup status item, all open status items that match the pattern are evaluated. The value of the most critical status item becomes the value of the dynamic rollup status item. For example, if one status item has a warning value and another has a critical value, the value of the dynamic rollup status item becomes critical.

This “worst of” processing takes the following precedence from top to bottom:

- Critical
- Warning
- OK
- None

Defining a pattern

A pattern for use as a status item can consist of a combination of characters that make up that status name and wildcards. The only wildcard implemented is the asterisk “*”. The only forms that the pattern may have are:

- abc.xyz An explicit status item name.
- abc.x* All status item names that begin with “abc.x”.
- abc.* All status item names that begin with “abc”.
- a*.xyz All status item names that begin with “a” and anything up to “.” followed by “xyz”.
- a*.x* All status item names that begin with “a” and anything up to “.” followed by “x” and anything that follows.
- a*. All status item names that begin with “a” and anything up to “.” followed by anything.
- .*xyz All status item names followed by “.xyz”.

The wildcard “*” matches zero or more characters.
**Example**

The following example defines a dynamic rollup status item that is set to measure the worst of SYSA TSO response time, the CIMS overall CICS response time, and all SDM status item names that begin with the letter W and anything up to “.” followed by anything.

```
ITEM='OMVTEST.ROLLUP3'
RULE='WORSTOF(SYSA.RST_STAT,CIMS.TIME_STAT,||,
    ITEMSWHERE(SDMSTNAM,EQ,"W*."))'
RET=STCREATE(COMVW1,ITEM,'A TEST ROLLUP',RULE)
```
Opening a Status Item

Overview

STOPEN() opens an already defined status item. An item may be opened only once per session; it remains open until it is closed or the task that opened it comes down. 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.

Column names

After you execute an AF/OPERATOR REXX exec that contains the STOPEN() function, OMEGAVIEW returns a list of currently accessible column variable names. These provide diagnostic data from OMEGAVIEW, such as problem description, operator who assisted, and time between onset and resolution.

Example

To list the column variable names currently available, use SDMLINE.0 in your exec, as shown in the following example:

```rexx
/* REXX */
AO='applhndl'
ITEM='item.name'
RET = STOPEN(AO,ITEM,INPUT)
IF RET <> 0 THEN
  DO
    SIGNAL TROUBLE
  END
  DO I= 1 to SDMLINE.0
    SAY SDMLINE.I
  END

Refer to “REXX Line Variables” on page 234 for an explanation of the REXX variable SDMLINE.
Updating Status Item Information

Overview

The STUPDATE() function updates status column information by assigning data to a REXX variable whose name matches the target column name. The variable is then specified in the STUPDATE() function call as a literal value. The STUPDATE() function will not modify the SDMLINE stem variables.

Updating data columns

To update data columns, refer to the column by name. The REXX variable name should be the same as the OMEGAVIEW Status Column Name. The values of most of the columns displayed can be modified using a subsequent STUPDATE() function. The values of all columns displayed can be retrieved using a subsequent STGET() function.

Candle suggests that you explicitly set the value of columns to ensure the validity of their contents (see the following example). The interface creates a REXX variable name synonymous to the OMEGAVIEW status item column name. This REXX variable is set to the value of the data to be used to update the OMEGAVIEW column data.

Example

The following exec assigns null values to the fields indicated. Note that the value must be enclosed in single quotes and separated by commas. The double commas (,,) used at the end of some lines indicate a delimiter followed by a continuation character. This information is sent to OMEGAVIEW to update the respective column.

/* REXX */

SDMCOMNT=
SDMAGENT=
SDMADESC=
SDMSFXDT=
SDMXFXDT=
SDMXA1=
SDMXA2=
SDMXA3=
SDMXA4=
SDMXA5=
SDMXA6=
SDMXA7=
SDMXA8=
SDMTEXT=

Communicating with OMEGAVIEW
Updating Status Item Information

RET=STUPDATE(OMV1,ITEM,'SDMADESC','SDMAGENT',
   'SDMCOMNT','SDMSFXDT','SDMXFXDT',
   'SDMXA1','SDMXA2','SDMXA3','SDMXA4','SDMXA5','SDMXA6',
   'SDMXA7','SDMXA8','SDMTEXT')

SAY 'UPDATE = ' ITEM 'RC=' RET
Assigning a Status Value

Overview

The STSIGNAL() function assigns the status value to a status item by changing its color. If the
STSIGNAL() 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.

STSIGNAL 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 STSIGNAL in this case will cause a return code 21, operation not
supported.

Note: Null values are not valid for transient text when specified for STSIGNAL.

Examples

The following exec updates an item’s status value with the null rule (status value of 20) and
changes its color to yellow.

RET=STSIGNAL(OMV1,ITEM,20)
SAY ‘SIGNAL = ‘ ITEM ‘RC=’ RET

The following example has two execs, MVDEMO and MVUPDATE. MVDEMO (Figure 20 on
page 246) opens the status item PAYROLL and sets it to red in OMEGAVIEW.

Note: Before you can invoke this routine, you must be logged onto the OMEGAVIEW
session OMVW9.
Assigning a Status Value

FIGURE 20. MVDEMO Exec Example

MVUPDATE (Figure 21 on page 247) is a general REXX routine that either changes the item’s status (color) or the associated action text.
Assigning a Status Value

FIGURE 21. MVUPDATE Exec Example

```rexx
/* REXX                                                           */
/* STATUS_ITEM MUST BE AN OPEN STATUS ITEM                         */
/* STATUS_VALUE MUST CONTAIN: A COLOR OR ACTION TEXT              */
/* STATUS_FLAG MUST CONTAIN: C (COLOR) OR T (TEXT)                */
/*                                                              */
/TRACE R

PARSE ARG STATUS_ITEM  STATUS_VALUE  STATUS_FLAG       /* INPUT       */
/* ------ UPDATE THE FIELD                                         */
SELECT
  WHEN STATUS_FLAG = 'C' THEN /* SET THE STATUS CODE */
    DO
      SELECT
        WHEN STATUS_VALUE = 'GREEN' THEN STATUS_CD = 10
        WHEN STATUS_VALUE = 'YELLOW' THEN STATUS_CD = 20
        WHEN STATUS_VALUE = 'RED' THEN STATUS_CD = 30
        OTHERWISE STATUS_CD = 0 /* DEFAULT TO TURQUOISE */
      END /* SELECT */
      RET = STSIGNAL('OMVW9',STATUS_ITEM,STATUS_CD,"")
      IF RET <> 0 THEN SIGNAL TROUBLE
    END /* END SELECT */
  WHEN STATUS_FLAG = 'T' THEN, /* SET THE ACTION TEXT */
    DO
      SDMAGENT=STATUS_VALUE
      RET = STUPDATE(OMVW9,STATUS_ITEM,'SDMAGENT'
      IF RET <> 0 THEN SIGNAL TROUBLE /* NULL PARAMETER AT END */
    END /* END SELECT */
  OTHERWISE                        /* MUST BE A BAD STATUS_FLAG VALUE */
    DO
      RET = 99
      SIGNAL TROUBLE
    END /* END SELECT */
END /* END SELECT */

"LOGOFF OMVW9"
RETURN 0
/* -------- TROUBLE ENCOUNTERED                                    */
TROUBLE:
"WTO 'OMEGAVIEW UPDATE FAILED; "RE"' MSGID(MVUPDATE)"
RETURN -1
```
Returning OMEGAVIEW Column Information

Overview
The STGET() function will return OMEGAVIEW column information. AF/OPERATOR will automatically store the column data in REXX variables whose names are the same as the OMEGAVIEW column names. The STGET() function will not modify the SDMLINE stem variables.

Example
The following exec fragment uses the STGET() function to retrieve data from the status item column SDMDESCR created in “Using a null rule” on page 239:

```rexx
/* REXX */
HNDL='OMV1'
ITEM='OMV1.TESTITEM'

/* THE FOLLOWING RETRIEVES AND DISPLAYS THE STATUS */
/* ITEM COLUMN DATA FROM OMEGAVIEW. */

RET=STGET(HNDL,ITEM)
SAY 'GET = ' ITEM 'RC=' RET
SAY 'SDMDESCR = ' SDMDESCR
```

Your console output will display the data in the SDMDESCR column; it should look like this:

FIGURE 22. Console Output Example

```
GET = OMV1.TESTITEM RC=0000
SDMDESCR = THIS IS A TEST ITEM
```
Automated Session Control

Overview
You can use a combination of traps and Candle-supplied REXX execs to automate regularly scheduled session changes in OMEGAVIEW instead of making the changes manually through OMEGAVIEW’s Configuration Manager.

Controlling OMEGAVIEW sessions from AF/OPERATOR
OMEGAVIEW’s Automated Session Control feature enables you to automate the following actions for OMEGAVIEW status data collector sessions:

- Stop a session
- Start a session
- Update session information
- Stop, update, and then restart a session
- Stop Session Manager, which stops all collector sessions
- Start Session Manager, which starts all collector sessions that have their Automatic Start option set to Yes.

See the OMEGAVIEW documentation for details.
Introduction

Many events occurring within a CICS address space result in messages being issued. Some of these are directed to the MVS console, but some important messages are directed only to transient data queues. Examples are events such as:

- printer or terminal outages
- abends of applications running under CICS
- IRC failures
- CICS logons and logoffs

and other significant events handled by CICS management modules and support programs.

This chapter explains how to use the AF/OPERATOR CICS Interface to trap and process CICS messages written to transient data queues (also referred to as transient data destinations). Examples show you how to:

- trap CICS messages in transient data queues
- take extended action based on these messages

Chapter Contents

What You Need to Know ................................................. 252
Handling CICS Messages .................................................. 253
Trapping CICS Messages in Transient Data Queues ...................... 255
Trapping CICS Terminal Deletion Messages ................................. 258
Using the CICS Interface requires a working knowledge of REXX and CICS, plus familiarity with AF/OPERATOR commands.

If you need to brush up on REXX, review the following IBM references:
- TSO/E REXX/MVS Reference
- TSO/E REXX/MVS User’s Guide

If you need to refer to CICS documentation, review the following IBM references:
- CICS/ESA Customization Guide
- CICS/ESA Facilities and Planning Guide
- CICS/ESA General Information
- CICS/ESA Messages and Codes
- CICS/ESA Resource Definition (Macro)
- CICS/ESA Resource Definition (Online)
- CICS/ESA System Definition Guide

To familiarize yourself with AF/OPERATOR commands, review the AF/OPERATOR Command Reference Manual.
Handling CICS Messages

Forwarding CICS messages

A Candle-supplied CICS Interface exit program is installed as a CICS User Exit during AF/OPERATOR customization. This exit program will be invoked whenever a message is written to a Transient Data destination. It can be customized to select any combination of messages and Transient Data destinations, for example, all messages directed to Transient Data destination CSNE. The selected messages will be directed to AF/OPERATOR using the AOSIM facility. You can then use the AF/OPERATOR TRAP ADD WTO command to define traps that process these messages just as you would MVS console messages. (For information on AOSIM, see “Defining WTO and Command Events with AOSIM” on page 269.)

Transient data queues

Transient data queues, or destinations, are the storage mechanism CICS uses to hold data created by one transaction for later use by the same or a different transaction. You must predefined the transient data queues in the destination control table (DCT).

CICS writes messages to the following transient data queues:

<table>
<thead>
<tr>
<th>Destination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CADL</td>
<td>VTAM resource definition log</td>
</tr>
<tr>
<td>CAIL</td>
<td>Autoinstall terminal model manager (AITM) log</td>
</tr>
<tr>
<td>CCPI</td>
<td>Common Programming Interface for communications messages</td>
</tr>
<tr>
<td>CDBC</td>
<td>CICS-DBCTL interface log</td>
</tr>
<tr>
<td>CDUL</td>
<td>Transaction dump messages</td>
</tr>
<tr>
<td>CMIG</td>
<td>Log for messages reporting use of unsupported functions</td>
</tr>
<tr>
<td>CRDI</td>
<td>Log for installed resource definitions</td>
</tr>
<tr>
<td>CSCS</td>
<td>Sign on/off security log</td>
</tr>
<tr>
<td>CSFL</td>
<td>File allocation and related messages</td>
</tr>
<tr>
<td>CSKL</td>
<td>Log for transaction and profile resource definitions</td>
</tr>
<tr>
<td>CSMT</td>
<td>Write term errors and abends from DFHTACP and DFHACP</td>
</tr>
<tr>
<td>CSPL</td>
<td>Log for program resource definitions</td>
</tr>
<tr>
<td>CSRL</td>
<td>Log for partner resource definitions</td>
</tr>
<tr>
<td>CSTL</td>
<td>Term I/O error messages from DFHTACP</td>
</tr>
<tr>
<td>CSML</td>
<td>Sign on/off messages</td>
</tr>
<tr>
<td>CSNE</td>
<td>Terminal error messages issued from DFHZNAC</td>
</tr>
</tbody>
</table>
Handling CICS Messages

Selecting CICS messages to forward

The person who installs and customizes the CICS Interface at your site selects the criteria for forwarding messages to AF/OPERATOR from the following:

- By destination ID.
  The CICS Interface will forward to AF/OPERATOR any transient data queue messages CICS sends to the destination ID beginning with a specified character. The default character is C (that is, CICS reserved destinations).

- By message prefix.
  By default, the CICS Interface forwards all transient data queue messages to AF/OPERATOR.

- By both destination ID and message prefix.

Summary of CICS Interface commands

The CICS Interface responds to the following CICS commands:

**EXEC CICS LINK or XCTL**

Enables or disables the CICS exit from a user program.

**EXEC CICS START**

Enables or disables the CICS exit from a user program.

**KABC DISABLE**

Disables the CICS exit from a blank CICS screen.

**KABC ENABLE**

Enables the CICS exit from a blank CICS screen.

The CICS Interface responds to the following AF/OPERATOR command:

**TRAP ADD WTO**

Traps messages written by CICS applications to the CICS transient data queues.

For a complete description and syntax of the TRAP ADD WTO command, see the *AF/OPERATOR Command Reference Manual*. For description and syntax of CICS commands, refer to IBM’s CICS documentation.
Trapping CICS Messages in Transient Data Queues

Description
You can trap CICS messages in transient data queues by issuing the AF/OPERATOR TRAP ADD WTO command from the AF/OPERATOR console or from within an AF/OPERATOR REXX exec. After trapping the message you can process it as you would other WTOs. For example, you can trap CICS terminal status messages and route them to other consoles or to system programmers for analysis.

Prerequisites
Before using the AF/OPERATOR CICS Interface to trap and process CICS transient data queue messages, you should verify that someone at your site has:

- Selected the criteria for forwarding CICS messages to AF/OPERATOR by modifying CICS Interface exit program KABCICS (for CICS V2) or KABCICS3 (for CICS V3) as described in the AF/OPERATOR Configuration and Customization Guide.
- Installed CICS Interface exit program KABCICS or KABCICS3 as described in the AF/OPERATOR Configuration and Customization Guide.
- Defined and installed processing program table (PPT) and program control table (PCT) entries for exit program KABCICS or KABCICS3 as described in the AF/OPERATOR Configuration and Customization Guide.
- Started both CICS and AF/OPERATOR.
- Enabled the CICS Interface exit.

Procedure
Follow these steps to create and invoke REXX execs to trap and process CICS messages:

1. Code the AF/OPERATOR REXX exec.
   A. Use the TRAP ADD WTO command to define a trap for CICS transient data queue messages as you would for WTO console messages. For information on coding traps, see “Trapping System Events” on page 35. For a description and syntax of the TRAP ADD command, see the AF/OPERATOR Command Reference Manual.
   B. Include the code you need to take action based on CICS messages. For information on using REXX execs, see the AF/OPERATOR Command Reference Manual.

2. To establish communication between AF/OPERATOR and CICS, ensure that both products are running.

3. Enable the CICS exit by using one of the following methods:
   - Automatically enable the exit on CICS startup by executing either:
     - `DFHPLT TYPE=ENTRY,PROGRAM=KABCICS` (for V2) or
     - `DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM`
Enable the exit from a blank CICS screen by entering:

**KABC ENABLE**

The ENABLE parameter must be exactly one space after the transaction ID (KABC), or the exit will not be enabled, and you will receive message AOPCI007.

Enable the exit from the MVS console by issuing:

**F cicsjobname,KABC ENABLE**

You must position the ENABLE parameter exactly one space after the transaction ID (KABC), or AF/OPERATOR will not enable the exit, and you will receive message AOPCI007. In addition, you must have defined and signed onto the CICS console from which you issue the command. This may necessitate authorizing the console through your security system.

Enable the exit from a user program by issuing the EXEC CICS START command:

**EXEC CICS START TRANSID(‘KABC’) FROM(DATAAREA) + LENGTH(=Y(L’DATAAREA’))**

where DATAAREA contains ENABLE.

Enable the exit from a user program by issuing the EXEC CICS LINK or XCTL command:

**EXEC CICS LINK PROGRAM(‘KABCICS’) COMMAREA(CAREA) + LENGTH(=Y(L’CAREA’))**

where CAREA contains ENABLE.

4. Invoke the REXX exec to trap and process CICS messages.

5. If you need to disable the CICS exit for any reason, use one of the following methods:

Enable the exit from a blank CICS screen by entering:

**KABC DISABLE**

You must position the DISABLE parameter exactly one space after the transaction ID (KABC), or AF/OPERATOR will not disable the exit, and you will receive message AOPCI007.

Disable the exit from the MVS console by issuing:

**F cicsjobname,KABC DISABLE**

You must position the DISABLE parameter exactly one space after the transaction ID (KABC), or AF/OPERATOR will not disable the exit, and you will receive message AOPCI007.

Disable the exit from a user program by issuing the EXEC CICS START command:

**EXEC CICS START TRANSID(‘KABC’) FROM(DATAAREA) + LENGTH(=Y(L’DATAAREA’))**
where DATAAREA contains DISABLE.

- Disable the exit from a user program by issuing the EXEC CICS LINK or XCTL command:

