CA Automation Point

Command and Keyword Reference Guide

Release 11.4.1

Second Edition

Contact CA Technologies

Contact CA Support

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

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

Providing Feedback About Product Documentation

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

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

Documentation Changes

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

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

- Updated the URIs, HTTP Methods, and XML Documents (see page 389) section.

Contents

Chapter 1: Introduction

This guide describes the syntax for the various CA Automation Point keywords, parameters, and command processors. The following topics are covered:

- Keyboard parameters
- Scan code parameters
- Rules keywords
- Script keywords
- Miscellaneous keywords
- AXC command processors
- PPQ command processors
- GLV command processors
- VOX command processors
- TNG command processors
- OPS command processors
- Notification Manager command processors

Notation Conventions

The following legend helps you read the syntax diagrams provided for each statement and command:

MIXed Case

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

lowercase italics

Identifies a value that you must supply.

[ ] Brackets

Identifies optional operands.

{ } Braces

Indicate a list of mutually exclusive (meaning you must specify only one) constants separated by a vertical bar or bars {|}.

Underlined Text

Indicates the default value.

Note: Positional parameters, keywords, and operands are indicated by indented text in the syntax diagrams.

Chapter 2: CA Automation Point Parameters

This section contains the following topics:

Keyboard Parameter Summary

Keyboard parameters define keystrokes in the keyboard parameter file. The following table lists the CA Automation Point parameters you need to define when specifying your keyboard parameter file.

KEY

Identifies the name of the operation.

SCAN

Indicates the electronic location of the key.

MAP

Specifies the type of window on which this operation applies.

Keyboard Parameter Syntax

You can enter keyboard parameters in any case.

- Start each key definition in column 1.
- Separate the parameters by placing a comma between them.
- Begin a comment line with an asterisk (*).
- When specifying a comment on the same line as a statement, leave two or more blanks between the last parameter and the start of the comment.
- Operands shown in brackets ([ ]) are optional.

KEY Parameter

The KEY parameter identifies the name of the operation.

This parameter has the following format:

KEY=*operation*

operation

Specifies the operation or key label.

Examples:

KEY=PF9

KEY=3270_1

KEY=A

KEY=a

SCAN Parameter

The SCAN parameter indicates the electronic location of the key.

This parameter has the following format:

SCAN=*number* [(*shifts*)]

number

Specifies the sequence number associated with a given key. This number changes depending on the physical layout of the keyboard (for example, 83-key keyboard, 101-key keyboard). If you do not use the utility program, look up this location in the reference manual for your keyboard.

shifts

(Optional) Indicates any shift states that must also be true for this operation. Possible shifts values are as follows:

RIGHT_SHIFT

Press the Shift key located on the right-hand side of the keyboard.

LEFT_SHIFT

Press the Shift key located on the left-hand side of the keyboard.

SHIFT

Press either the left or right Shift key.

LOWER

Indicates a lowercase letter (a through z).

CAPS_LOCK

Press the Caps Lock key. You must press the Caps Lock key twice-once for the operation and once to unlock the CAPS LOCK.

UPPER

Press the right Shift key, the left Shift key, or the Caps Lock key. You must press the Caps Lock key twice-once for the operation and once to unlock the CAPS LOCK.

SCROLL_LOCK

Press the Scroll Lock key. You must press the Scroll Lock key twice-once for the operation and once to unlock the SCROLL LOCK.

NUM_LOCK

Press the Num Lock key. You must press the Num Lock key twice-once for the operation and once to unlock the NUM LOCK.

CTRL

Press the control (Ctrl) key.

ALT

Press the alternate (Alt) key.

ANY

Indicates that whatever shift state is true, the operation is always mapped to the indicated key.

Examples:

```
SCAN=30(LOWER)

SCAN=30(UPPER)
```

MAP Parameter

The MAP parameter specifies the type of window on which this operation applies.

This parameter has the following format:

MAP=*sessiontype*

sessiontype

Valid values are:

AUTOMATE

Specifies that the key applies only in asynchronous session windows, CA Automation Point function windows, and CA Automation Point menus and dialogs.

TERMINAL

Specifies that the key applies only in CA Automation Point 3270 or 5250 terminal emulator session windows.

(AUTOMATE, TERMINAL)

Specifies that the key applies in both types of session windows.

Default: (AUTOMATE,TERMINAL)

Scan Code Parameters

Scan code parameters define operations in scan code files. For details about customizing these types of files, see the appendix on customizing special CA Automation Point files in the *Administrator Guide*.

The following are the CA Automation Point parameters you need to define when specifying your scan code parameter file.

3270_KEY

Specifies the operation that is being defined (for example, A, PF1, and so on).

3270_SCAN

Specifies the terminal-specific scan code for the operation.

When defining operations in a scan code file, follow these rules:

- You can enter scan code parameters in uppercase or lowercase.
- Start each definition in column 1.
- Separate the parameters by placing a comma between them.
- Begin a comment line with an asterisk character (*).
- Leave two or more blanks between the last parameter and the start of the comment when specifying a comment on the same line as a statement.

3270_KEY Parameter

The 3270_KEY parameter specifies the operation that is being defined (for example, A, PF1, and so on).

This parameter has the following format:

3270_KEY=*operation*

operation

Specifies the name of the operation (similar to a command name).

Examples:

3270_KEY=PF9

3270_KEY=A

3270_KEY=a

3270_SCAN Parameter

The 3270_SCAN parameter specifies the terminal-specific scan code for the operation.

This parameter has the following format:

3270_SCAN=*scancode*

scancode

Specifies the terminal-specific scan code of the operation.

Examples:

3270_SCAN=(4d,12,cd)

3270_SCAN=4b

Chapter 3: CA Automation Point Keywords

This chapter describes the keywords that you use to create rules and for use in scripts. For details about the CA Automation Point rules language, see "Writing Rules" in the *Administrator Guide*.

This section contains the following topics:

Rules Keyword Summary

Rules keywords are used in writing CA Automation Point rules. The following sections summarize the rules keywords.

Keywords for Defining Automation Criteria

Use the following keywords to define automation criteria.

CMDIN

Defines the beginning of a command rule (which activates when the specified command is issued).

EVERY

Specifies how often you want an action to be taken.

LIMIT

Specifies how many times a CA Automation Point rule can execute in a given minute.

MATCHLIM

Limits the number of times that an action specified by a rule takes effect.

MSGID

Defines the beginning of a message rule (which activates when a message having the specified ID appears).

TIME

Defines the start of a time rule (which activates at a certain time or after a specified time period has passed).

WHEN

Defines additional conditions that must be true for a rule to activate.

Keywords for Responding to System Events

Use the following keywords to respond to system events.

DOSCMD

Issues an operating system command or executes a command file.

OSCMD

Issues an operating system command to a console.

PPQWRITE

Writes an item to a PPQ.

REPLY

Specifies the reply to a WTOR message.

REXX

Invokes a REXX program.

SCRIPT

Starts a CA Automation Point script.

SESSCMD

Sends a keystroke string to a session.

SESSION

Restricts the processing of MSGID rules to a given session.

SET

Creates, deletes, modifies, or assigns a value to a status variable.

XCCMD

Invokes a CA Automation Point command processor from a rule.

Keywords for Controlling the Display

Use the following keywords to control the display.

COLOR

Specifies the color in which you want a message to appear.

DISPLAY

Displays a previously suppressed message in the Normal, Action, or Merged Msg window.

DOM

Deletes an action message from the Action Message Recall window and from the action message area of the Merged Msg window.

HILIGHT

Displays a message in the Action Message Recall window and in the action message area of the Merged Msg window.

LOWLIGHT

Displays a message in the Normal Message Recall window and in the main messages area of the Merged Msg window.

PREFIX

Specifies the prefix of the messages that appear on Automation Point Message Recall and Merged Msg windows.

REWORD

Alters the text of a message.

SUPPRESS

Prevents a message from being displayed in Automation Point Message Recall and Merged Msg windows.

WTO

Issues a message to the Merged Msg window and either the Action Message Recall window or the Normal Message Recall window.

WTXC

Issues a write-to-operator message and sends it to the CA Automation Point Msg window.

Keywords for Logging Messages

Use the following keywords to log messages.

LOG

Sends a message to the host message log file or the CA Automation Point message log file.

NOLOG

Prevents a message from being sent to the host message log file.

NOPRINT

Prevents a message from being printed on the hardcopy log.

PRINT

Prints a message on the hardcopy log.

Keyword for Controlling Commands

Use the following keyword to control commands:

CMDIN

Defines the start of a command rule.

Keywords for Notification

Use the following keywords for notification tasks.

ALARM

Sounds an alarm when a specified message occurs.

NOALARM

Specifies that no alarm be sounded when a message appears.

Keywords for CA NSM Event Traffic Controller

Use the following keywords for CA NSM Event Traffic Controller.

NOUNIFWD

Tells CA Automation Point *not* to forward a message that is processed by rules to CA NSM.

SNMPTRAP

Sends an SNMP trap to the specified host.

UNICMD

Tells CA NSM Event Manager, which resides on the specified host, to execute the supplied command.

UNIFWD

Tells CA Automation Point to forward a message that is processed by rules to all recorded CA NSM hosts.

UNIWTO

Sends the supplied message to CA NSM Event Manager on the specified host.

Keyword for Interfacing with Third-party Applications

Use the following keyword for Third-party Applications.

EXPORTMSG

Tells CA Automation Point to send the message to an ApExportMsg function created by the user, which then forwards messages to a third-party software application.

Rules Keyword Syntax

When writing CA Automation Point rules, follow these guidelines:

- Begin each rule with the MSGID keyword or the TIME keyword; follow it with other appropriate keywords.
- Use positions 1 through 255 of each line.
- Do not split a keyword phrase across lines.
- Separate rules keywords within a line by using a comma, one or more blanks, or both.
- Write a comment line by placing an asterisk (*) in column 1.
- Operands shown in brackets ([ ]) are optional.

Keywords for Defining Automation Criteria

The following sections describe the CA Automation Point keywords used to define automation criteria.

EVERY Keyword

The EVERY keyword controls how often a rule executes again after it has executed the first time.

This keyword has the following format:

EVERY*(n interval*)

n

Indicates how often or at what time interval you want the action to be taken. You can specify any positive integer, but the value you specify cannot be a status variable.

interval

Specifies the unit of time for the interval. This value can be any of the following:

SECOND(S)

Tells CA Automation Point to take the action every *n* seconds.

MINUTE(S)

Tells CA Automation Point to take the action every *n* minutes.

HOUR(S)

Tells CA Automation Point to take the action every n hours. In this case, the *n* value can be any number from 0 through 23.

Example:

Suppose that the following message, which shows the current spool usage, appears more frequently than necessary:

$HASP093 *xx*% SPOOL UTILIZATION

The following rules cause this message to appear at intervals at least ten minutes apart:

```
MSGID($HASP093), SUPPRESS
MSGID($HASP093), EVERY(10 MINUTES), DISPLAY
```

LIMIT Keyword

The LIMIT keyword specifies the number of times a rule can execute within a given minute. If one of your rules contains an error causing CA Automation Point to get caught in an endless loop, the limit prevents CA Automation Point from executing the same rule repeatedly.

This keyword has the following format:

LIMIT(*integer*)

integer

Specifies the number of times a rule is to execute in a given minute. You can specify any value between 1 and 32767. You cannot specify a status variable as the operand.

Usage Notes:

- If you do not specify the LIMIT keyword, CA Automation Point uses a default limit of 25 for rules that contain any of the following keywords:

  DOSCMD

  Issues an operating system command or executes a command file.

  OSCMD

  Issues native operating system commands to a session. Operating system commands include z/OS, JES3, VSE, z/VM, and i5/OS (iSeries).

  REPLY

  Replies to a WTOR message.

  REXX

  Invokes a REXX procedure.

  SCRIPT

  Invokes CA Automation Point script files.

  SESSCMD

  Sends a keystroke string to a session.

  WTXC

  Issues write-to-operator messages to the AP Messages window.

  XCCMD

  Invokes CA Automation Point command processors.

Example:

Suppose that you want a script called LOGONRCS to execute no more than twice within one minute when the IEF244I message is issued. You could write a rule like this one:

```
MSGID(IEF244I), SCRIPT(LOGONRCS.SCR), LIMIT(2)
```

MATCHLIM Keyword

The MATCHLIM keyword specifies the maximum number of times a message can be seen by rules. After CA Automation Point receives the MATCHLIM number of messages matching the MSGID criteria for this rule, CA Automation Point no longer executes this rule.

This keyword has the following format:

MATCHLIM(*integer*)

integer

Specifies the maximum number of times a message can be seen by rules. This can be any value from 1 to 32767. However, you cannot specify a status variable as the value for the MATCHLIM keyword.

Example:

Suppose that when you first start CA Automation Point you want to set a status variable (called &START_TIMESTAMP) that contains the startup date and time. The following rule executes once when CA Automation Point processes its first message:

```
MSGID(*), MATCHLIM(1), SET(&START_TIMESTAMP=&DATE:&TIME)
```

MSGID Keyword

The MSGID keyword defines the start of a message rule.

This keyword has the following format:

MSGID(*characters*)

characters

Identifies a message that you want CA Automation Point to process. You can specify multiple messages by specifying the portion of the message ID that indicates a specific group of messages.