  ```
  EXEC CICS LINK PROGRAM('KABCICS') COMMAREA(CAREA) +
  LENGTH(=Y(L'CAREA'))
  ```

  where CAREA contains DISABLE.
Trapping CICS Terminal Deletion Messages

Introduction
A common problem for data centers is managing print resources. Printers may be used by several applications and systems, and they often have several logical identifiers. Normally an operator would have no indication that CICS had deleted a printer terminal ID until the change affected users. Then, given the CICS terminal ID, the operator would have to refer to a file or document to determine the logical ID for the print management system.

Example
This example shows how to use an AF/OPERATOR trap to detect when CICS deletes the termid for a printer. AF/OPERATOR then executes a REXX exec to match the CICS termid with a system printer ID so that the operator may delete it.

```
"TRAP ADD(PRTPURGE)",
"WTO(DFHZC6966 *DTE *TME *PRO Deletion for terminal: *TERMID *)",
"ACTION(EX PRTPURGE) ENA",
"LOG JOBNAME(ABC1)"
```

Trap command details
This section describes the function of each line of the TRAP ADD WTO command in this example.

```
"TRAP ADD(PRTPURGE)",
```
defines the name of the trap, PRTPURGE.

```
"WTO(DFHZC6966 *DTE *TME *PRO Deletion for terminal: *TERMID *)",
```
indicates that it is a WTO trap and specifies the text of the message to be trapped, in this case the transient data queue message

```
DFHZC6966 Deletion for terminal
```
The following pattern variables are also defined:

<table>
<thead>
<tr>
<th>Pattern Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*DTE</td>
<td>Date</td>
</tr>
<tr>
<td>*TME</td>
<td>Time</td>
</tr>
<tr>
<td>*PRO</td>
<td>Program</td>
</tr>
<tr>
<td>*TERMID</td>
<td>CICS Terminal ID</td>
</tr>
</tbody>
</table>
“ACTION('EX PRTPURGE') ENA”,
defines the extended action for this trap, which is to execute the PRTPURGE exec.

“LOG JOBNAME(CICS)”
turns logging on and specifies that only messages originating from JOBNAME CICS will trigger the trap.
Trapping CICS Terminal Deletion Messages
Introduction

The AF/OPERATOR IMS Interface enables you to use the IMS automated operator interface (AOI) facility to monitor and control IMS activities from the AF/OPERATOR console or from within an AF/OPERATOR REXX exec.

This chapter explains how to use the AF/OPERATOR IMS Interface to trap and process IMS messages and commands. Examples show you how to:

- issue IMS commands from AF/OPERATOR
- trap and process IMS commands
- trap and suppress IMS messages.

Chapter Contents

What You Need to Know .......................................................... 262
Summary of IMS Interface Commands ........................................ 263
Issuing Commands to IMS Regions .......................................... 264
Trapping and Processing IMS Commands ................................. 265
Trapping and Processing IMS Messages ................................. 266
What You Need to Know

Uses of the IMS interface

You can use the IMS Interface to:

- Issue commands to an IMS region by embedding each IMS command in an AF/OPERATOR IMSOPER command.
- Intercept and process IMS system messages destined for the IMS master terminal.
  The IMS interface passes a copy of IMS system messages destined for the master terminal to AF/OPERATOR. You can use AF/OPERATOR’s TRAP ADD WTO command to:
  - trap and suppress the messages
  - modify the messages
  - reroute the messages to another terminal
- Intercept and process IMS commands, entered by an operator at the master terminal, and the responses to those commands.
  The IMS interface passes a copy of the commands and command responses to AF/OPERATOR. You can use AF/OPERATOR’s TRAP ADD CMD command to trap and process them. Although you cannot modify IMS commands or command responses, you can:
  - route a copy of the command or response to other applications
  - examine commands or responses and generate messages to other applications.

Prerequisite knowledge

Using the IMS Interface requires a working knowledge of REXX and IMS, plus familiarity with AF/OPERATOR commands.

If you need to brush up on REXX, review the following IBM references:

- TSO/E REXX/MVS Reference
- TSO/E REXX/MVS User’s Guide

If you are not already familiar with IMS, review the following IBM references:

- IMS/ESA General Information
- IMS/ESA Operations Guide
- IMS/ESA Operator’s Reference
- IMS/ESA System Customization Guide
- IMS/ESA System Definition Reference

To familiarize yourself with AF/OPERATOR commands, review the AF/OPERATOR Command Reference Manual.
Summary of IMS Interface Commands

The IMS Interface responds to the following AF/OPERATOR commands:

**IMSOPER**
Issues a command from AF/OPERATOR to an IMS region by using the IMS WTOR reply ID (described in “Issuing Commands to IMS Regions” on page 264).

**TRAP ADD CMD with JOBTYPE(IMS)**
Traps IMS MTO commands (described in “Trapping and Processing IMS Commands” on page 265).

**TRAP ADD WTO with JOBTYPE(IMS)**
Traps IMS messages (described in “Trapping and Processing IMS Messages” on page 266).

For a complete description and syntax of the IMSOPER and TRAP commands, see the *AF/OPERATOR Command Reference Manual*. 
Issuing Commands to IMS Regions

Description
You can use the AF/OPERATOR IMSOPER command to issue IMS commands to an IMS region. You can issue the IMSOPER command from the AF/OPERATOR console, from a REXX exec, or as the action specification of a trap. See the AF/OPERATOR Command Reference Manual for the syntax of the IMSOPER command.

You can also use the IMSOPER command to build IMS command macros and consolidate IMS commands as follows:
- The operator at an IMS or MVS console enters a short mnemonic text that stands for a sequence of commands to IMS.
- AF/OPERATOR traps and suppresses the mnemonic and executes that sequence of commands by issuing an IMSOPER command.

You must issue an IMS command macro from the MVS console, not from the IMS master terminal, because you cannot suppress or modify commands issued from the IMS master terminal.

Note: If you include more than one IMSOPER command in an exec, AF/OPERATOR attempts to serialize the commands to ensure that they execute in the proper sequence.

Prerequisites
Before using AF/OPERATOR’s IMSOPER command, make sure that both AF/OPERATOR and IMS are running.

Procedure
Follow these steps to create and invoke an AF/OPERATOR REXX exec to issue IMS commands:

1. Code the AF/OPERATOR REXX exec.
   Use AF/OPERATOR IMSOPER commands to issue the appropriate IMS commands. For a description and syntax of the IMSOPER command, see the AF/OPERATOR Command Reference Manual.

2. To establish communication between AF/OPERATOR and IMS, ensure that both products are running.

3. Invoke the REXX exec you have written to issue the IMS commands.
**Trapping and Processing IMS Commands**

**Description**

When an operator enters a command at the IMS master terminal, IMS passes a copy of the command to AF/OPERATOR through the IMS Interface before executing the command. You can define a trap for the IMS command using the AF/OPERATOR TRAP ADD CMD command with JOBTYPE(IMS).

You can write traps for IMS console commands in the same way you would for MVS console commands, with the following qualifications:

- You can trap valid IMS commands issued at the IMS master terminal, but you cannot suppress the commands.
- You can issue a command to IMS from a trap by using the IMSOPER command (described in “Issuing Commands to IMS Regions” on page 264).
- You can use traps and IMSOPER commands to build IMS command macros (described in “Issuing Commands to IMS Regions” on page 264).

**Prerequisites**

Before using the AF/OPERATOR IMS Interface to trap and process IMS commands, you must have:

- installed the AF/OPERATOR IMS Interface AOI exit program KABIMX02 as described in the AF/OPERATOR Command Reference Manual.
- started both IMS and AF/OPERATOR.

**Procedure**

Follow these steps to create and invoke a REXX exec to trap and process IMS commands:

1. Code the AF/OPERATOR REXX exec.
   
   A. Use the TRAP ADD CMD command with JOBTYPE(IMS) to define a trap for IMS commands. For information on coding traps, see “Trapping System Events” on page 35. For a description and syntax of the TRAP ADD CMD command, see the AF/OPERATOR Command Reference Manual.
   
   B. Include the code you need to take action based on IMS commands. For information on AF/OPERATOR commands, see the AF/OPERATOR Command Reference Manual.

2. To establish communication between AF/OPERATOR and IMS, make sure that both products are running.

3. Invoke the REXX exec you have written to trap and process the IMS commands.
Trapping and Processing IMS Messages

Description

When an IMS application generates a message destined for the IMS master terminal, IMS passes a copy of the message to AF/OPERATOR through the IMS Interface before sending the command to the master terminal. You can define a trap for the IMS message using the AF/OPERATOR TRAP ADD WTO command with JOBTYPE(IMS).

You can write traps for IMS console messages in the same way as you would for MVS console messages.

- You can trap IMS system messages destined for the master terminal operator (MTO) and then suppress, modify, or reroute the messages.
- You can trap responses to commands entered by the MTO. You can route copies of the responses to other applications or generate and send new messages to other applications. You cannot suppress or modify the responses themselves.
- IMS occasionally sends a message to the MTO and then issues the same message as a WTO to the MVS console. This results in two events being monitored by AF/OPERATOR. You should code traps that distinguish between the two events. Use the JOBNAME, JOBTYPE, or SYSID keywords on the TRAP ADD command to filter the events that will trigger the trap. These keywords take special parameters in the IMS environment.

Each IMS message or command has the following characteristics:

- The job type (AOJTYPE) is IMS. (That is, the job type is designated by the character string IMS.)
- The job name (AOJNAME) is the LTERM for which the message was destined. (This is often MTO.)
- The system ID (AOSID) is the name of the IMS control region.

If a WTO trap includes the SUPPRESS option and the operand JOBTYPE(IMS), it will prevent the messages it traps from being displayed on the IMS MTO.

Since IMS messages are intercepted by an IMS exit, AF/OPERATOR cannot trap messages that are not routed through this exit.

To trap IMS messages, set the SYSID parameter of the TRAP ADD WTO command to the name of the IMS control region in which the message originates, unless you have installed the Subsystem Logging Facility. In this case, you can use the actual SYSID and IMS region name instead of the IMS control region name.
Prerequisites

Before using the AF/OPERATOR IMS Interface to trap and process IMS messages, you must have:

- installed the AF/OPERATOR IMS Interface AOI exit program KABIMX02 (described in the AF/OPERATOR Configuration and Customization Guide.)
- started both IMS and AF/OPERATOR

Procedure

Follow these steps to create and invoke a REXX exec to trap and process IMS messages:

1. Code the AF/OPERATOR REXX exec.
   A. Use the TRAP ADD WTO command with JOBTYPE(IMS) to define a trap for IMS messages. For information on coding traps, see “Trapping System Events” on page 35. For a description and syntax of the TRAP command, see the AF/OPERATOR Command Reference Manual.
   B. Include code to take action based on these IMS messages. For information on AF/OPERATOR commands, see the AF/OPERATOR Command Reference Manual.

2. To establish communication between AF/OPERATOR and IMS, make sure that both products are running.

3. Invoke the REXX exec you have written to trap and process the IMS messages.
Defining WTO and Command Events with AOSIM

Introduction

AOSIM is a Candle program that provides a way to pass messages and commands from user-written or third-party vendor programs to AF/OPERATOR without using MVS SVCs. It is particularly useful in environments in which you can not issue SVCs (for example, CICS, JES3 or IMS). A typical implementation is to call AOSIM from a message or command exit within the special environment. That exit must:

- acquire a work/save area for AOSIM to use
- build a parameter list telling AOSIM what to do
- call AOSIM
- react to any return codes from AOSIM
- free the work/save area

Chapter Contents

AOSIM Components .......................................................... 270
KAB$SAVE — Begin Save Area/Working Storage ..................... 272
KAB$SAVD — End of Save Area/Working Storage .................... 273
KAB$SIMS — Working Storage for AOSIM ............................. 274
KAB$SIMP — AOSIM Parameter Block ................................. 275
Modifiable Global Match Variables ................................. 278
Setting Up the Environment .............................................. 280
Creating the Save Areas .................................................. 281
Identifying a Request Type Parameter Block ......................... 283
Identifying an Override Type Parameter Block ....................... 284
Information Returned to the Caller by AOSIM ....................... 285
Examples ............................................................. 286
AOSIM Components

The interface provides these assembler macros and copy code to help create the correct control structures for AOSIM.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KAB$$GLB</td>
<td>Copy code Defines assembler global variables used within the AOSIM macro set.</td>
</tr>
<tr>
<td>KAB$$SAVE</td>
<td>Macro Begins the building of a save/work area for the calling program.</td>
</tr>
<tr>
<td>KAB$$SAVD</td>
<td>Macro Correctly terminates the calling program’s save/work area.</td>
</tr>
<tr>
<td>KAB$$SIMS</td>
<td>Macro Sets up the save/work area for AOSIM.</td>
</tr>
<tr>
<td>KAB$$SIMP</td>
<td>Macro Creates or maps an AOSIM parameter entry.</td>
</tr>
</tbody>
</table>

The interface to AOSIM is similar to the MVS standard linkage:

- General-purpose register 15 must point to the entry point of AOSIM.
- Register 14 must point to the address to which AOSIM is to return control.
- Register 13 must point to the special register save area and AOSIM work area
- Register 1 points at the AOSIM parameter blocks.

Accessing AOSIM

You can access AOSIM using the MVS CALL macro (or any other method that results in a BALR R14,R15 or BASR R14,R15 type invocation). But take care to construct and address the save/work areas and parameter blocks correctly.

In order to avoid issuing SVCs to the LINK, LOAD, or ATTACH services, Candle recommends that you link-edit AOSIM into your module. You must relink your programs to take advantage of any maintenance to the AOSIM module.

AOSIM macro reference

The assembler macros provided for the AOSIM interface provide only data mapping and definition. The following sections describe these macros and their parameters. These descriptions should be reviewed in conjunction with the code fragment on page 286.

Requirements for parameter blocks

Parameter blocks used for AOSIM must meet these requirements:

- They must be in contiguous storage.
- They must be addressed by register 1 on entry.
- The first parameter block must be a request type block (TYPE=R or TYPE=N).
- The first parameter block must contain a count of the number of subsequent override blocks.
Override blocks must each describe a value field, which can be anywhere in addressable storage, by specifying the field’s length and providing the address of the override data.

**Note:** AOSIM currently runs with addressing mode ANY and residency mode 24 (AMODE-ANY, RMODE-24), but all storage areas and parameter blocks passed must be in 24-bit addressable storage. Your calling program can be either 31- or 24-bit addressing mode.
KAB$SAVE — Begin Save Area/Working Storage

Description
The KAB$SAVE macro begins the definition of the storage necessary for a register save area and working storage for AOSIM. On calling AOSIM, register 13 must address this area. (The work area starts with a 72-byte register save area, which the invoking program can use as its save area for non-AOSIM calls.)

This description should be reviewed in conjunction with the code fragment on page 286.

Syntax
The KAB$SAVE macro is written as follows:

```
name
KAB$SAVE
```

Parameters
The parameters for KAB$SAVE are:

```
DSECT=xxx
```

Specifies whether a DSECT card should be generated:

**YES**
A DSECT card is generated.

**NO**
A DSECT card is not generated.

By default the macro declares the area as a dummy section, in which case the DSECT name will be AOSAVE.
KAB$SAVD — End of Save Area/Working Storage

Description

The KAB$SAVD macro formally delimits the end of the save area for AOSIM. This macro declares an assembler symbol SAVESIZE that equates to the full length of the working storage area begun by the KAB$SAVE macro.

*Note:* This macro has no parameters.

This description should be reviewed in conjunction with the code fragment on page 286.

Syntax

The KAB$SAVD macro is written as follows:

- **name**: Name: symbol. Begin name in column 1.
- One or more blanks must precede KAB$SAVD.
- **KAB$SAVD**: One or more blanks must follow KAB$SAVD.
Description

The KAB$SIMS macro declares the working storage required by AOSIM. Callers of the AOSIM service must acquire this storage for AOSIM’s use.

Note: AOSIM assumes that this work area is contiguous to the caller’s save area (addressed by register 13) as defined by a KAB$SAVE macro.

This description should be reviewed in conjunction with the code fragment on page 286.

Syntax

The KAB$SIMS macro is written as follows:

name

Name: symbol. Begin name in column 1.
One or more blanks must precede KAB$SIMS.

KAB$SIMS

One or more blanks must follow KAB$SIMS.
KAB$SIMP — AOSIM Parameter Block

Description
Use the KAB$SIMP macro to construct or map the parameter blocks needed for a call to AOSIM. The macro constructs one block per invocation and can be used to begin generation of both types of parameter block needed by the interface. Optionally, you can use the macro to build a mapping dummy section in order to address fields within the parameter blocks.

This description should be reviewed in conjunction with the code fragment on page 286.

Syntax

```
name        Name: symbol. Begin name in column 1.
            One or more blanks must precede KAB$SIMP.
KAB$SIMP    One or more blanks must follow KAB$SIMP.
```

Parameters
The parameters for KAB$SIMP are:

```
DSECT=xxx    Specifies whether a DSECT card should be generated:
            YES    A DSECT card is generated.
            NO     A DSECT card is not generated.
            By default, the macro declares the area as a dummy section; in this case,
            the DSECT name is AOSIMPRM.
            Note: If you specify PREFIX=prefix the macro will begin the DSECT
            name with the specified prefix.

PREFIX=prefix Specifies a 2-character prefix the macro uses to name the fields it declares.
            The default prefix is AO.

TYPE=x       Specifies which type of parameter block to build:
            R       Builds a request parameter block.
            O       Builds a variable override parameter block.
            N       Builds a non-queued request block.
```
**SSNAME= subsystem_name**

Sub-system name: symbol.
Specifies the 1- to 4-character subsystem name AOSIM calls to process the request. If omitted, the macro generates a field of binary zeros that direct the event to the first active AF/OPERATOR address space.

**Note:** If you specify SSIREQ=BROADCAST then the SSNAME parameter is ignored. This is only valid for TYPE=R or TYPE=N macro expansions.

**SSIREQ= xxxx xxxxxx**

Sub-system name: symbol
Specifies how the event should be broadcast:

- **DIRECT** Broadcast event to a specific AF/OPERATOR subsystem.
- **BROADCAST** Broadcast event to all active AF/OPERATOR subsystems and ignore the SSNAME= parameter.

**Note:** This parameter is only valid for TYPE=R or TYPE=N macro expansions.

**BLOCKS= number**

*number*: symbol or decimal digit.
Specifies the number of override blocks that follow for this request. (Only valid for TYPE=R or TYPE=N macro expansions.)

**REQTYPE= x**

Specifies the type of request:

- **C** Command simulation.
- **W** WTO (message) to AF/OPERATOR simulation.

**Note:** This parameter is valid only for TYPE=R or TYPE=N macro expansions.

**VARNANME= varname**

*varname*: symbol.
Specifies the name of the variable to override. For which global variables you can override, see “Modifiable Global Match Variables” on page 278. This parameter is valid for TYPE=O macro expansions only.

**VARLEN= length**

*length*: any valid value from 1 to 128.
Specifies the length of the override data to insert in the variable. Only valid for TYPE=O macro expansions.

**VARTYPE= x**

Specifies the data type of the variable AOSIM is to override:
C  Character variable
I  Binary integer variable
   This parameter is valid only for TYPE=O macro expansions.

VARVAL=value_addr
   value_addr: A-type address.
   Specifies the address of the field that contains the data to insert in the override variable. This parameter is valid only for TYPE=O macro expansions.
Modifiable Global Match Variables

Introduction

The AOSIM facility can modify a subset of AF/OPERATOR-supplied global match variables.

Description

AOSIM supports message text of up to 128 characters, which is the same length as a match. This table lists the AF/OPERATOR global match variables whose fields you can modify, and describes their default attributes.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Type</th>
<th>Field Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOTEXTW</td>
<td>Message (WTO) text</td>
<td>Char</td>
<td>128</td>
</tr>
<tr>
<td>AOTEXTC</td>
<td>Command text</td>
<td>Char</td>
<td>128</td>
</tr>
<tr>
<td>AOTEXT</td>
<td>Alternate name for AOTEXTW or AOTEXTC</td>
<td>Char</td>
<td>128</td>
</tr>
<tr>
<td>AOJCLAS</td>
<td>Job class</td>
<td>Char</td>
<td>1</td>
</tr>
<tr>
<td>AOASID</td>
<td>Issuer address space ID</td>
<td>Char</td>
<td>2</td>
</tr>
<tr>
<td>AOJTYPE</td>
<td>Job type</td>
<td>Char</td>
<td>3</td>
</tr>
<tr>
<td>AOJID</td>
<td>Job ID</td>
<td>Char</td>
<td>5</td>
</tr>
<tr>
<td>AOJNAME</td>
<td>Job name</td>
<td>Char</td>
<td>8</td>
</tr>
<tr>
<td>AOSID</td>
<td>System ID</td>
<td>Char</td>
<td>8</td>
</tr>
<tr>
<td>AOUSER</td>
<td>User field</td>
<td>Char</td>
<td>4</td>
</tr>
<tr>
<td>AOAUTH</td>
<td>Command authority level</td>
<td>Char</td>
<td>1</td>
</tr>
<tr>
<td>AOMATDYS</td>
<td>Day of week bit map</td>
<td>Char</td>
<td>1</td>
</tr>
<tr>
<td>AOPNAME</td>
<td>JES2 Programmer name</td>
<td>Char</td>
<td>20</td>
</tr>
<tr>
<td>AOACCT</td>
<td>JES2 Account number</td>
<td>Char</td>
<td>4</td>
</tr>
<tr>
<td>AOROOMN</td>
<td>JES2 Bin number</td>
<td>Char</td>
<td>4</td>
</tr>
<tr>
<td>AOTSUID</td>
<td>JES2 Notify user ID</td>
<td>Char</td>
<td>7</td>
</tr>
<tr>
<td>AONONDE</td>
<td>JES2 Originating node ID</td>
<td>Char</td>
<td>8</td>
</tr>
<tr>
<td>AOMCLAS</td>
<td>JES2 Message class</td>
<td>Char</td>
<td>1</td>
</tr>
<tr>
<td>AOINDEV</td>
<td>JES2 Input device</td>
<td>Char</td>
<td>8</td>
</tr>
<tr>
<td>AOMSGPFX2</td>
<td>Write-to-Operator Message prefix</td>
<td>Char</td>
<td>23</td>
</tr>
<tr>
<td>AOWTONO2</td>
<td>Write-to-Operator number</td>
<td>Char</td>
<td>3</td>
</tr>
</tbody>
</table>
### Variable Name | Description | Type | Field Length
--- | --- | --- | ---
AOROUT2 | Write-to-Operator Route code | Char | 2
AODESC2 | Write-to-Operator Descriptor code | Char | 2
AOREPID2 | Write-to-Operator Reply ID | Char | 2
AOCONNM | Console name | Char | 8
AOCONS | Console ID | Int | 4
AOLID | LINKID of AF/OPERATOR address space sending the event | Char | 8

**Notes:**
- AOTEXTW only applies to message (WTO) type requests. (KAB$SIMP TYPE=R, REQTYPE=W parameter block macro expansions.)
- AOTEXTC only applies to command type requests. (KAB$SIMP TYPE=R, REQTYPE=C parameter block macro expansions.)
Setting Up the Environment

Requirements

The KAB$SAVE and KAB$SIMS macros map the AOSIM work/save area. The work/save area must be large enough to contain all of the fields included in the KAB$SIMS macro. In order for AOSIM to properly locate its work area, allocate storage and use the KAB$SAVE macro in conjunction with a KAB$SIMS macro, then have register 13 address the beginning of the structure. The KAB$SIMS work area must be contiguous to the KAB$SAVE data structure.

The KAB$SAVE structure begins with a 72-byte save area that the calling program can use as its register save area. If the calling program meets the previous requirements for its work/save area, then it can allocate storage in excess of that set aside for AOSIM as general working storage.

Caution

Use of AOSIM with a work/save area of insufficient size can result in miscellaneous abends, possible termination of AF/OPERATOR, and possible damage to MVS itself. AOSIM will use the first 72 bytes to save the caller’s registers. AOSIM needs the remainder in order to complete its work.

On entry to AOSIM register 1 must address a parameter block as follows:

- One request type parameter block for the type of request: a command or a message.
- One or more override type parameter blocks to identify specific attributes that AOSIM should alter. For example, you may want a message from JES3 to appear to be from JOBXX, since the message is about JOBXX.

By default, AOSIM takes task attributes from the address space that calls it. The message or command text will default to blanks.

The KAB$SIMP macro is available to help build and map these request and override blocks.
Creating the Save Areas

Layout of KAB$SAVE save area

The table below shows the layout of the basic save area for a call to AOSIM generated by the KAB$SAVE macro.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOSAVE</td>
<td>+00</td>
<td>Structure</td>
<td>varies</td>
<td>Beginning of save area</td>
</tr>
<tr>
<td>AOSAVSIZ</td>
<td>+00</td>
<td>Signed</td>
<td>4</td>
<td>Size of save area getmained</td>
</tr>
<tr>
<td>AOSAVSPL</td>
<td>+00</td>
<td>Signed</td>
<td>1</td>
<td>Subpool of save area getmained</td>
</tr>
<tr>
<td>AOSAVHI</td>
<td>+04</td>
<td>Address</td>
<td>4</td>
<td>Address of next higher save area</td>
</tr>
<tr>
<td>AOSAVLOW</td>
<td>+08</td>
<td>Address</td>
<td>4</td>
<td>Address of next lower save area</td>
</tr>
<tr>
<td>AOSAVR14</td>
<td>+0C</td>
<td>Signed</td>
<td>4</td>
<td>Caller’s saved register 14</td>
</tr>
<tr>
<td>AOSAVR15</td>
<td>+10</td>
<td>Signed</td>
<td>4</td>
<td>Caller’s saved register 15</td>
</tr>
<tr>
<td>AOSAVR00</td>
<td>+14</td>
<td>Signed</td>
<td>4</td>
<td>Caller’s saved register 0</td>
</tr>
<tr>
<td>AOSAVR0</td>
<td>+14</td>
<td>Signed</td>
<td>4</td>
<td>Alternate name for AOSAVR00</td>
</tr>
<tr>
<td>AOSAVR01</td>
<td>+18</td>
<td>Signed</td>
<td>4</td>
<td>Caller’s saved register 1</td>
</tr>
<tr>
<td>AOSAVR1</td>
<td>+18</td>
<td>Signed</td>
<td>4</td>
<td>Alternate name for AOSAVR1</td>
</tr>
<tr>
<td>AOSAVR02</td>
<td>+1C</td>
<td>Signed</td>
<td>4</td>
<td>Caller’s saved register 2</td>
</tr>
<tr>
<td>AOSAVR2</td>
<td>+1C</td>
<td>Signed</td>
<td>4</td>
<td>Alternate name for AOSAVR2</td>
</tr>
<tr>
<td>AOSAVR03</td>
<td>+20</td>
<td>Signed</td>
<td>4</td>
<td>Caller’s saved register 3</td>
</tr>
<tr>
<td>AOSAVR3</td>
<td>+20</td>
<td>Signed</td>
<td>4</td>
<td>Alternate name for AOSAVR3</td>
</tr>
<tr>
<td>AOSAVR04</td>
<td>+24</td>
<td>Signed</td>
<td>4</td>
<td>Caller’s saved register 4</td>
</tr>
<tr>
<td>AOSAVR4</td>
<td>+24</td>
<td>Signed</td>
<td>4</td>
<td>Alternate name for AOSAVR4</td>
</tr>
<tr>
<td>AOSAVR05</td>
<td>+28</td>
<td>Signed</td>
<td>4</td>
<td>Caller’s saved register 5</td>
</tr>
<tr>
<td>AOSAVR5</td>
<td>+28</td>
<td>Signed</td>
<td>4</td>
<td>Alternate name for AOSAVR5</td>
</tr>
<tr>
<td>AOSAVR06</td>
<td>+2C</td>
<td>Signed</td>
<td>4</td>
<td>Caller’s saved register 6</td>
</tr>
<tr>
<td>AOSAVR6</td>
<td>+2C</td>
<td>Signed</td>
<td>4</td>
<td>Alternate name for AOSAVR6</td>
</tr>
</tbody>
</table>
## Creating the Save Areas

### Layout of KAB$SAVD

The table below shows the layout of the KAB$SAVD save area delimiter macro.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAVESIZE</td>
<td>Varies</td>
<td>Symbol</td>
<td>0</td>
<td>Length of save area</td>
</tr>
</tbody>
</table>
Identifying a Request Type Parameter Block

Parameter layout

The parameter layout to identify the request (mapped by a KAB$SIMP TYPE=R,PREFIX=AO macro call) is:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOSIMVAR</td>
<td>+00</td>
<td>Char</td>
<td>4</td>
<td>Always SIMP for AOSIM calls.</td>
</tr>
<tr>
<td>AOSIMSREQ</td>
<td>+04</td>
<td>Binary</td>
<td>1</td>
<td>SSIREQ flags.</td>
</tr>
<tr>
<td>AOSIMFLGS</td>
<td>+05</td>
<td>Binary</td>
<td>3</td>
<td>Reserved.</td>
</tr>
<tr>
<td>AOSIMBLK</td>
<td>+08</td>
<td>Binary</td>
<td>2</td>
<td>Number of override parameter entries that follow this block. (Not including this block.)</td>
</tr>
<tr>
<td>AOSIMTYP</td>
<td>+0A</td>
<td>Char</td>
<td>1</td>
<td>Type of call: C for a command and W for a message (WTO.)</td>
</tr>
<tr>
<td>AOSIMFLG</td>
<td>+0B</td>
<td>Char</td>
<td>1</td>
<td>Always D for the first parameter block.</td>
</tr>
<tr>
<td>AOSIMSSN</td>
<td>+0C</td>
<td>Char</td>
<td>4</td>
<td>Name of the subsystem to receive the call. If blanks or binary zeros, the first AF/OPERATOR address space started after the system IPL will receive the call.</td>
</tr>
<tr>
<td>AOSIMELN</td>
<td>+10</td>
<td>Symbol</td>
<td>0</td>
<td>Length of the parameter block.</td>
</tr>
</tbody>
</table>
Identifying an Override Type Parameter Block

Parameter layout

This table gives the parameter layout to identify an override (mapped by a KAB$SIMP TYPE=O,PREFIX=AO macro call)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOSIMVAR</td>
<td>+00</td>
<td>Char</td>
<td>8</td>
</tr>
<tr>
<td>AOSIMLEN</td>
<td>+08</td>
<td>Binary</td>
<td>2</td>
</tr>
<tr>
<td>AOSIMTYP</td>
<td>+0A</td>
<td>Char</td>
<td>1</td>
</tr>
<tr>
<td>AOSIMFLG</td>
<td>+0B</td>
<td>Char</td>
<td>1</td>
</tr>
<tr>
<td>AOSIMVAL</td>
<td>+0C</td>
<td>Addr.</td>
<td>4</td>
</tr>
<tr>
<td>AOSIMELN</td>
<td>+10</td>
<td>Symbol</td>
<td>0</td>
</tr>
</tbody>
</table>

- **AOSIMVAR**: The name of the AF/OPERATOR global match variable whose value you want to override (for example AOJNAME or AOSID).
- **AOSIMLEN**: Length of the value passed in AOSIMVAL.
- **AOSIMTYP**: Type of override value: C for a character value.
- **AOSIMFLG**: Always 0 for an override block.
- **AOSIMVAL**: This is a pointer to the override value field. The field itself may be a binary integer of length 1—4, or it may be character.
- **AOSIMELN**: Length of the parameter block.
Information Returned to the Caller by AOSIM

Hexadecimal return codes

When AOSIM returns control, register 15 contains one of the following hexadecimal return codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>No match was found.</td>
</tr>
<tr>
<td>04</td>
<td>A match was found.</td>
</tr>
<tr>
<td>08</td>
<td>There was an error in the input parameter list.</td>
</tr>
<tr>
<td>0C</td>
<td>AF/OPERATOR was not active.</td>
</tr>
</tbody>
</table>

Results area offsets

If the return code in register 15 is 4, then register 1 contains the address of a results area that contains:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+00</td>
<td>2</td>
<td>New descriptor codes</td>
</tr>
<tr>
<td>+02</td>
<td>2</td>
<td>New route codes</td>
</tr>
<tr>
<td>+04</td>
<td>1</td>
<td>Message management flags</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Suppress.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOLOG specified.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Descriptor code update</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Route code update.</td>
</tr>
<tr>
<td>+05</td>
<td>1</td>
<td>Flag byte.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Trap had the WAIT option.</td>
</tr>
</tbody>
</table>

The contents of registers 2—14 contain the same information as before control was given to AOSIM.
AOSIM calling sequence

The following example shows how to call AOSIM using Basic Assembly Language.