Usage Note:

CA Automation Point processes MSGID rule clauses in the following order:

MSGID(), MSGID(*string*), MSGID(*)

Examples:

- In the following example, z/OS issues message IRA201I when there is a critical shortage of auxiliary storage. The following rule highlights message IRA201I:

  ```
  MSGID(IRA201I), HILIGHT
  ```

- In the following example, CA Automation Point issues the following message when it cannot send keystrokes to a given session because of a workstation problem or host failure:

  ```
  AXC0901E  Continued sendkey failures to session S008
  ```

  The following rule dispatches a REXX procedure called CHKSESS to determine whether a workstation problem or a host problem generated message AXC0901E:

  ```
  MSGID(AXC901E), EVERY(5 MINUTES),
    REXX(CHKSESS SESSION(&WORD7))
  ```

- In this example, OpenVMS issues this message when a VAX printer goes offline:

  ```
  %%%%%%%%%% OPCOM   8-APR-1994 06:10:19.32 %%%%%%%%%%
  DEVICE $1$LPA0: (VAXD) IS OFFLINE
  ```

  The following rule sounds an audible alarm when this message contains the text LP and OFFLINE:

  ```
  MSGID(DEVICE),
    WHEN(&WORD2(4:5) EQ 'LP' AND &WORD5 EQ OFFLINE),
      ALARM(3)
  ```

  Note: The rule shows DEVICE as the MSGID. When you write messages for OpenVMS sessions, the MSGID begins after the OPCOM header.

  In this example, MSGID rule clauses are case-sensitive. For the message Please mount tape 1233, use the following:

  ```
  MSGID(Please)
  ```

  For the message PLEASE MOUNT TAPE 1233, use the following:

  ```
  MSGID(PLEASE)
  ```

TIME Keyword

The TIME keyword specifies the time when a rule can execute. The time you specify cannot be a status variable value.

This keyword has the following format:

TIME(*hh:mm*)

hh:mm

Specifies the time when a rule is to execute in hours (*hh*) and minutes (*mm*) on a 24-hour clock.

Example:

If your site routinely IPLs the mainframe at 7:00 p.m., write the following time rule to notify TSO users ten minutes before the IPL occurs:

```
TIME(18:50),
OSCMD(('SEND “WARNING; IPL IN 10 MINUTES” '),SESSION(SYS_A))
```

WHEN Keyword

The WHEN keyword specifies additional conditions that must be true in order for a rule to take effect.

This keyword has the following format:

WHEN*(expression)*

expression

This option has the following format:

[*operand relational_operator operand*] [*Boolean_operator operand relational_operator operand*]

operand

Can be a character string, an environmental variable, or both.

relational_operator

Can be any of the following character strings:

EQ--Equal

NE--Not equal

GT--Greater than

GE--Greater than or equal

LT--Less than

LE--Less than or equal

IN--Contains a character string

NOTIN--Does not contain a character string

Boolean_operator

Can be one of the following:

- AND
- OR

You can join logical expressions together using the Boolean operators (AND and OR) and nested parentheses. Unquoted blanks that immediately precede or follow operators are not considered text.

Usage Notes:

- CA Automation Point converts all strings without quotation marks in a WHEN clause to uppercase. Case is preserved for all strings with quotation marks. There is no difference between the following two examples:

```
WHEN(&WORD3 EQ hello)
WHEN(&WORD3 EQ HELLO)
```

- To match lowercase or mixed case strings, use strings with quotation marks.

  The following example only matches the lowercase word:

  ```
  WHEN(&WORD3 EQ 'hello')
  ```

Example:

The following message indicates that OS39OSTC has terminated abnormally:

```
IEF4501I OS39OSTC--ABEND=S0C4 U0000 REASON=00000000
```

The following rule checks the second word and the last two characters of the fifth word in the message before alerting the operator:

```
MSGID(IEF4501I)
 WHEN(&WORD2 EQ OS39OSTC AND &WORD5(9:10) NE 22),
 COLOR(BRIGHT RED), ALARM(19)
```

The rule determines whether OS39OSTC has terminated if the second word (&WORD2) of the message is OS39OSTC and the last two digits of the fifth word (&WORD5) are not 22. When these conditions exist, the rule alerts the operator to the problem.

Keywords for Responding to System Events

The following sections describe the CA Automation Point keywords used to respond to system events.

DOSCMD Keyword

The DOSCMD keyword allows you to issue a workstation operating system command or to execute a command file.

This keyword has the following format:

DOSCMD(*cmdtext*)

cmdtext

Specifies the text of the operating system command or the name of the command file.

Usage Notes:

- When issuing a command with the DOSCMD keyword, make sure that the command is valid for the operating system targeted to receive it. The DOSCMD keyword passes the command to the current operating system without checking to see if the command works in that environment.
- Make sure that the command or command file can execute completely without input, keystrokes, or other help from an operator.
- CA Automation Point does not directly capture output from the DOSCMD keyword. If you use DOSCMD to issue operating system commands and want to view the output of those commands, redirect the output to disk files.

Example:

Suppose that CA OPS/MVS sends the following message to request CA Automation Point to check the local area network (LAN) for activity:

```
AXC000I CHECK LAN, PLEASE
```

The following rule invokes a batch file that checks the LAN and sends the result back to CA OPS/MVS:

```
MSGID(AXC000I CHECK LAN), DOSCMD(CHECKLAN)
```

OSCMD Keyword

The OSCMD keyword tells CA Automation Point to issue an operating system command to the session that sent the current message. For example, you could use the OSCMD keyword to invoke a CA OPS/MVS REXX program.

This keyword has the following format:

OSCMD(*commandtext*)

or

OSCMD((*commandtext*) SESSION(*sessid*))

commandtext

Specifies the text of the command. If you specify the SESSION operand, the OSCMD keyword directs the operating system command to the indicated session (specified by *sessid*), rather than to the session that generated the message.

SESSION

Specifies the session to which the OSCMD command is to be directed. This operand cannot be used to specify the name of an internal session (AXC, VOX, or OPS).

Usage Note:

If you use the OSCMD keyword in a time rule, the SESSION operand is *required* because time events are not associated with specific sessions.

Example:

The following message indicates that CA OPS/MVS is active:

```
OPS0123O AOF INITIALIZATION COMPLETE
```

The following rule issues a command to CA OPS/MVS indicating that CA Automation Point is active and ready for CA OPS/MVS to use it:

```
MSGID(OPS0123O), OSCMD(AXC HERE)
```

PPQWRITE Keyword

The PPQWRITE keyword writes an item to a PPQ.

This keyword has the following format:

PPQWRITE(ITEM(*item*) QUEUE(*queuename*))

ITEM

Specifies the item to write to the queue. The value of item can be either a literal string (such as “this is an item”) or a simple variable name (not enclosed in quotation marks so that REXX can evaluate it).

QUEUE

Specifies the name of the queue. This value must be in uppercase.

Example:

The following sends information to a queue named MESSAGE:

```
PPQWRITE((HELLO THERE) QUEUE(MESSAGE))
```

REPLY Keyword

The REPLY keyword specifies the reply for a write-to-operator-with-reply (WTOR) message.

This keyword has the following format:

REPLY(*string*)

string

Specifies the text of the reply for a WTOR message.

Example:

z/OS issues the following two messages when JES2 cannot start because of an incorrectly defined exit routine:

```
$HASP857 EXIT013 NOT DEFINED WITHIN CURRENTLY LOADED JES2 MODULES
*00 $HASP441 REPLY 'Y' TO CONTINUE OR 'N' TO TERMINATE
```

The following two rules automate the reply to message $HASP441, letting other messages fall through to the operator:

```
MSGID($HASP857),
 SET(&HASP441_REPLY = Y)
MSGID(IEA120A),
 WHEN(&HASP441_REPLY NE ''),
 REPLY(&HASP441_REPLY)
 SET(&HASP441_REPLY = '')
```

REXX Keyword

The REXX keyword invokes the REXX procedure specified by filename.

This keyword has the following format:

REXX(*filename arguments*)

filename

Specifies the name of the file containing the REXX procedure.

arguments

Specifies any additional arguments needed when you invoke the procedure.

Example:

Suppose that CA Automation Point receives the following message from CA OPS/MVS on the host:

```
OPS000I REXX IPL9672
```

The following sample rule invokes a REXX procedure called APCMOS, initiating the IPL of a 9672 processor:

```
MSGID(OPS000I REXX IPL9672),
 REXX(APCMOS ACTION(PSWRESTART) CPC(&CPC) LPAR(&LPAR))
```

SCRIPT Keyword

The SCRIPT keyword specifies a CA Automation Point script to be invoked in the session that issued the current message.

This keyword has the following format:

SCRIPT(*filename*)

or

SCRIPT((*filename*) SESSION(*sessid*))

filename

Specifies the name of the CA Automation Point script to be invoked.

SESSION

Specifies the session in which the script is to be invoked. If your SCRIPT value contains the SESSION operand, CA Automation Point directs the script to the indicated session instead. Use the session ID value to identify the session that should invoke the script. This operand cannot be used to specify the name of an internal session (AXC, VOX, or OPS).

Usage Note:

If you use the SCRIPT keyword in a time rule, the SESSION operand is *required* because time events are not associated with particular sessions.

Example:

The following example starts the MYSCRIPT script when message $HASP308 is issued:

```
MSGID ($HASP308), SCRIPT(MYSCRIPT.SCR)
```

SESSCMD Keyword

The SESSCMD keyword specifies a keystroke string that can contain text or keystrokes to be sent to the session that issued the current message.

This keyword has the following format:

SESSCMD(*keystrokes*)

or

SESSCMD((*keystrokes*) SESSION(*sessid*))

keystrokes

Specifies the keystrokes to be sent to the specified session. This value can be any character string with a maximum length of 488 characters (after expanding all variables). The command line of the receiving session must be long enough to accept all of the characters in the string.

SESSION

Specifies the session to which to send the command. If you specify the SESSION operand, the SESSCMD keyword directs the string to the indicated session (*sessid*), rather than to the session that issued the message. This operand cannot be used to specify the name of an internal session (AXC, VOX, or OPS).

Usage Note:

Use the SESSCMD keyword instead of the OSCMD keyword to send PF keys and other special keystrokes such as MODE SELECT.

Example:

The following message appears on a mainframe processor console during IPL and indicates that z/OS is waiting for the operator to press the TOD enable switch:

IEA889A DEPRESS TOD CLOCK SECURITY SWITCH

Once the TOD clock is active, z/OS sets the mainframe clock.

The following rule sets the mainframe clock by sending an S1 command (the TOD enable switch) to the console on session MACH1_PCON:

MSGID(IEA889A), SESSCMD((S1) SESSION(MACH1_PCON))

SESSION Keyword

The SESSION keyword restricts rules processing by session name, by a set of session names, or by session type. Using the SESSION keyword instead of complex WHEN clause improves overall message processing throughput. Multiple SESSION statements for the same MSGID rule are cumulative.

This keyword has the following format:

SESSION(*name*|$*type*)

name

Specifies the name of an enabled session defined to the active session definition set. Specifying SESSION(*name*) causes CA Automation Point to evaluate the message rule only if the message originated from the specified session. Specifying SESSION(*) causes CA Automation Point to evaluate the message rule for messages originating from all sessions.

To restrict the rule to only those messages issued by CA Automation Point itself, use the ”AXC” session name. To restrict the rule to messages generated by the Notification Server, use the “VOX” session name. To restrict the rule to messages received by the CA OPS/MVS Interface, use the “OPS” session name.

$type

Specifies the type of session. Valid values are:

ASYNCH	DTX	JES3
JESMCS	MCS	RCS
SYSPLEX	TANDEM	TANDEMALL
VAX	VAXALL	VM
VSE		

Specifying SESSION($*type*) causes CA Automation Point to evaluate the message rule only if the message originated from an enabled session of the specified type.

Note: The TPFASYNCH session type is included with other sessions of type ASYNCH. For TPF3270 sessions, you must use the session name to identify the session.

Examples:

- The following rule alerts the operator when the specified message is received for the session named MACH1_PCON:

```
MSGID(IEF4501I)
 SESSION(MACH1_PCON)
 ALARM(19)
```

- The following rule alerts the operator when the specified message is received by sessions with names of MACH1_PCONA, MACH1_PCONB, and MACH1_PCONC. Messages from sessions not specified in the rule skip the rule.

```
MSGID(IEF4501I)
 SESSION(MACH1_PCONA)
 SESSION(MACH1_PCONB)
 SESSION(MACH1_PCONC)
 ALARM(19)
```

- The following rule alerts the operator when the specified message is received by sessions of type MCS. Messages from sessions that are of type ASYNCH skip the rule.

```
MSGID(IEF4501I)
 SESSION($MCS)
 WHEN(&WORD2 EQ OS390STC AND &WORD5(9:10) NE 22),
 COLOR(BRIGHT RED),
 ALARM(19)
```

SET Keyword

The SET keyword creates and assigns a value to a status variable. You can also use the SET keyword to delete a status variable or update its value. You can use the SET keyword more than once in a rule.

This keyword has the following format:

SET(*&varname=text*)

&varname

Specifies the name of the status variable, which can contain up to 32 characters. An ampersand (&) must always precede the variable name.

text

Specifies the text string you want to assign to the status variable. A text string can contain characters, environmental variables, or other status variables. If the value of text is null, CA Automation Point deletes the status variable that you are trying to set. The maximum length of text is 256 characters (after expanding all variables).

Usage Note:

Storing values in status variables uses up memory. Therefore, if a rule references a variable only once or a few times and never uses that variable again, set the variable to null after the rule using it executes. Leave only frequently-used variables in memory.