```
SIMSAMP   CSCT   ,
PUSH    PRINT
PUSH    OFF
COPY    KAB$$GLB
POP     PRINT
...
  ... NORMAL PROGRAM ENTRY CODE --
  ... ALLOCATE AND INITIALIZE STORAGE ETC.
LA      R13, AOSAVE
USING   AOSAVE,R13
...
  ... MORE SETUP CODE
...
CALLSIM  CALL    AOSIM,MF=(E,SIMPARMS)
...
  ... CHECK RETURN CODES, MODULE CLEANUP ETC.
...
* ------------------------------------------------------------------
*          STATIC STORAGE, LITERAL POOL ETC.
* ------------------------------------------------------------------
*          USER PROGRAM’S FIXED STORAGE AND CONSTANTS
SIMPARMS  KAB$$IMP DSECT=NO,TYPE=R,PREFIX=AO,
          SSNAME=O120,REQTYPE=W,BLOCKS=1,SSIREQ=DIRECT
KAB$$IMP DSECT=NO,TYPE=O,PREFIX=O1,
          VARNAME=AOTEXT,VARTYPE=C,VARLEN=L,MYTEXT,VARVAL=MYTEXT
...
MYTEXT    DC     CL128 ’THIS IS THE WTO MESSAGE TEXT’
...
  ... MORE CONSTANTS
...
LITERALS  LTORG  ,
* ------------------------------------------------------------------
*          WORKING STORAGE
* ------------------------------------------------------------------
KAB$$SAVE,
KAB$$SIMS
  ... USER=PROGRAM’S WORKING STORAGE FIELDS ETC.
KAB$$SAVD ,
END    SIMSAMP
```
Parameter lists

In this example the parameter list would expand to the following:

```
AOSIMPARM DS OD
AOSIMVAR DC CL4’SIMP’ VARIABLE NAME
AOSIMREQ DC AL1(01000000) SSIREQ FLAG
AOSIMFLGS DS XL3 RESERVED
AOSIMBLK DC H’1’ NUMBER OF OVERRIDES
AOSIMTYP DC CL1’W’ WTO TYPE LIST
AOSIMFLG DC CL1’D’ "DESCRIPTOR” BLOCK
AOSIMSSN DC CL4’0120’ TARGET SUBSYSTEM NAME
O1SIMPARM DS OD
O1SIMVAR DC CL8’AOTEXT’ VARIABLE NAME
O1SIMLEN DC 0H’0’, Y (L’MYTEXT) LENGTH OF VARIABLE
O1SIMTYP DC CL1’C’ CHARACTER TYPE VARIABLE
O1SIMFLG DC CL1’O’ "OVERRIDE” BLOCK
O1SIMVAL DC A(MYTEXT) ADDRESS OF VARIABLE TEXT
```

AF/OPERATOR provides a working sample program, KABSIMX0, in the USERASM library. The program simulates issuing a WTO and builds an override block for the AOJNAME variable. Once assembled and built, run it as a batch job or execute it from TSO/E using TSO’s CALL command.
This Appendix describes the way in which you can manage the use of REXX in the AF/OPERATOR environment.
REXX Performance — In-Storage EXEC Management

Introduction

AF/OPERATOR provides support for maintaining REXX execs in storage instead of re-reading them from DASD. If your site uses many REXX execs, this will improve system performance by avoiding unnecessary I/O to the command file library.

You can manage REXX execs by retaining them in storage after execution, pre-loading them into storage, or removing them from storage. You accomplish this by using the AUTOREF/NOAUTOREF operands of the AF command, and the following AF/OPERATOR commands:

- EXECLOAD
- EXECDROP
- EXECMAP

AF/OPERATOR supports both interpreted and compiled REXX execs if the IBM support for these features is installed at your site.

Use count

A fundamental aspect of in-storage execution of REXX execs is ensuring that they cannot be removed from storage while being used. AF/OPERATOR maintains a use count for each exec in storage. The count is incremented every time an exec is executed and decremented when it ends.

An in-storage exec will not be removed from storage unless its use count is zero: even when the AUTOREF operand of the AF command is in effect. Also, an exec which is in use cannot be reloaded from DASD.

AUTOREF/NOAUTOREF commands

The AUTOREF/NOAUTOREF operands of the AF command control retention in storage of REXX execs.

- **AUTOREF** Causes removal of an exec from storage when its use count reaches zero. AUTOREF assists in development of REXX execs.
- **NOAUTOREF** Causes an exec to be retained in storage when its use count reaches zero. NOAUTOREF produces optimum performance of REXX execs.

See the AF/OPERATOR Command Reference Manual for more information about these operands.
REXX Error Recovery

Introduction

Abends in AF/OPERATOR commands intercepted by recovery will raise the failure condition in REXX, which can be intercepted by a SIGNAL ON FAILURE statement. If the SIGNAL ON FAILURE statement is used, execution will proceed to the routine specified. If the statement is omitted, execution will continue with the next REXX statement following the failing command. In either case, the REXX variable RC (return code from the command) will be set to the decimal value of the abend code. For example 193 = x “0C1”.

Non-zero return codes from AF/OPERATOR commands will raise the error condition. This condition can be intercepted using the SIGNAL ON ERROR statement. If the SIGNAL ON ERROR statement is omitted, execution will proceed with the next REXX statement following the command. In either case the REXX variable RC will be set to a positive value if an error occurs.

If SIGNAL ON FAILURE is omitted and a SIGNAL ON ERROR statement is used instead, then either a failure condition or an error condition can initiate an error recovery routine. The error routine specified on the SIGNAL ON ERROR statement can determine which condition occurred by using the REXX function CONDITION().

Using SIGNAL ON ERROR

You can use a SIGNAL ON ERROR statement to verify the successful completion of an AF/OPERATOR command, as shown in the following example. This REXX exec fragment allocates a dataset using the ALLOC command. If either a command failure occurs or an error occurs (for example, the specified dataset name does not exist), program execution proceeds to the error recovery section.