Example:

The following rule creates a status variable called &JOB_START_TIME with the same value as the &HOSTTIME environmental variable:

```
MSGID(IEF490I), SET(&JOB_START_TIME=&HOSTTIME)
```

XCCMD Keyword

The XCCMD keyword causes a rule to invoke a CA Automation Point command processor specified by *cmdtext*. For descriptions of CA Automation Point command processors and their operands, see ADDRESS AXC Commands (see page 67).

This keyword has the following format:

```
XCCMD(cmdtext)
```

cmdtext

Specifies the text of any CA Automation Point command processor, its operands, and the operands' values.

Usage Notes:

- The XCCMD rules keyword does not invoke REXX and WTOH CA Automation Point command processors. Issue the REXX command processor from a rule with the REXX rules keyword. Issue the WTOH command processor from a rule with the WTO and HILIGHT rules keywords.
- When the command processor specified with the XCCMD keyword executes, it *does not* return information to the CA Automation Point rules processor because rules cannot process the information.

Example:

The following rule clause names the window that displays the MsgPre graph:

```
XCCMD('PLOT MsgPre DEFINE GRAPH WINDOW(My_Plot_Window)')
```

Keywords for Controlling the Display

The following sections describe the CA Automation Point keywords used to control the CA Automation Point display.

COLOR Keyword

The COLOR keyword sets the color in which CA Automation Point displays a message in the Merged Messages, Action Message Recall, and Normal Messages Recall windows.

This keyword has the following format:

COLOR(*colorname*)

colorname

Specifies the color of message text. The *colorname* value can be any of the following:

BLUE	YELLOW	TURQUOISE
BRIGHT BLUE	GREEN	BRIGHT TURQUOISE
RED	BRIGHT GREEN	PURPLE
BRIGHT RED	BROWN	BRIGHT PURPLE
WHITE	BRIGHT WHITE	GRAY
BLACK		

Default: WHITE

Usage Note:

If the message was issued by the Notification Server, the color change appears in the function window with the default name of AP Notification Messages (VOXMSG window type). If the message was received by the CA OPS/MVS Interface, the color change appears in the function window with the default name of CA-OPS/MVS Messages (OPSMSG window type).

If the color black is selected for a message that is to be displayed in a window with a black background, the color of the message changes to white.

Example:

The following message indicates that the WTO buffer is full:

```
IEA404A  SEVERE WTO BUFFER SHORTAGE--100% FULL
```

The following rule alerts the operator to the buffer shortage problem by displaying message IEA404A in bright red text:

```
MSGID(IEA404A), COLOR(BRIGHT RED)
```

DISPLAY Keyword

The DISPLAY keyword causes CA Automation Point message recall windows to display a message that may have been previously suppressed with the SUPPRESS keyword.

This keyword has the following format:

DISPLAY

Usage Note:

The default color of the displayed message is white, but you can specify a different color with the COLOR keyword.

Example:

The following message shows the current spool usage and appears more frequently than necessary:

$HASP093 *xx*% SPOOL UTILIZATION

The following rules limit the number of times the operator sees this message by forcing the message to display at intervals at least ten minutes apart:

```
MSGID($HASP093), SUPPRESS
MSGID($HASP093), EVERY(10 MINUTES), DISPLAY
```

DOM Keyword

The DOM keyword deletes an action message from the action message area of the Merged Messages window and from the Action Message Recall window.

This keyword has the following format:

DOM(DOMID(*domid*))

DOMID

Specifies the internal pointer to an action message, which must already be stored in a status variable.

Examples:

- Suppose that you want rules to delete an action message whenever CA Automation Point receives a terminating message. Assuming that message 111ABC is the starting message and message 999XYZ is the terminating message, these sample rules statements show how to delete the starting message:

  ```
  MSGID(111ABC), HILIGHT, SET(&SAVE111ABC=&DOMID)
  MSGID(999XYZ), DOM(DOMID(&SAVE111ABC)), SET(&SAVE111ABC='')
  ```

  CA Automation Point deletes message 111ABC after receiving message 999XYZ and resets the &SAVE111ABC status variable to null.

- Suppose that you have a large number of tape mount messages and you want to ensure that all CA Automation Point processed tape mount messages are removed from the CA Automation Point Action Message window, even if the original tape mount message has scrolled off the screen. The following rules delete the messages:

  ```
  *** Handle tape mount messages
  * IEF233a m 455,812887,,RPNLCOPY,STEP1 and
  * IEF234e k 455,812887,,PVT,PROPDEPT,STEP03
  * See the mount request and set drive_var to domid *
  MSGID(IEF233A), SET(&(DRIVE_&WORD3)=&DOMID)
  * See that message is satisfied and delete value *
  * of &(DRIVE_&WORD3)                    *
  MSGID(IEF234E) DOM(DOMID(&(DRIVE_&WORD3))) SET(&(DRIVE_&WORD3)='')
  ```

HILIGHT Keyword

The HILIGHT keyword promotes a captured message to action message status. If the message is to be displayed in the Merged Messages window, it will be shown in both the Action Message Area of this window and the Action Message Recall window.

This keyword has the following format:

```
HILIGHT
```

Usage Note:

Use this keyword only if the message is not already an action message.

Example:

```
MSGID($HASP13I), HILIGHT
```

LOWLIGHT Keyword

The LOWLIGHT keyword demotes a captured message to normal message status. If the message is to be displayed in the Merged Messages window, it will be shown in both the Normal Message Area of this window and the Normal Message Recall window.

This keyword has the following format:

LOWLIGHT

Example:

MSGID($HASP093), LOWLIGHT

PREFIX Keyword

The PREFIX keyword specifies the prefix of messages appearing in certain CA Automation Point windows or files.

This keyword has the following format:

PREFIX(*string*)

string

Specifies the prefix of the messages appearing in the following CA Automation Point windows or files:

- Merged Message window
- Normal Messages window
- Action Messages window
- Host messages log file
- Print log file
- CA Automation Point notification messages
- CA OPS/MVS messages

Specify this prefix information in a series of character strings separated by blank spaces. You form these character strings from a set of twelve alphanumeric characters, each representing a character from a different type of information. For example, the letter T represents a character from the message time stamp, and the letter J represents a character from a job name.

For example, if your message string contains TTTTTT, all six characters of the time stamp appear in the message text:

122433 IEC501A M 480, MIM,BLP, 1600 BPI, MIM02, CLONETAP

The order and punctuation of the character strings determine the order and punctuation of the actual pieces of information they represent. For example, the following character strings display both the time and the job name, in that order.

TT:TT:TT JJJJJ
12:24:33 **MIM10** IEC501A M 480, MIM,BLP, 1600 BPI, MIM02, CLONETAP

Character	Meaning
3	Represents &JES3name for SYSPLEX or JES3MCS consoles, or system name for other sessions.
A	Represents the action indicator. If the message is issued as an action message, this indicator has a non-blank value, usually the asterisk (*) or the at-sign (@).
C	Represents a character of the &MONTYPE access variable. Use uppercase C only; stands for category.
E	Represents a character of the &MONPRTY access variable. Use uppercase E only; stands for error.
H	Represents a character of the &MONNAME access variable. Use uppercase H only; stands for host.
J	Represents a character of the job name or identifier from the message. Use uppercase J only.
L	Represents the string specified in the session definition as a window title. Use uppercase L only.
N	Represents a character of the &MONNUM access variable. Use uppercase N only; stands for number.
P	Represents a character of the default system name that is displayed with messages received from a session. Use uppercase P only. (You set a default system when you define your session.) Some processors need this character to route the information properly.
R	Represents a character of the reply ID for a write-to-operator with reply (WTOR) message. Use uppercase R only.
S	Represents a character of the session name for the session receiving the message. Use uppercase S only. This name is the ID that you assign to a session when you define the session.
T	Represents a character of the timestamp from the message. Use uppercase T only.

Global Default: SSSSSSSS TTTTTT JJJJJJJJ RRA.

To provide no prefix, specify two single quotes ('') in the edit box. This setting overrides the default.

Default: The value the Local Session Prefix in the session definition

Usage Note:

CA Automation Point formats multi-line messages so that each line uses all 80 columns of the display area. If these messages included prefixes, CA Automation Point splits the messages onto two lines.

Example:

The following message is the first line of a multi-line message issued in response to a DISPLAY CONSOLES command:

```
IEE249I 13.30.23 CONSOLE DISPLAY 741
```

The following rule prevents the CA Automation Point prefix from appearing at the beginning of the message, and prevents CA Automation Point from splitting the message:

```
MSGID(IEE249I), PREFIX()
```

REWORD Keyword

The REWORD keyword rewords the text of a message before it is displayed in the Merged Messages window, the Action Message Recall window, or the Normal Message Recall window; or rewords the text of a command before CA Automation Point sends it to the console.

This keyword has the following format:

REWORD(*text*)

text

Specifies the words of the message or command in the order in which you want them sent.

Usage Notes:

- If the message was issued by the Notification Server, the rewording of this message will be displayed in the function window with a default name of AP Notification Messages (window type of VOXMSG).
- If the message was received by the CA OPS/MVS Interface, the rewording of this message will be displayed in the function window with a default name of CA-OPS/MVS Messages (window type of OPSMSG).

Examples:

- The following is the JES2 $HASP373 message:

  ```
  $HASP373 TESTJOB STARTED - INIT 1 - CLASS C - SYS SYSA
  ```

  To remove the last three words from the message and make it fit on one line, write the following rule:

  ```
  MSGID($HASP373), WHEN (&JOBID(1:3) EQ JOB),
  REWORD(&WORD1 &WORD2 &WORD3 &WORD4 &WORD5 &WORD6 &WORD7
  &WORD8 &WORD9)
  ```

  After the rule executes the message, the message appears like this:

  ```
  $HASP373 TESTJOB STARTED - INIT 1 - CLASS C
  ```

- A user enters the following command in the Command Window from the CA Automation Point Desktop:

  ```
  SLIP0C1
  ```

  The following rule rewords this pseudo command, and issues the complex z/OS command to the console instead:

  ```
  CMDIN(SLIP0C1), REWORD(SLIP SET, ID=P0C1, COMP=0C1, A=SVCD, MATCHLIM=1,
  JOBNAME=TESTPGR, END)
  ```

SUPPRESS Keyword

Specify the SUPPRESS keyword to prevent CA Automation Point from displaying a message in the Merged Messages window, and in the Action or Normal Message Recall windows; or to instruct CA Automation Point to prevent a command from being sent to the console.

This keyword has the following format:

```
SUPPRESS
```

Usage Notes:

- If the message was issued by the Notification Server, this message will not appear in the function window with the default name of AP Notification Messages (window type of VOXMSG).
- If the message was received by the CA OPS/MVS Interface, this message will not appear in the function window with the default name of CA-OPS/MVS Messages (window type of OPSMSG).

Examples:

- RACF issues the following messages to indicate when a user last logged on with a user ID:

  ICH70001I *userid* LAST ACCESS AT *date*

  The following rule suppresses message ICH70001I:

  MSGID(ICH70001I), SUPPRESS

- A user enters the following command, trying to stop CA OPS/MVS, in the Command dialog from the Remote Viewer:

  STOP OPSS

  The following rule prevents this z/OS STOP command from being sent to the console, and issues a write-to-operator message to the CA Automation Point message console.

  CMDIN(STOP), WHEN(&WORD2 EQ 'OPSS'), SUPPRESS, WTXC (Nobody is authorized to stop CA OPS/MVS)

WTO Keyword

The WTO keyword issues a write-to-operator message.

This keyword has the following format:

WTO(*text*)

text

Specifies the text of the WTO message.

Usage Note:

If an active message caused the WTO message to be issued, the WTO message appears in the Action Message Recall window and in the action message area of the Merged Messages window. Otherwise, the WTO message appears in the Normal Message Recall window and the main message area of the Merged Messages window.

Example:

The following message indicates that an improper device was entered in response to a device allocation recovery request:

IEF490I JOB1--INVALID REPLY

The following rule clarifies this message by issuing an additional message:

```
MSGID(IEF490I),
 WTO(ENTER a DEVICE FROM ORIGINAL DEVICE LIST)
```

WTXC Keyword

The WTXC keyword issues a write-to-operator message to the CA Automation Point message console. Rules process all messages issued by the WTXC keyword. WTXC messages issued from rules are not processed through rules. (WTXC messages issued from REXX programs *are* processed through rules.)

This keyword has the following format:

WTXC(*msgtext*)

msgtext

Specifies the value of *msgtext.* Can be any valid text string.

Example:

The following message notifies the operator that CA Automation Point has successfully initialized the SYSA processor. CA Automation Point does not process the message through rules.

```
MSGID(IPLSYSA)
 WTXC(IPL of SYSA processor is complete.)
```

Keywords for Logging Messages

The following sections describe the CA Automation Point keywords used to log messages.

LOG Keyword

The LOG keyword tells CA Automation Point to write a message to the host message log file.

This keyword has the following format:

LOG

Example:

The following message indicates the expiration date for the SMR product:

SMR9920 WARNING--SMR EXPIRES *date*

The following rule sends message SMR9920 to the host message log file:

MSGID(SMR9920), LOG

NOLOG Keyword

Specify the NOLOG keyword to tell CA Automation Point not to place a message in the host message log file. NOLOG is the default.

This keyword has the following format:

```
NOLOG
```

Example:

The following rules send all CICS messages (except messages that begin with CICS99) to the host message log file:

```
MSGID(CICS), LOG
MSGID(CICS99), NOLOG
```

NOPRINT Keyword

Specify the NOPRINT keyword to tell CA Automation Point not to print a message to the hardcopy log.