```
SIGNAL ON ERROR
.
  SOME STATEMENTS
.
  ADDRESS APHOST "ALLOC DDNAME(OUTFILE) DSN('VALID.DS.NAME') SHR"
.
  MORE STATEMENTS
.
ERROR:
  IF CONDITION('C') = 'FAILURE' THEN DO
    SAY "ALLOC COMMAND HAS ABENDED"
    EXIT RC
  END
  ELSE IF CONDITION('C') = 'ERROR' THEN RETURN
  ELSE DO
    SAY "THE CONDITION TRAPPED IS" CONDITION('C')
    EXIT RC
  END
```
Using MSMF to Communicate between AF/OPERATOR Peers

Introduction

You can establish bi-directional (peer-to-peer) communication, through VTAM, between AF/OPERATOR address spaces on the same MVS system or on different MVS systems using the AF/OPERATOR optional feature called MSMF (Multi-System Management Facility). MSMF should already be installed and customized for your site.

Appendix Contents

About MSMF ................................................................. 294
Summary of MSMF Commands ......................................... 295
Starting the MSMF Interface ........................................... 296
Stopping the MSMF Interface .......................................... 298
Defining Links Between Systems ..................................... 300
Starting Links Between AF/OPERATOR Systems ................. 302
Stopping Links Between Systems ..................................... 304
Deleting Link Definitions ............................................... 306
Displaying Link Information .......................................... 307
Sending WTOs to a Target System .................................. 308
Issuing Commands to a Target System .............................. 310
About MSMF

You can use MSMF to:

- Escalate events across AF/OPERATOR address spaces
- Verify events on one AF/OPERATOR address space before taking action on another
- Consolidate messages from several AF/OPERATOR address spaces and send them to one master console
- Consolidate console traffic of remote and local sites into a single-image console display for multiple installations or for an installation with multiple processors

Multiple address space considerations

Consider these points when setting up peer-to-peer communication in a multiple address space environment.

- Candle recommends that you keep your peer-to-peer links simple. You should start by linking only the AF/OPERATOR address spaces on each MVS system that have a relative order of zero RO(0).
- In order to issue a command to an AF/OPERATOR address space from within a REXX exec running in a different AF/OPERATOR address space, you must use peer-to-peer communication. The subsys prefix only applies to commands issued from the console.
- Every event message includes the sending AF/OPERATOR address space’s LINKID. This allows the receiving peer address space to filter events using the LINKID option on the TRAP ADD and TRAP CHANGE commands.
- You can find the LINKID value from the sending system in the match variable AOLID.
Summary of MSMF Commands

The following AF/OPERATOR commands, which can be used to communicate between AF/OPERATOR address spaces on different MVS systems, are discussed in this chapter.

For a complete description and the syntax of each, see the See “Syntax Diagrams” on page 13..

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEER START</td>
<td>Starts the MSMF interface</td>
</tr>
<tr>
<td>PEER STOP</td>
<td>Stops the MSMF interface</td>
</tr>
<tr>
<td>PEERLINK DEFINE</td>
<td>Defines links to peer address spaces</td>
</tr>
<tr>
<td>PEERLINK START</td>
<td>Starts links to peer address spaces</td>
</tr>
<tr>
<td>PEERLINK STOP</td>
<td>Stops links to peer address spaces</td>
</tr>
<tr>
<td>PEERLINK DELETE</td>
<td>Deletes link definitions</td>
</tr>
<tr>
<td>PEERLINK DISPLAY</td>
<td>Displays link information</td>
</tr>
<tr>
<td>WTO</td>
<td>Sends WTOs to a target AF/OPERATOR address space</td>
</tr>
<tr>
<td>OPER</td>
<td>Issues Commands to a target AF/OPERATOR address space</td>
</tr>
</tbody>
</table>
Starting the MSMF Interface

Introduction

The PEER START command activates the MSMF interface on a local AF/OPERATOR system. When establishing communication links between AF/OPERATOR products, you must start the MSMF interface on each AF/OPERATOR address space in your peer-to-peer configuration, so the PEER START command will be issued once for each copy of AF/OPERATOR.

If the MSMF interface has not been started, none of the commands to activate communication links between peer systems will work. For this reason, PEER START must be the first command issued in a REXX exec that establishes a peer-to-peer configuration.

Requirements

To establish peer-to-peer communications, each system in your planned configuration must meet the following requirements:

- AF/OPERATOR must be installed and running
- the VTAM environment must be operational and customized for AF/OPERATOR (described in the AF/OPERATOR Configuration and Customization Guide)

What you need to know

To start the MSMF interface you will need the following information from the person who configured AF/OPERATOR on the local system:

- the application ID (LU name) that defines AF/OPERATOR to VTAM on the local system
- the VTAM logmode table entry that defines the desired session parameters and characteristics for the local system

Startup process

When you start the MSMF interface:

1. The AF/OPERATOR interface to VTAM initializes.
2. The interface activates system resources such as local network services, the resource manager, and the control operator.
3. The interface identifies AF/OPERATOR system and session requirements for VTAM.
4. The local system’s access control block opens so VTAM can find the AF/OPERATOR APPLID.
PEER START

For a complete description and syntax of the PEER START command, see the AF/OPERATOR Command Reference Manual.

```
subsys 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 AF/OPERATOR to VTAM on the local system.

**MODE(modename)**
The VTAM logmode table entry definition that contains the session parameters and characteristics for the local system.

**Procedure**

Follow this procedure to start the MSMF interface:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Vary the luname active on the local system:</td>
</tr>
<tr>
<td></td>
<td>V NET,ACT,ID=aonode</td>
</tr>
<tr>
<td></td>
<td>where <code>aonode</code> is a major node containing the AF/OPERATOR <code>luname</code>.</td>
</tr>
<tr>
<td>2</td>
<td>Execute the PEER START command to start the MSMF interface:</td>
</tr>
<tr>
<td></td>
<td><strong>PEER START APPLID(luname) MODE(modename)</strong></td>
</tr>
<tr>
<td><strong>Result:</strong></td>
<td>The console displays messages describing the startup process. When the console displays <strong>PEER INTERFACE INITIALIZATION SUCCESSFULLY COMPLETED</strong> the MSMF interface is activated.</td>
</tr>
</tbody>
</table>
**Stopping the MSMF Interface**

**Introduction**

Use the PEER STOP command to shut down the MSMF interface on each AF/OPERATOR in your configuration.

**PEER STOP**

For a complete description and syntax of the PEER STOP command, see the *AF/OPERATOR Command Reference Manual*.

![PEER STOP command syntax diagram]

- **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.

- **IMMEDIATE**
  - Terminates MSMF after all active messages are sent and before new active messages arrive.

- **FORCE**
  - Deactivates the MSMF interface and any AF/OPERATOR operations using the interface.

- **TIMEOUT(sec)**
  - Overrides the default timeout value of 60 seconds.

**When to override PEER STOP command defaults**

<table>
<thead>
<tr>
<th>IF...</th>
<th>THEN use...</th>
</tr>
</thead>
<tbody>
<tr>
<td>you cannot shut down AF/OPERATOR because of in-progress PEERLINK activities</td>
<td>the FORCE option</td>
</tr>
<tr>
<td>you need more or less than the default timeout period of 60 seconds to allow peer-to-peer messages to drain</td>
<td>the TIMEOUT(sec) option</td>
</tr>
</tbody>
</table>
When PEER STOP is unnecessary

You do not always have to issue the PEER STOP command in order to stop the MSMF interface:

- If peer-to-peer communication terminates due to network problems, the PEER STOP command is issued automatically.
- When AF/OPERATOR is shutdown, the AF STOP command always issues a PEER STOP command.

Procedure

Issue the PEER STOP command to deactivate the MSMF interface on the local system:

PEER STOP

Result: When the console displays the message

PEER SUBTASK AND LOCAL LU TERMINATED

the MSMF interface is deactivated.

Shutdown process

The default MSMF interface shutdown process is:

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Matches executing outstanding WTORs are cancelled.</td>
</tr>
<tr>
<td>2</td>
<td>In-progress work is allowed to complete until the timeout period expires (default value: 60 seconds).</td>
</tr>
<tr>
<td>3</td>
<td>Once the timeout period expires, outstanding WTOs and commands are cancelled.</td>
</tr>
<tr>
<td>4</td>
<td>The VTAM access control block for AF/OPERATOR closes. The MSMF interface is deactivated, but any AF/OPERATOR operations that were using the interface remain active.</td>
</tr>
</tbody>
</table>
Defining Links Between Systems

Introduction
When you define a peer-to-peer link using the PEERLINK DEFINE command, you provide AF/OPERATOR with information about the target system. AF/OPERATOR uses this information:

- to create the session startup procedure you will use to start the peer-to-peer link
- to set up the data queues AF/OPERATOR will use to communicate with its peer on that system

What you need to know
To define peer links to other systems, you will need the following information from the person who configured AF/OPERATOR on the target system:

- LINKID of the target AF/OPERATOR
- application ID (LU name) that defines AF/OPERATOR to VTAM on the target system
- VTAM logmode table entry that defines the desired session parameters and characteristics on the target system

PEERLINK DEFINE
For a complete description and syntax of the PEERLINK DEFINE command, see the AF/OPERATOR Command Reference Manual.
Defining Links Between Systems

When to override the VTAM Logon Mode definition

If the default logmode table entry for the target system defines information inappropriate for communication with the local system, use the MODE(modename) option to override the default definition and specify the correct VTAM definition.

Procedure

Issue the PEERLINK DEFINE command to define a communication link to another system:

```
PEERLINK DEFINE LINKID(linkid) APPLID(luname)
```

```
MODE(modename)
```

Result: When the console displays the message

```
PEER LINK linkid DEFINITION COMPLETE FOR APPL = applid
```

the link is defined.

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 of the target AF/OPERATOR.

APPLID(luname)  The application ID that defines AF/OPERATOR to VTAM on the target system.

MODE(modename)  The name of the VTAM logmode table entry definition to be used by the target system. If you do not specify modename here, AF/OPERATOR uses the one specified when the PEER START command was issued.
Starting Links Between AF/OPERATOR Systems

Guidelines
When you start a peer-to-peer link using the PEERLINK START command, you activate the session startup procedure created when you defined the link. Keep the following two points in mind:

- You cannot start an undefined peer-to-peer link. Therefore, you must always define a link with the PEERLINK DEFINE command before you can use the PEERLINK START command to start it.
- Once you start a communication link between two AF/OPERATOR address spaces, it remains active until you terminate the session using the PEERLINK STOP command (described on page 304).

What you need to know
To start a peer link to another system, you need the LINKID of the target AF/OPERATOR from the person who configured that AF/OPERATOR. (This identifier may have been redefined during AF/OPERATOR customization as something other than the SMFID.)

PEERLINK START
For a complete description and syntax of the PEERLINK START command, see the AF/OPERATOR Command Reference Manual.

```
subsys

PEERLINK START LINKID(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.

LINKID(linkid)
The LINKID of the target AF/OPERATOR.
Procedure

Issue the PEERLINK START command to start a defined communication link to another system:

    PEERLINK START LINKID(linkid)

Result: When the console displays the message

PEER LINK linkid STARTED SUCCESSFULLY

the MSMF interface has initiated a session between the local AF/OPERATOR and the AF/OPERATOR on the target system. You can begin to exchange data between the two systems.
Stopping Links Between Systems

Introduction

The PEERLINK STOP command terminates the communication session with the specified system.

Process

When you stop peer-to-peer communication, the following process occurs:

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The communication session terminates at the completion of the next transaction to be processed through the specified link.</td>
</tr>
<tr>
<td>2</td>
<td>All system resources involved in peer-to-peer communication are released.</td>
</tr>
</tbody>
</table>

Note: You must terminate a peer-to-peer session using the PEERLINK STOP command before you can delete the link definition.

PEERLINK STOP

For a complete description and syntax of the PEERLINK STOP command, see the AF/OPERATOR Command Reference Manual.

```
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 of the target AF/OPERATOR.

FORCE

Terminates all active units of work currently using the link to the specified system.
Using the FORCE option

Use the FORCE option if the link must be terminated immediately (for example, when you cannot shut down AF/OPERATOR because of in-progress PEERLINK activities).

Procedure

Issue the PEERLINK STOP command to stop peer-to-peer communication with a specified system:

```
PEERLINK STOP LINKID(linkid)
```

**Result:** When the console displays the messages

**PEERLINK STOP COMMAND ACCEPTED FOR linkid**

**PEERLINK STOP COMMAND COMPLETED SUCCESSFULLY, linkid NOW INACTIVE:**

the communication session terminates.
Deleting Link Definitions

Introduction

When you delete a peer-to-peer link, you erase its session startup procedure and all corresponding internal communication support programs for that link.

**Note:** You cannot delete an active link. You must issue the `PEERLINK STOP` command to deactivate the link before you can erase it.

**PEERLINK DELETE**

For a complete description and syntax of the `PEERLINK DELETE` command, see the AF/OPERATOR Command Reference Manual.

```
PEERLINK DELETE LINKID(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.

**LINKID(linkid)**
The LINKID of the target AF/OPERATOR.

**Procedure**

Issue the `PEERLINK DELETE` command to delete a peer-to-peer link to a specified system:

```
PEERLINK DELETE LINKID(linkid)
```

**Result:** When the console displays the message

```
PEERLINK REQUESTED LINK ENTRY linkid DELETED SUCCESSFULLY
```

the peer-to-peer link definition has been erased.
Displaying Link Information

Introduction

Use the PEERLINK DISPLAY command to show information on specified peer-to-peer links. You can display the following information for each defined peer-to-peer link:

- current session status (active or inactive)
- number of transactions the specified system has sent or received during the current session

PEERLINK DISPLAY

For a complete description and syntax of the PEERLINK DISPLAY command, see the AF/OPERATOR Command Reference Manual.

![Diagram of PEERLINK DISPLAY command syntax]

<table>
<thead>
<tr>
<th>subsys</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINKID(linkid)</td>
<td>The LINKID of the target AF/OPERATOR.</td>
</tr>
<tr>
<td>LINKID(ALL)</td>
<td>Displays all defined (active and inactive) links. (This is equivalent to the * wildcard.)</td>
</tr>
<tr>
<td>ACTIVE</td>
<td>Displays only active links.</td>
</tr>
<tr>
<td>INACTIVE</td>
<td>Displays only inactive links.</td>
</tr>
</tbody>
</table>

Procedure

Issue the following command to show the status and transaction counts of peer-to-peer links:

```
PEERLINK DISPLAY LINKID(linkid or ALL)
```

Result: The console displays messages similar to the following:

```
PEER LINK linkid SEND COUNT(nnmnn), RECEIVE COUNT(nnmnn), STATUS(ACTIVE)
```
Sending WTOs to a Target System

Introduction
This section describes how to use the WTO command to send WTO messages to a target system by a peer-to-peer communication link.

What you need to know
To send a WTO message to a target system, you must know the LINKID of the target system as specified in its link definition.

WTO
For a complete description and syntax of the WTO command, see the AF/OPERATOR Command Reference Manual.

```
subsys  WTO 'message' 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 1 to 126 characters in length. You can use continuation signs but not in column 72.

SYSID(linkid)
The LINKID of the target AF/OPERATOR to which you are sending the WTO.

Note: Other options on the WTO command may be used to modify the WTO message or how it is delivered. See the AF/OPERATOR Command Reference Manual for a complete description of the WTO command and its options.

OSYSNAME startup parameter
The AF/OPERATOR startup parameter OSYSNAME determines whether event information from the originating system is retained in global match variables.

If set to Y, the SYSID, job name, job type, and job ID respectively are stored in the global match variables AOSID, AOJNAME, AOJTYPE, and AOJID.

If set to N, AOSID, AOJNAME, AOJTYPE, and AOJID reflect the values of the system specified in the LINKID parameter.

See the AF/OPERATOR Configuration and Customization Guide for more information.
Example

If System B issues the command:

```
WTO 'ATTENTION SYSTEM A' SYSID(SYSA)
```

the message **ATTENTION SYSTEM A** appears on the System A console.
Issuing Commands to a Target System

Introduction
You can issue OPER commands to any target system that has a defined and started communication link to the local system. The OPER command’s SYSID option directs the command in single quotes to the specified system.

Prerequisites
The subsystem consoles must be defined with AUTH(ALL) in SYS1.PARMLIB (required if you want response text from the command returned).

Before you issue commands to a target system, you need the following information from the person who customized AF/OPERATOR:
- the LINKID of the target AF/OPERATOR as specified in its link definition
- whether AF/OPERATOR can trap on responses from the target system (the TRAPRESP startup option must be set to Y)

OPER
For a complete description and syntax of the OPER command, see the AF/OPERATOR Command Reference Manual.

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)
Using Commands to a Target System

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.

**Using a TIMEOUT value**

The AF/OPERATOR startup parameter OPERRESP(MAXTVAL) serves as the default timeout for single-line WTO responses and for MLWTOs with no end marker. The default value is 5 seconds. You can override the default by coding a different TIMEOUT value as part of the OPER RESP command.

Using the TIMEOUT keyword also affects how AF/OPERATOR handles an MLWTO response:

- If you do not code a TIMEOUT value, an OPER RESP for an MLWTO response ends when the end marker is received.
- If you do code a TIMEOUT value, the MLWTO end marker is ignored.

(See the **AF/OPERATOR Configuration and Customization Guide** for a complete description of the OPERRESP(MAXTVAL) parameter.)
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 .............................................. 314
  – 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 ....................................... 318
  – Assigned Support Center Representative (ASCR)
  – Maintenance Assessment Services (MAS)
  – Multi-Services Manager (MSM)

Customer Support Contact Information ....................... 320
  – 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, using the World Wide Web at 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.

- **Severity 1 Crisis**
  
  A crisis affects your ability to conduct business, and no procedural workaround exists. The system or application may be down.

- **Severity 2 High**
  
  A high-impact problem indicates significant business effect to you. The program is usable but severely limited.

- **Severity 3 Moderate**
  
  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.

- **Severity 4 Low**
  
  A low-impact problem is a “how-to” or an advisory question.

- **Severity 5 Enhancement Request**
  
  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.
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</td>
<td>90% within one</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Answer</td>
<td>minute</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 1 Response</td>
<td>90% within 5</td>
<td>90% within one</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Normal Business</td>
<td>minutes</td>
<td>hour</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hours)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 2 Response</td>
<td>Warm Transfer</td>
<td>90% within two</td>
<td>90% within eight hours</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Normal Business</td>
<td></td>
<td>hours</td>
<td>hours</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hours)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scheduled follow-up</td>
<td>Hourly or as</td>
<td>Daily or as</td>
<td>Weekly or as</td>
<td>Notification is made when an enhancement is incorporated into a generally available product.</td>
<td></td>
</tr>
<tr>
<td>(status update)</td>
<td>agreed</td>
<td>as agreed</td>
<td>as agreed</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.
This glossary defines words and phrases that have special meanings in the context of AF/OPERATOR or within the systems interacting with AF/OPERATOR. Special characters are alphabetized in this glossary as follows:

< > See ANGLE BRACKET.
& See AMPERSAND.
&& See DOUBLE AMPERSAND.
* See ASTERISK.
{} See BRACES.
[] See BRACKETS.
$ See DOLLAR SIGN.
^ See ESCAPE SIGN.
= See EQUAL SIGN.
- See MINUS SIGN.
% See MODULO.
| See OR BAR.
# See POUND SIGN.
? See QUESTION MARK.
. See QUOTE.

ABEND Abnormal end of a task due to an error.

ACB See ACCESS METHOD CONTROL BLOCK.

ACB NAME The name specified in the acbname parameter of the VTAM APPL statement. This name permits multiple copies of the same application program in a network to specify the same APPLID name.

ACCESS METHOD CONTROL BLOCK (ACB) A control block that links an application program to VTAM.

ACF/VTAM™ Advanced Communication Function for the Virtual Telecommunications Access Method. Synonym for VTAM.

ACTION ROUTINE A user-supplied REXX program executed to resolve a problem or react to a set of conditions.

ACTIVE MATCH A match that has not finished executing.

ALPHABETIC CHARACTER One of A-Z or a-z.

ALPHANUMERIC CHARACTER One of A-Z, a-z, or 0-9.

AMPERSAND (&) An ampersand is prefixed to a variable name in symbolic substitution. See also DOUBLE AMPERSAND.

ANGLE BRACKET (< >) In a conventional on-screen command syntax description, <A> means that element A is optional.
APPLICATION PROGRAM IDENTIFICATION (APPLID)  The symbolic name that identifies an application program to VTAM. It is defined in the acbname parameter of the APPL statement.

APPLICATION PROGRAM INTERFACE (API)  The programming language interface between a program and its user.

ARITHMETIC FUNCTION  Either the single function represented by a minus sign (-) that changes the sign of a number or one of the binary functions represented by +, -, *, /, %, &&, or |.

ARITHMETIC OPERATOR  Takes one or two expressions as input and returns the result of an arithmetic operation.

ASSIGNMENT  You can assign values to user-defined variables with the DCL, LET, and PROC commands. Values can be user-assigned to some system variables and some match variables.

ASTERISK (*)  (1) A pattern character that matches any string.  (2) As an arithmetic operator: Multiplies expressions.  (3) In an expression: The sign for multiplication.  (4) In certain commands: In some cases, using the asterisk as the only character in a keyword subfield is equivalent to not using the keyword.  (5) In TOD commands: Represents the current (system) time.

AUTOREFRESH OPTION  The option invoked by AF AUTOREF.

AVAILABILITY MANAGEMENT SYSTEMS (AMS)  An Availability Management System is an OMEGACENTER implementation designed using Candle’s methodology for managing systems from a business orientation.

BIND  A request to activate a session between two logical units (LUs) in an SNA network. See also SNA NETWORK.

BLANK  (1) A parsing device that divides the target string (between “words”).  (2) A pattern character that matches one or more blanks.

BOOLEAN  Operates on one or two comparisons to make a condition.  One of the following: AND, OR, or NOT.

BOUNDARY FUNCTION  Capability in an SNA sub-area node that provides protocol support for adjacent peripheral nodes.

BRACES ({ })  In a conventional command syntax description, {A} means that element A is required. Note that an unbracketed element is also required.

BRACKETS ([ ])  In a conventional printed command syntax description, [A] means that element A is optional.

CALL A FILE  Sometimes used to mean executing a command file with the EXEC command. Note that “execute a (command) file” is a more precise usage.

CALL A PROGRAM  To run a program by issuing the CALL command from a command file.

CFILE  See COMMAND FILE.

CHARACTER STRING  A concatenation of printable characters. A character string is one kind of constant in AF/OPERATOR command syntax. It must be enclosed in quotes unless it contains only alphanumeric characters.

CHARACTER TYPE  A variable of character type contains as its value a string of printable characters (including spaces).

COMMAND FILE  A file consisting of commands written in the AF/OPERATOR
command file language that can be executed by an EXEC command.

**COMMENT** In a command file or REXX exec, text used for documentation that is marked off so that it plays no part in the file execution. /* Comments are marked off like this. */

**COMPARATOR** One of six abbreviations (EQ NE GT LT GE LE) that (1) expresses a quantitative comparison, or (2) stands between expressions in a comparison.

**COMPARISON** A clause in a condition statement of the form: `expression comparator expression`. For example: `COUNT GE 2`.

**COMPILER** The part of the AF/OPERATOR program that interprets and executes AF/OPERATOR commands (As used in explanations of symbolic substitution and using quotes.)

**CONDITION (OR CONDITION STATEMENT)** Part of a conditional command that consists of a comparison or a Boolean combination of comparisons. For example, `VAR1 LT 1` or `VAR1 GT 1`.

**CONDITIONAL COMMAND** A type of command that takes action depending on whether a given condition statement is true or false. A conditional command contains one of the following structures: DO UNTIL, DO WHILE END, IF THEN ELSE.

**CONSOLE COMMAND** A command that can be issued from the console.

**CONSOLE ID** Each operator console is known to the operating system by a unique console ID, a number from 1 to 255.

**CONSTANT** A numeric integer coded in decimal, hexadecimal, or binary, or a character string enclosed in quotes.

**CONVERSATION** Term used in LU 6.2 APPC describing a temporary connection between two application programs that allows them to communicate with each other.

**CROSS-SYSTEM ESCALATION** In AF/OPERATOR, the ability to route all console messages in a network to one designated site.

**D**

**DATA MANAGER** A program that controls access to a data source. For example, it can be used to reduce the number of data requests.

**DARK OPERATIONS** Synonym for unattended remote operations.

**DEFINITION STATEMENT** In VTAM, the means of describing a network element.

**DELETE A FILE** Erase a file from DASD

**DELETE A TRAP** Deleting a trap deactivates it and erases its definition.

**DELETE A VARIABLE** Erase the definition of a variable.

**DESCRIPTOR CODE** A numeric code that sets a certain attribute of a console display.

**DIAGNOSIS ROUTINE** A user-supplied REXX program executed by the Probe Manager when a managed object’s attribute status is Warning or Critical. A diagnosis routine may collect data, update OMEGAVIEW status items, send requests to the operator, or perform other functions.

**DISABLE A TRAP** Disabling a trap deactivates it but leaves its definition in storage.

**DISTRIBUTED SYSTEM ID** The identifier associated with a distributed system. It is identical to the domain ID.

**DIVISION (/) ARITHMETIC OPERATOR** Divides the first expression by the second.

**DOLLAR SIGN ($)** An initial dollar sign is used to form the name of an AF/OPERATOR function; for example, `$DAYOFWEEK`. 
DOMAIN  The MVS image on which a managed object resides.

DOMAIN ID  The identifier associated with a domain.

DOUBLE AMPERSAND (&&)  A double ampersand (&&) in an arithmetic function indicates “anding” of numeric values. When used in the ACTION or ALTACT portion of a trap, indicates that the named variable is evaluated when the trap fires, instead of when it is added to the system.

ENABLE A TRAP  Enabling a trap activates it, making it fully functional.

END USER  The originator or recipient of application information flowing through an SNA network. An end user can be an application program or a terminal user.

ENVIRONMENT ID  The 8-character OMEGAMON application ID used by the Data Managers and probes.

EQUAL SIGN (=)  Assigns a value to a variable.

ESCAPE SIGN (^)  If the escape sign occurs just before an instance of a special pattern character, that instance is not counted as a pattern character, but as an ordinary alphanumeric character.

EVENT PROBE  A probe that is executed as the result of a system event (for example, an alert, batch job exception, or failure).

EXCEPTION LIST  An OMEGAVIEW panel, included with OSM/MVS, which lists system exception conditions.

EXPRESSION  Either a constant or a variable or part of a command that resolves to one of these. Examples are 2, VAR1, VAR1 + 2, $HHMMSS(VAR1 + 2).

FOLLOW-UP ROUTINE  A user-supplied REXX program, executed by OSM/MVS in response to a change in the status of a managed object, that determines if a problem has been successfully resolved.

FUNCTION  Part of a command that takes one or more arguments (expressions) as input and returns a value as output.

GLOBAL FILE  A file that can be referenced from any AF/OPERATOR file.

GLOBAL VARIABLE  A variable that can be referenced from any AF/OPERATOR file.

GROUP COMMAND  A series of commands between DO and END that AF/OPERATOR treats syntactically as one command.

HEARTBEAT PROBE  A probe that determines if a system resource is active and/or available.

HEXADECIMAL TYPE  A variable of hexadecimal type contains as its value a hexadecimal number.

HEXADECIMAL NUMBER  A number expressed in base 16.

INITIALIZATION PROBE  A program module that is run once during the OSM/MVS start-up phase. It may be a Query, Heartbeat, or an Event probe, or a utility function.

INTEGER  A positive or negative whole number.
**INTEGER TYPE** A variable of integer type contains as its value an unsigned integer.

**K**

**KEYWORD** A word in a command with spelling and meaning specified by AF/OPERATOR command syntax. For example, ADD in TRAP ADD.

**L**

**LINE CONTINUATION** A command in a command file may be written so as to continue over more than one physical line by using one or more plus signs as continuation characters.

**LINE VARIABLE** A variable with a value of (1) The number of lines in a multiline screen display, (LINE#), or (2) The text of a line in a multiline screen display (LINE1, etc.).

**LINKID** The ID that identifies a target AF/OPERATOR in AF/OPERATOR Multi-System Management Facility (Peer-to-Peer) PEER commands.

**LOCAL FILE** A file that can be referenced only by a file that it directly executes.

**LOCAL VARIABLE** A variable that can be referenced only within the file that creates it.

**LOGICAL UNIT (LU)** A port or entry point into an SNA network through which an end user accesses network services and communicates with other network users.

**LOGON** An initiation request for a session between logical units in an SNA network.

**LOGON MODE** A set of session parameters entered in a VTAM Logon Mode table that specify LU-LU session requirements.

**LOGON MODE TABLE** A VTAM table that contains sets of entries for each logon mode. A logon mode name (MODEENT) identifies each logon mode.

**LU** Logical Unit.

**LU-LU SESSION** A temporary connection between two logical units that is the vehicle for end-user communication in an SNA network.

**LU 6.2** The SNA logical unit type implemented in IBM’s Advanced Program-to-Program Communication (APPC) facility that provides a standard protocol for program-to-program communication in a distributed data processing environment.

**M**

**MANAGED OBJECT** One or more objects monitored by an OSM/MVS probe and represented on an OMEGAVIEW display by situation or attribute status lights. For example, a Payroll Department, On-line Services, or Corporate LAN. Managed objects can also include lower levels of detail, such as TSO Users in Performance Group 210, Run Times for a set of batch jobs, and ABEND conditions for a set of batch jobs.

**MATCH** The execution of a command, REXX exec, or series of REXX execs that results when a trap is triggered.

**MATCH VARIABLE** A global variable that is defined when a match occurs.

**MINUS SIGN (-)** In an expression, the minus sign represents either the unary (change-of-sign) or the binary (subtraction) function.

**MODEENT** A parameter in a VTAM APPLID definition statement that defines an entry in a logon mode table containing session parameters for a given application.

**MODETAB** A parameter in a VTAM APPLID definition statement that defines the name of the logon mode table to be used for a given application.

**MODULO (%)** Returns remainder from dividing the first expression by the second.
MULTI-SYSTEM MANAGEMENT FACILITY (MSMF) A Candle facility that performs direct (peer-to-peer) communication between AF/OPERATOR systems.

NAME In command syntax a name consists of up to eight alphanumeric (including national) characters and begins with an alphabetic character.

NAME FIELD A subfield in an operand set that must be occupied by a name.

NETWORK ADDRESSABLE UNIT (NAU) In an SNA network, the logical unit, physical unit, or system services control point that is the origin or destination of information transmitted by the SNA path control network. Each NAU has a network address that represents it to the path control network.