This keyword has the following format:

```
NOPRINT
```

Example:

The following rules send all CICS messages (except messages that begin with CICS99) to the hardcopy log:

```
MSGID(CICS), PRINT
MSGID(CICS99), NOPRINT
```

PRINT Keyword

Specify the PRINT keyword to tell CA Automation Point to print a message to the hardcopy log.

This keyword has the following format:

```
PRINT
```

Usage Notes:

- To activate the hardcopy log, open the Message Logging Settings window in the Configuration Manager and enable the Print a Hardcopy Log of Console Messages option.

- If your workstation cannot communicate with the printer because the printer is not online, the workstation issues a message informing you of the problem. Paper jams and running out of paper are two common reasons that printers go offline. Do not specify the PRINT keyword unless an operator is nearby to oversee the printer.

Example:

The following message indicates that all SYS1.DUMP data sets are full:

```
IEA994E  ALL SYS1.DUMP DATA SETS ARE FULL
```

The following rule writes message IEA994E to the hardcopy log:

```
MSGID(IEA994E), PRINT
```

Keywords for Notification

The following sections describe the CA Automation Point keywords used for notification.

ALARM Keyword

The ALARM keyword tells CA Automation Point to sound an audible alarm (a rising or rising-and-descending tone). By default, CA Automation Point sounds an alarm for all highlighted messages.

This keyword has the following format:

ALARM(*alarmnumber*)

alarmnumber

Specifies alternate alarms. This value is a number from 1 to 20. If the number is 10 or less, the alarm consists of the specified number of rising tones. If the number is between 11 and 20, the alarm consists of a rising tone followed by a descending tone, repeated one to ten times depending on the number specified. For example, ALARM(20) sounds an alarm tone that rises and descends ten times.

You can use some of the alternate alarms for more serious problems so that they stand out. However, try not to use more than five different alarms because the differences between too many alarms are difficult to distinguish.

When you specify no *alarmnumber* value, you get a single beep, which is the standard alarm.

Usage Notes:

- To activate the ALARM facility, do the following:
  1. From the Configuration Manager, choose Expert Interface, Automation, Automation Point Desktop Settings.
  2. Select LOCAL Only from the Sound Audible Alarms pull-down list.

     The ALARM facility is activated.
- You cannot use a status variable as the *alarmnumber* operand.

Example:

This message indicates that the WTO buffer is 80 percent full:

```
IEA405E  WTO BUFFER SHORTAGE--80% FULL
```

The following rule alerts the operator to this problem using a tone that rises and descends five times:

```
MSGID(IEA405E), ALARM(15)
```

ALARMSAY Keyword

The ALARMSAY keyword tells CA Automation Point to issue a text-to-speech alarm.

This keyword has the following format:

ALARMSAY(*alarmtext*)

alarmtext

Specifies the content of the text-to-speech alarm.

Usage Note:

Use the CA Automation Point desktop setting, Sound Audible Alarm, to activate the ALARMSAY facility.

Examples:

- The following message indicates that the WTO buffer is 80 percent full:

  ```
  IEA405E WTO BUFFER SHORTAGE 80% FULL
  ```

  The following rule alerts the operator to this problem:

  ```
  MSGID(IEA405E), ALARMSAY(WTO Buffer shortage 80% full)
  ```
- The following rule enables a Speak pseudo-command to be issued from CA Automation Point:

  ```
  CMDIN(Speak), SUPPRESS, ALARMSAY(&CMD(6:255)), WTXC(&CMD)
  ```

NOALARM Keyword

Specify the NOALARM keyword to prevent CA Automation Point from sounding an audible alarm when a message is displayed. This keyword overrides the default, which is to sound an alarm for any highlighted messages. It also silences the alarm for a message for which you specified the ALARM keyword.

This keyword has the following format:

NOALARM

Example:

The following message indicates the expiration date for CA JMR:

JMR9920 WARNING--JMR EXPIRES *date*

The following rule suppresses any alarms associated with this message:

MSGID(JMR9920), NOALARM

Keyword for Controlling Commands

The CMDIN keyword defines the start of a command rule.

CMDIN Keyword

Specify the CMDIN keyword to define the start of a command rule.

This keyword has the following format:

CMDIN(*characters*)

characters

Specifies the name of a command that you want CA Automation Point to process.

Usage Notes:

- CA Automation Point processes CMDIN rule clauses in this order: CMDIN(), CMDIN(*string*), CMDIN(*). CMDIN() is best used to set global defaults, and CMDIN(*) is best for global override.
- You can specify multiple commands by specifying the portion of the command that indicates a specific group of commands. For example, specifying CMDIN(STOP) tells CA Automation Point to process all commands with identifiers beginning with STOP. Specifying CMDIN() or CMDIN(*) causes CA Automation Point to process all messages.

Examples:

- The following rule prevents any remote user, other than user SYSADMIN, from using an z/OS STOP command to stop CA OPS/MVS, and issues a write-to-operator message to the CA Automation Point message console.

  ```
  CMDIN(STOP),
    WHEN(&WORD2 EQ 'OPSS' AND &USER NE '' AND &USER(1:9) NE 'SYSADMIN@'),
    SUPPRESS, WTXC (&USER is not authorized to stop CA OPS/MVS)
  ```

- The following rule enables an operator to get a complete list of the outstanding replies using the DRL pseudo-command, which is converted to the D R, L, CN=(ALL) command.

  ```
  CMDIN(DRL), REWORD(D R, L, CN=(ALL))
  ```

- The following rule rewords a complex z/OS command so that the operator can issue the pseudo command SLIP0C1 instead of the complete z/OS command.

  ```
  CMDIN(SLIP0C1), REWORD(SLIP SET, ID=P0C1, COMP=0C1, A=SVCD, MATCHLIM=1,
  JOBNAME=TESTPGR, END)
  ```

Keywords for CA NSM Event Traffic Controller

The following sections describe the CA Automation Point keywords used to work with CA NSM.

NOUNIFWD Keyword

Specify the NOUNIFWD keyword to prevent CA Automation Point from forwarding a message that is processed by rules to CA NSM.

This keyword has the following format:

```
NOUNIFWD
```

Example:

The following rule prevents CA Automation Point from forwarding a message beginning with IEA405E to CA NSM:

```
MSGID(IEA405E), NOUNIFWD
```

SNMPTRAP Keyword

SNMPTRAP keyword sends an SNMP trap to the specified host.

This keyword has the following format:

SNMPTRAP(HOST(*HostName*|*) [COMMUNITY(*CommunityName*)]
GTRAP (*GenericTrapNum*) STRAP (*SpecificTrapNum*)
DATA(OID, *type*, *value*)[DATA(OID, *type*, *value*)]...)

HOST

Specifies the name of the host to which to send the trap.

Note: Specifying an asterisk (*) sends an SNMP trap to all host names designated for message forwarding in the Configuration Manager.

COMMUNITY

Specifies the SNMP community to which to send the trap. This option may be used to override the default of PUBLIC.