NETWORK NAME The name of the APPL statement that defines a VTAM application program.

NODE In VTAM, a physical point in a network defined by a symbolic name. In SNA, a physical point in a network that contains one or more network components.

NOTIFICATION FACILITY The OSM/MVS facility that automatically informs individuals of a status change, problem, event, etc.

NOTIFICATION LIST A list of names, methods, and times to be used in the notification process.

NUMERIC Numeric characters are 0-9.

OMEGACENTER Gateway for MVS (OG/MVS) A Candle product that provides two-way communication between Candle’s OMEGACENTER products and other products, such as console automation products, IBM NetView, and ASCII hosts in a distributed systems environment.

OMEGACENTER Status Manager for MVS (OSM/MVS) A Candle product that provides a set of tools and interfaces designed to help you build availability management applications.

OMEGAVIEW Candle’s status display product. It displays concise information about and provides access to all of Candle’s performance monitoring products.

OSM/MVS ASSISTANT An OSM/MVS facility that ensures that all necessary parameters have been entered for a managed object.

OPERAND A unit of a command other than the command verb.

OPERAND SET An operand possibly followed by a subfield.

OPERATOR COMMAND An operator command is a command entered from an operator console to monitor and control jobs, devices, and MVS itself.

OPERATOR CONSOLE An operator console is a computer terminal specifically assigned for use by computer operators to monitor operator messages and issue operator commands.

OPERATOR, ARITHMETIC See ARITHMETIC OPERATOR. OR BAR (|)
In an expression, an or bar means the logical “oring” of two numbers. In a command syntax description, A | B | C means to choose only one of the set ABC.

OBJECT A system resource or workload.

OG/MVS FUNCTION Function supplied by Candle as part of AF/OPERATOR.
PATTERN  A string of text that can contain special characters that function as wild-cards in a trap pattern.

PATTERN CHARACTER  A character that can be used in a text pattern as a wild-card character for string matching and to define a trap pattern variable. The pattern characters are ?, #, and *.

PATTERN VARIABLE  A variable named and defined with the use of a special character in a trap pattern.

PEER-TO-PEER  See MULTI-SYSTEM MANAGEMENT FACILITY.

PEER-TO-PEER COMMUNICATION  Direct program-to-program (end user) communication in a telecommunications network.

PERIOD (.) (1) A pattern character that terminates a pattern variable. (2) A parsing device that skips a word (a string between blanks).

PLUS SIGN (+) (1) The line continuation character in command files. (2) Adds expressions.

POSITIONAL OPERAND  An operand required by AF/OPERATOR syntax to occur in a certain place in a command relative to other operands.

POUND SIGN (#)  A pattern character that matches any numeric character in a full pattern.

PROBE  A program written in REXX that collects data about a managed object. See HEARTBEAT PROBE, QUERY PROBE, and EVENT PROBE.

PROBE CONTROL PROGRAM  A user-defined REXX routine that performs preprocessing tasks for a probe. Every event probe requires a Probe Control Program.

PROBE ID  The identifier associated with a probe.

PROBE MANAGER  A system driver module that controls the execution of a probe and, through the Status Manager, determines a status, and updates OMEGAVIEW.

PROTOCOL  The formal set of rules that control format, timing, sequencing, and error handling of information exchanged between users or network components in a communications network.

QUERY PROBE  A probe that collects measurement values which represent the current attributes of a managed object. See MANAGED OBJECT.

QUESTION MARK (?)  A special pattern character that matches any alphanumeric character in a full pattern.

QUOTE (')  The single quote character (') when referring to quoted strings.

QUOTED STRING  A parsing device that divides at the string.

ROLLUP (ROLLUP STATUS BAR)  A graphic representation on an OMEGAVIEW display of an aggregate of basic status items or of both basic and rollup status items.

RULE  An equation used in situation definitions. It consists of an object, a comparator, and a value. The object is the status item for either a managed object or a situation. The comparator is one of the standard set of mathematical tests (equal to, greater than, less than, not equal to, etc.). The value is a status value (OK, Warning, Critical, or None). See SITUATION RULE SET.

RULE SET  See SITUATION RULE SET.

ROUTE CODE  A numeric code that directs a WTOR to a certain console or consoles.
S

SAA  Systems Applications Architecture™. The IBM facility that defines the logical structure, format, and operational sequences of applications.

SCOPE OF A FILE  A region from which the file can be referred to (local, global, or system).

SCOPE OF A VARIABLE  A region from which the variable can be referred to (local, global, or system).

SDM  See STATUS DATA MANAGER DATABASE.

SESSION  A controlled exchange of information between two users in an SNA network.

SESSION PARAMETERS  The entries that define session requirements (such as pacing) for a session between two network addressable units.

SIGNED INTEGER +/- N  When used as a parsing device, moves the parsing pointer nn positions forward or back.

SIMPLE COMMAND  A command that performs an immediate action. A command that is not a conditional command. See CONDITIONAL COMMAND.

SITUATION  See SITUATION OBJECT.

SITUATION OBJECT  A named, composite attribute consisting of up to five rules combined with AND or OR. All situations evaluate to either True or False.

SITUATION MANAGER  The OSM/MVS component that evaluates the rule set and determines the status settings of situation objects.

SITUATION RULE SET  The set of rules defining the analytical logic used to determine a situation’s status. Each rule of the set defines a formula involving other status items, operators, and attribute status or values. A Rule Set may contain up to five rules. See RULE.

SITUATION STATUS  A setting assigned by the Status Manager. Situations are only evaluated in True/False terms, so a situation status can only have settings of OK (green) or critical (red).

SNA  Systems Network Architecture. The IBM facility that defines the logical structure, formats, protocols, and operational sequences that must be adhered to in controlling the configuration, operation, and information exchange in an SNA network.

SNA NETWORK  Portion of a distributed data processing network adhering to the format and protocols of the IBM Systems Network Architecture. It consists of the network addressable units (NAUs), boundary function components, and the path control network.

SPIKE SMOOTHING  A user-selected option in Status Manager that performs a statistical normalization of statuses. It provides a more representative picture of attributes whose statuses vary greatly over short periods of time.

SSCP  System Services Control Point.

STATUS BAR  A graphic representation on an OMEGAVIEW display of the health of some aspect of a system’s performance, such as response time or availability. The status bar’s color corresponds to a status: Green (OK), Yellow (Warning), Red (Critical), and Blue (None). See also BASIC STATUS BAR and ROLLUP STATUS BAR.

STATUS DATA MANAGER DATABASE (SDM)  The OMEGAVIEW database that stores and maintains the status items displayed on OMEGAVIEW panels. OSM/MVS or AF/OPERATOR can update the SDM.

STATUS ITEM  A system-wide identifier, defined by the user, that associates a managed object attribute or a situation to an OMEGAVIEW status bar.
**STATUS MANAGER**  The OSM/MVS component that compares an attribute value to a set of thresholds and assigns a status (OK, Warning, or Critical). The Status Manager is an integral part of the Probe Manager.

**SYMBOLIC VARIABLE**  A variable preceded by one or more ampersands for symbolic substitution.

**SYSID**  In AF/OPERATOR MSMF operations, the name that identifies the target AF/OPERATOR in WTO and OPER commands.

**SYSTEM FILE**  A file that can be referenced by any AF/OPERATOR file.

**SYSTEM ID**  An identifier representing the MVS image on which a probe executes. To be distinguished from the domain ID, which is the MVS image the probe monitors. The domain and the system may be identical.

**SYSTEM SERVICES CONTROL POINT (SSCP)**  The focal point in an SNA network that functions as the network configurator and manager. The SSCP resides in a host processor.

**SYSTEM VARIABLE**  A variable that can be referenced from any AF/OPERATOR file.

**THRESHOLD**  The user-specified value against which the current attribute value is compared to derive a status.

**TRANSLATE ROUTINE**  See XLATE ROUTINE.

**TRAP**  A user-defined rule statement which defines how AF/OPERATOR responds to specific system events such as messages, MVS commands, OMEGAMON exceptions, and date and time information.

**TSO TERMINAL**  A computer terminal used to access the TSO time sharing system.

**U**

**UNBIND**  The SNA term that describes a request to deactivate a session (connection) between two logical units in a network.

**UNSIGNED INTEGER N**  When used as a parsing device, it divides the source string at the n'th column.

**V**

**VARIABLE**  An element in the AF/OPERATOR command language that contains a value and can contain another value at another time.

**VARIABLE LENGTH**  A variable whose length is independent of the length of other related variables. A variable of “variable length” is a character-type variable with the default maximum length of 255 characters.

**VTAM**  (Virtual Telecommunications Access Method) The IBM program product that controls communication and the flow of data in an SNA network. See also ACF/VTAM.

**W**

**WORK LIST**  An OMEGAVIEW panel included with OSM/MVS which lists system problem areas requiring operator attention.

**WORKING TRAP**  An enabled trap in storage.
accessing
OG*TSO  84
action
issue a single command
execute a REXX exec  65
specifying  65
ACTION field of a trap
described  65
example of  68
using quotes in  66
using single quotes in  68
ACTION keyword
ampersand  63
double ampersand  63
variables  63
actions
defined  38
repeating at regular intervals  47
specifying trap responses  65
Adobe portable document format  11
advanced TCP/IP communications  124
AF TCP/IP communications protocol  124
automated link management  148
creating a client application  129
open TCP/IP communications  146
AF communications protocol
TCP/IP  124
AF TCP/IP communications protocol  124
Candle AF packet format  124
data types  125
EXEC  127
ID  125
response  127
response data types  127
ID data type  125
ID record keywords  125
linkid or message token  125
AF/OPERATOR
client  98
connecting AF/OPERATOR systems  101
connecting to AF/REMOTE (Windows/NT)  106
defined  18
message log  67
operation  18
server  98
AF/OPERATOR commands  209, 210
invoking from NetView  209
issuing console commands  22
AF/OPERATOR systems, connecting  101
AF/REMOTE
returning data to  113
sending data to
example  113
AF/REMOTE (Windows/NT)
defining a link to  106
AFADDR, OMEGAMON session  195
alert
See NetView alerts
allowed message types  128
ampersand  63
ACTION field  63
traps in REXX execs
variables  63
an extended action
ACTION keyword  68
AOCMD()  203, 209, 211
AOI
See IMS automated operator interface (AOI)
AONVCLST  204
AONVICMD  204
AONVICTM  204
AORC  213
AOSIM facility
CICS Interface use of  253
definition  269
used with NetView exit DSIEX01  220
APPC
configurations requirements  116
APPC/MVS configuration  116
APPL statements  116
applids
See LU 6.2 applids  116
APPLIDs, POVI  154
ASCII connections  141
ASCR
assigned support center representative  318
asterisk
See wildcard patterns
wildcard character  57, 68
at time interval
TOD trap example  48
automated LINK management  148

B

Basic Operator, POVI
  definition
    APPLID 154
  using 175
blanks in a full pattern 58
Break key, POVI
  definition
    usage 166
    selection 165
    verification 166

C

Candle AF packet format 124
catch-up processing 47
change a trap definition 70
CICS Interface 254, 255, 256
  description of 253
  exit 255
  overview 251
CICS Interface exit
  disabling from blank CICS screen 256
  disabling from MVS console 256
  disabling from user program 256
  enabling automatically 255
  enabling from blank CICS screen 256
  enabling from MVS console 256
  enabling from user program 256
CICS messages 255, 258
  selection criteria 254
  trapping 258
  trapping and processing 255
CICS statistics 255
CID(OG*TSO) 89
client AF/OPERATOR 94, 98
client application, TCP/IP
  creating 129
CMD
  AF/OPERATOR response 45
  cautions 46
  logic 45
  selection criteria 45
CMD trap
  cautions 46
  defined 37
  described 45
  example of 46
  logic 45
CMD traps 45
cold starting AF/OPERATOR 32
color changes 245
column information 248
COMDATA.1
  parsing 1-line substrings 111
command files
  cross referencing 74
COMMAND parameter 32
command response
  suppress or modify
    execute AF/OPERATOR command 45
commands 45, 254
  See commands
  See IMS commands
  See NetView console commands
  trapping
  communicating with
    AFADDR 195
to general 191–197
    LOGOFF 193
  communicating with AF/REMOTE 106
  communicating with OMEGAMON
to general 191
  communication commands 95
  communication services
    features of 95
  communication trace facility 96
  communications env 93
  communications environment
    overview 94
  communications match 97
  COMSDRCV function
    using, examples 109
configuration requirements
  for APPC 116
  for TCP/IP 122
  for XCF 120
configuring OG*TSO 85
CONFIRM session 118
connecting AF/OPERATOR systems 101
console commands, issuing 22
creating a TCP/IP client application 129
cross reference reports
  error report 80
  printing 81
  types 75–79
Cross-Reference Facility 72
cross-reference facility 74–81
cross-referencing traps and command files 74
cross-referencing traps and REXX execs 72
CUA Operator, POVI
  definition
    APPLID 154
  using 175
customer support
  base maintenance plan 314
  contact information 320
enhanced support services 318

eSupport 315
severity levels 315
telephone support 314

dangling conversation 114
DATA 192
data retrieval
  through NetView Interface 211
data types
  DATARRERR 127
  DATARPLY 127
  EXEC 127
  ID 125
  response 127
  used by AF TCP/IP communications protocol 125
data, outbound
  EBCDIC 107
DATARRERR data type 127
DATARPLY data type 127
datasets
  TKANMOD
    RKANCMD 81
date
  in TOD trap 47
DCT
  See destination control table (DCT)
defining
  null rules 239
  status rules 238
  threshold rules 239
  variables 61
deleting
  trap definitions 70
destination control table (DCT) 253
destination ID
  selecting CICS messages by 254
developing
  patterns from software documentation 62
diagnostics, POVI 180
disabling 69
avantages 69

disabling traps 69
displaying
  trap definitions 71
  trap matches 71
double
  in ACTION field 63
  traps in REXX execs 63
  variables 63
double ampersand
  in ACTION field 63

dynamic rollup 240

EBCDIC format 107
editing a POVI script 170–171
elements of
  action 38
  control options 38
  ENABLE keyword 37
  extended action
    ACTION keyword 38
  immediate action 38
  name 37
  triggering condition 37
  triggering conditions 37
  type 37
enabling traps 69
environment commands
  K 185
  LOC 185
  RC 185
  S 185
  SEE 186
  SET 169, 186
  STATUS 186
  TYPE 186
  WAIT 187
error messages 32
error recovery 114
eSupport
  customer support 315
examples 51, 68, 258
  AF/OPERATOR commands, invoking from
    NetView 210
  AF/OPERATOR data, retrieving through the
    NetView Interface 212
  NetView alerts, generating 225
  NetView commands, trapping 221
  NetView Interface status, displaying 221
  NetView VTAM messages, trapping 223
  OMEGAMON exception
    XOC 51
  OMEGAMON for MVS data, retrieving through the
    NetView Interface 215
TOD
  action at time interval 48
WTO
  message translation 50
exception trap
  described 51, 194
exception analysis 190
LOG option, trap command 67
log, AF/OPERATOR message 67
logging messages 67
LOGOFF
defined 193
LOGON
  appltype, VTAM 192
defined 191
  INTERVAL 192
  NAME 192
  OMEGAMON for CICS 192
  OMEGAMON for IMS
    OMEGAMON II for DB2 192
  PASSWORD 192
  USERID 192
  virtual screen size 193
  VTAM 192
logon 155, 157
  from REXX exec 157
  from VTAM terminal 155
  POVI 155–157
LOGON command 193
  DATA 192
  LTERM 193
LTERM 193
LU 6.2 applids
  NOSHED type 117
LU 6.2 applids
  APFL statements 116
  defining to APPC/MVS 117
  defining to VTAM 116
  SCHED type 118

M
maintenance assessment service
  MAS 318
MAS
  maintenance assessment service 318
match variables
  ampersands 50, 63
match variables, using 67
message 49, 50, 68, 254
  highlighting 49
  log 67
message prefix
  selecting CICS messages by 254
message types
  allowed 128
  use of 128
message types, POVI 180
messages
  error 32
  modifying

  general 70
  POVI scripts 170
  replacing traps 70
  TRAP CHANGE 70
  TRAP DELETE 70
  traps 70
monitored
  AF/OPERATOR IMS 45
  JES job log 45
  SVC 34 45
monitoring
  AF/OPERATOR IMS 49
  CICS 254
  JES job log 49
  SVC 35 49
MSM
  multi-services manager 318
MSMF INTEGRITY startup option 118
Multiple Address Space Support 23
  defined 21
  issuing console commands 22
  LINKIDs 24
  order of event processing 23
multi-services manager
  MSM 318
Multi-System Management Facility 294
MVS command, issuing 111
MVS dispatching priority
  recommended for AF/OPERATOR 48

N
name pattern
  caution using 60
  defined
    described 59
  naming examples
    XCF 121
  naming rules
    for XCF groups and members 120
NetView alerts 224, 225
  Configuring NetView to trap 218
  definition of
    via the Program-to-Program Interface 224
NetView commands 226
  invoking from AF/OPERATOR
    example of 226
NetView console commands 220, 221
NetView console VTAM messages
  trapping 222
NetView Interface 204, 205, 207, 208, 211, 212, 215
  advantages
    description of 199
commands
  See commands
description of 201
retrieving data through 211
REXX functions 203
starting
  procedure for 205
starting and stopping 205
stopping
  procedure for 206
trapping console traffic 202
variables
  See variables
NetView REXX execs 226
  invoking from AF/OPERATOR
  example of 226
NetView VTAM messages 222, 223
NetView alerts
generating 224
NOLOG option, trap command 67
NOSCHED type LU 6.2 applid 117
null rules 239
NVALERT 224
NVIDATA() 203, 211
NVISHOW 204, 207
NVISTART 204, 205
NVISTOP 204, 206

O

OG*TSO 89, 91
  accessing 84
  configuring 85
  exiting
    Main Menu 84
  restrictions 92
  testing traps 90, 91
  under TSO 83
  using
    simulated AF/OPERATOR 83
OMEGAMON 191, 193, 195, 197
  AFADDR 195
  exceptions 194
  LOGOFF 193
  session 195
OMEGAMON exception trap
defined 51
example of 51
exception analysis defined 190
XO2 51
XOC 51
XOI 51
XOM 51
OMEGAMON for MVS

retrieving data from 215
OMEGAVIEW
  automated session control feature 249
defined 18
OMEGAVIEW Interface 238, 239
  causing color changes 245
  controlling OMEGAVIEW sessions from
    OG/MVS 249
  naming conventions 238
  returning column information 248
  status items
    prerequisites for accessing 238
    updating status item information 243
open TCP/IP communications 146
OPER command 226
  specifying response text variables 311
  specifying target console 310
OPER RESP command 91
operator command 46
Operator facility, POVI
  APPLID 154
  using 175
options
  KABMTYPE
    Session Selection panel 181
  KABRMENU
    error handling 181
  LOG
    NOLOG 67
options, POVI
  KABLOG 180
  KABMTYPE
    Session Selection panel 181
  KABRMENU
    error handling 181

P

panel
  POVI Break Key Selection 165
  POVI Break Key Verification 166
  POVI Option Menu 167
  POVI Recorder Change Facility 170
  POVI Script Selection Menu 165
  POVI Session Selection 156, 181
parallel session 116
PARSE command
  using in REXX execs 61
pattern
  asterisk wildcard 68
pattern characters 55
  adjacent to 61
  asterisk wildcard 55, 68
  escape 55
pattern matching
  pattern variables 55
period 55
pound sign 55
question mark 55
pattern matching 55
asterisk 55
asterisk to match text 57
blanks in pattern 58
define a pattern variable 61
defined 55
name pattern matching 56
overview
  pattern characters 55
pattern variables 61
purpose of 55
pattern variables
$MATCH 61
adjacent to character 61
asterisk
  question mark
  pound sign 61
asterisk to match text 57
blanks in pattern 58
defined 55, 61
defining a pattern variable 61
name pattern matching 56
pattern characters 55
pattern variables 61
termination characters 61
TRAP ADD 61
TRAP CHANGE 61
patterns
creating 62
developing from software documentation 62
example 61
OMEGAVIEW Interface 240
wildcard 57
PCT
  See program control table (PCT)
PEEK function, of POVI 173
Playback Manager
  definition
    APPLID 154
Playback Manager APPLID 154
POVI 155, 157, 168, 169, 175, 180, 181, 182, 185, 186, 187
APPLIDs
  default APPLIDs 154
basic 182
Break Key Selection panel 165
Break Key Verification panel 166
comparison of automation methods 152
CUA Operator APPLID 154
definition 151
diagnostics 180
error handling 179, 181
external 182
external extended 182
from REXX exec 157
from VTAM terminal 155
internal extended 182
KABLOG 180
Keystroke Recorder 164
logon 155–157
message types 180
migration from a previous version 152
Operator facility APPLID 154
Option Menu panel 167
options 180
OVIAO APPLID 154
OVICUA APPLID 154
OVIOPER APPLID
  Basic Operator APPLID 154
OVIUSER APPLID
  Recording Manager APPLID 154
PEEK function 173
purpose 152
Recorder Change Facility panel 170
sample REXX exec 161–163
Script Selection Menu panel 165
Session Selection panel 156, 181
variables 169
POVI options
diagnostics 180
KABLOG 180
KABMTYPE
  Session Selection panel 181
KABRMENU
  error handling 181
message types 180
PPT
  See processing program table (PPT)
printing problems 11
procedures 255, 256
AF/OPERATOR commands, invoking from
  NetView 210
CICS messages, trapping and processing 255
IMS commands, trapping and processing 265
IMS messages, trapping and processing 267
IMS regions, issuing commands to 264
NetView alerts, generating 225
NetView console commands, trapping 221
NetView Interface status, displaying 207
NetView Interface, retrieving data through 212
NetView Interface, starting 205
NetView Interface, stopping 206
NetView REXX execs, invoking from
AF/OPERATOR
NetView commands, invoking from
AF/OPERATOR 226
NetView VTAM messages, trapping 222
processing program table (PPT) 255
program control table (PCT)

Q
quoted string 66
quotes
defined 66
rules for using in traps 66
using with ACTION keyword
pointers for use of 68

R
RC 185
recommendations
MVS dispatching priority 48
Recorder Change Facility 170
Recording Manager
definition
APPLID 154
relative order 23
defining 23
displaying 23
replacement
message
command 65
replacing a trap definition 70
reports
command files executed by traps 78
command files performing trap management 75
cross-reference facility error report 80
running the cross-reference reports 81
trap definitions sorted by argument 79
traps executing command files 77
traps manipulated by command files 76
response data types 127
DATARRERR 127
DATARPLY 127
retrieving data
through NetView Interface 211
retrieving data from AF/OPERATOR
example of 212
retrieving data from OMEGAMON for MVS
example of 215
retrieving data through
prerequisites for 211
procedure for 212
return codes

OMEGAVIEW Interface 235
REXX
error recovery 114
SIGNAL ON ERROR procedure 114
REXX API functions 96
REXX exec
error recovery 291
REXX execs
cross-referencing to traps 72
maintaining in storage
improving system performance 290
order of execution 67
using PARSE command in 61
REXX functions
NetView Interface 203
POVI 159
RKANCMD 81
RKOGLOGM 32
actions logged in 72
messages logged in 67
use for debugging 91
running under
OG*TSO 83

S
S 185
SCHED type LU 6.2 applid 118
SCREEN SIZE 193
script
definition 152
edit 170–171
modification 170–171
playback 168, 169
recording 164–168
selection 165
security 182, 193
extended 182
selection criteria 45
descriptor code 49
destination 49
job 49
origin 49
origin of command 45
task name and type 45
text 49
text string 45
send only session 98
SENDRCV session 100
SENDRCV session types 100
server AF/OPERATOR 94, 98
server definitions
TCP/IP 122
session 156, 181
session types
internal sessions 99
SENDRCV 100
sessions
confirm 119
defined 98
internal 99
parallel 116
SENDRCV 100
types of 98
setting
described 41
dynamically 44
setting a trap
described 41
dynamically 44
severity levels
customer support 315
SHOW MATCHES command
described 71
SHOW MATCHES NVI 204, 207
SHOW TRAP
testing traps
OG*TSO 90, 91
SHOW TRAPS command
described 71
side information dataset 118
SIGNAL ON ERROR procedure 114
simulated environment
OG*TSO 83
single quotes
using with ACTION keyword 68
SLF
features 73
SMF data, obtaining 71
special characters, choosing and placing 46
specifying response
ACTION keyword 65
described 65
match variables 67
rules for quotes 66
starting
AF/OPERATOR 30
starting and stopping
prerequisites for 205
STATUS 186
status item
dynamic rollup
patterns 240
status items, naming 238
status rules 238
defining
types 238
null 239
threshold 239
status, displaying 207
example of 208
prerequisites for 207
procedure for 207
stopping AF/OPERATOR 33
Subsystem Interface (SSI) 202
Subsystem Logging Facility
features 73
subsystem name 21
suppressing
SVC 34 45
SVC 35 49

T
tasks
automating VTAM 3270 applications 151
TCP/IP
communications protocol for AF 124
configurations requirements 122
server definitions 122
TCP/IP client application 129
ASCII connections 141
common errors 139
connect to server 130
exchange data with server
sending data 131
receiving data 132
expected results 135
initialization 130
object REXX for Windows 142
OS/390 REXX 135
process flow 130
running 135
termination 134
TCP/IP communications, advanced 124
AF TCP/IP communications protocol 124
automated link management 148
open TCP/IP communications 146
telephone support
customer support 314
termination character
pattern variables
variable name 61
TEST option
trap 72
testing
traps in OG*TSO 90
REXX execs in OG*TSO 83
text string 56
threshold rules 239

TOD
   AF/OPERATOR response
     actions at regular intervals 47
cautions 48
   selection criteria 47
   TON trap
     action at time interval
       example of 48
defined 37
described 47
   logic 47
   using for catch-up processing 47
   TP profile dataset 118

trace
   diagnostic messages 180
   internal functions 175
trace facility, communication 96
transient data queues
   definition of 253
translation
   WTO trap example 50
   trap 37, 38, 41, 44, 45, 46, 47, 48, 50, 51, 63, 65, 66, 67, 68, 69, 70
   adding 41
   CMD 45
cross referencing to REXX execs 72
defined 36
elements of 37
   enabling 69
   examples 43, 44
   exception traps 51
IMS interface 265, 266
information about 71
JES3 interface 54
NetView console VTAM messages
   NetView console commands 202
OMEGAMON exception 51
   replacing 70
termology 37
   TEST option 72
testing in OG*TSO 83, 90
TOD 47
type 37
   use of ampersands 63
WTO 49
WTOR 49
XO2 51
XOC 51
XOI 51
XOM 51
   xtype traps 51
TRAP ADD CMD 263, 265
TRAP ADD command
   ACTION keyword 38
   ENABLE keyword 37
   ENABLE option 69
   LOG option
     NOLOG option 67
     specifying responses in 65
     syntax of 42
   using to replace traps 70
   using to set a trap 41
TRAP ADD WTO 253, 254, 255, 263
TRAP ADD xtype, OMEGAMON exceptions 52, 194
TRAP CHANGE command
   LOG option
     NOLOG option 67
     using pattern variables in 61
   using to modify traps 70
TRAP CMD 220
TRAP DELETE command 70
TRAP DISABLE command 69
TRAP ENABLE command 69
TRAP WTO 222, 267
   trap, CMD
     See command traps
   trap, WTO
     See wait to operator traps
trapping
   example of 221, 223, 258
   prerequisites for 220, 222
   procedure for 221, 222
   trapping and processing
     description of 255, 256, 266
     prerequisites for 255, 256, 267
     procedure for 255, 256, 267
   trapping MVS commands
     cautions
       CMD trap cautions 46
   traps
     cross referencing 74
   TSO 83
   TYPE 186
type
   CMD 37
   TOD 37
   WTO 37
   xtype 37

variable 180, 181
   in ACTION field
Index

in REXX execs 63
LINE 91
prefixes 63
REXX line variables, OMEGAVIEW Interface 234
substituting values for 63
termination 61
using ampersands in
in traps 63
using double ampersand in 63
variables
ampersands 63
AONVCLST 204
AONVICMD 204
AONVICTM 204
AOSSNAME 204
REXX shared variables 26
where to find information on 26
VSAM datasets, user-defined 118
VTAM
trapping NetView messages 202
VTAM application 152
VTAM configuration for APPC interface support 116

W
WAIT 187
WAIT exception, responding to 195
WAIT option
of the TRAP command 46
wildcard character
asterisk
in a pattern 68
wildcard patterns
asterisk 42, 57
WTO trap
defined 37
described 49
example of 50
message translation 50
specifying actions in 65
WTOR
WTO traps 49
WTOR, issuing 111
WTORT, issuing 112

X
XCF
configurations requirements 120
naming examples 121
rules for naming groups and members 120
XCTL command to CICS 254
XOC trap 51
XO2 trap 51
XOI trap 51
XOM trap 51