GTRAP

Defines the class of generic trap being sent. This value is a single digit, in the range of 0 to 6. Under most circumstances, use code 6 to indicate that a system-specific SNMP trap code is being used.

The following codes numeric codes have specific industry standard meanings as predefined by the Internet Activities Board (IAB):

0

Indicates a coldstart.

1

Indicates a warmstart.

2

Indicates a link down.

3

Indicates a link up.

4

Indicates an authentification failure.

5

Indicates EGP neighbor loss.

STRAP

Specifies a system-specific trap number up to a 32-bit integer.

DATA

Specifies the type and value of the data:

type

Specifies the type of the trap

value

Specifies the value of the data

You can repeat this operand up to 20 occurrences.

Example:

The following example sends the enterprise-specific trap number 1 to the host named UNIHOSTA. The trap information is related to an SNMP MIB (management information base) object whose OID (object identifier) is 999.2.1.3. The integer data value being reported for the OID is 777.

```
SNMPTRAP(HOST(UNIHOSTA) GTRAP(6) STRAP(1) DATA(999.2.1.3,INTEGER,777))
```

UNICMD Keyword

The UNICMD keyword tells the CA NSM Event Manager, which resides on the specified host, to execute the supplied command.

This keyword has the following format:

UNICMD(HOST(*HostName*) COMMAND(*CommandString*))

HOST

Specifies the name of the host on which the CA NSM Event Manager resides.

COMMAND

Specifies the command to be executed by the CA NSM Event Manager.

Example:

The following example copies a file from the temporary directory to an application directory on the remote Windows host named UNIHOSTA:

```
UNICMD(HOST(UNIHOSTA) COMMAND(COPY C:\TEMP\YOURDATA.DAT
                C:\YOURAPP\YOURDATA.DAT))
```

UNIFWD Keyword

The UNIFWD keyword tells CA Automation Point to forward a message that is processed by rules to all recorded CA NSM hosts.

This keyword has the following format:

```
UNIFWD
```

Example:

The following rule tells CA Automation Point to forward a message beginning with IEA405E to CA NSM:

```
MSGID(IEA405E), UNIFWD
```

UNIWTO Keyword

The UNIWTO keyword sends the supplied message to the CA NSM Event Manager on the specified host.

This keyword has the following format:

```
UNIWTO (HOST(HostName|*)
  MESSAGE(MessageString)[ORIGHOST(OrigHostName)]
  [SEVERITY(I|W|E|S|F)])
```

HOST

Specifies the name of the host on which the receiving CA NSM Event Manager resides.

Specifying an asterisk (*) sends the supplied message to all host names designated for message forwarding in Configuration Manager.

MESSAGE

Specifies the text string of the message being sent to the CA NSM Event Manager.

ORIGHOST

Specifies the name of the host on which the message originates.

Default: The name of the current session.

SEVERITY

Specifies the severity of the message.

I=Informational

W=Warning

E=Error

S=Success

F=Failure

Default: E, if the message is an action message, otherwise I.

Example:

The following example sends a message to the CA NSM Event Manager residing on the host named UNIHOSTA:

```
UNIWTO(HOST(UNIHOSTA) MESSAGE(HELLO EVENT MANAGER))
```

Keyword for Interfacing with Third-Party Applications

This section describes the keyword that allows interaction with third-party software.

EXPORTMSG Keyword

The EXPORTMSG keyword tells CA Automation Point to send the message to an ApExportMsg function created by the user, which then forwards messages to a third-party software application.

This keyword has the following format:

EXPORTMSG (MESSAGE(*MessageText*) [SOURCE(*SourceOfMsg*)] [SEVERITY(*OneCharSevCode*)] [APPTEXT(*ApplicationText*)])

MESSAGE

Specifies the text of the message to be sent to the ApExportMsg function. For example, the CA Automation Point variable &MSG could be supplied as this parameter.

SOURCE

Specifies the name of the source of the message. If the SOURCE parameter is not supplied when the EXPORTMSG command is issued from CA Automation Point rules, the name of the session from which the message came is used as a default.

SEVERITY

Specifies a one-character severity code whose use is defined by each site. The implementation of ApExportMsg can treat severity in any fashion appropriate for the site. If the SEVERITY parameter is not supplied when the EXPORTMSG command is issued from CA Automation Point rules, a default value of E is used if the exported message is an action message; otherwise the default value is I. The default value is always I when the EXPORTMSG command is issued from a REXX program.

APPTEXT

Specifies the application text specific to your implementation of ApExportMsg. It enables you to pass application-specific text from CA Automation Point rules into ApExportMsg. For example, you may have implemented ApExportMsg to send messages to any one of several third-party applications. The contents of the APPTEXT parameter could be used to indicator to which of these third-party applications the message is to be sent.

Usage Note:

To use this command, an APOPTIONS.DLL must be created containing a function named ApExportMsg. See the EXPORTMSG.TXT file in the CA Automation Point SAMPLE\EXPORTMSG directory for details on this functionality.

Example:

Suppose that CA Automation Point receives the following message from CA OPS/MVS on the host:

```
OPS000I REXX IPL9672
```

The following sample rule invokes EXPORTMSG, which passes the message to ApExportMsg (), which provides interaction with third-party software:

```
MSGID (OPS000I REXX IPL9672)
        EXPORTMSG (MESSAGE(&MSG) SEVERITY(E) SOURCE(AP3672) APPTEXT(IPL STATE))
```

The ApExportMsg function would take this information and make a call to the third-party software. The third-party software would evaluate the call and return a value to ApExportMsg. The ApExportMsg function would then either return a zero to CA Automation Point with no message or return a non-zero return code and an error message that would be recorded in the ASOTRACE.LOG.

Script Keywords Summary

This section describes the keywords that you need to define when setting up script files. For details about setting up these kinds of files, see "Customizing Special CA Automation Point Files" in the *Administrator Guide*.

Use the following keywords to set up script files:

ENDSEARCH

Terminates the preceding SEARCH.

ERROR

Specifies what happens if a search fails.

KEY

Sends a keystroke string to the target session.

SEARCH

Searches the entire screen for a string of data.

WAIT

Causes the script to wait for the preceding keystroke string to be accepted by the system.

XKEY

Sends a single CA Automation Point operation instruction to a host session, regardless of whether the keyboard is locked.

Script Keyword Syntax

Follow these rules when writing statements in a script file:

- Begin statements in any column.
- Do not continue statements onto the next line. The maximum line length is 256 characters.
- Use only one script keyword per line.
- Begin comment lines with an asterisk (*).

CA Automation Point executes script statements line by line. Each time a script executes, CA Automation Point evaluates all SEARCH statements in the order that they occur in the script.

Script Keyword Descriptions

The following sections describe the CA Automation Point script keywords.

ENDSEARCH Keyword

The ENDSEARCH keyword terminates a search initiated by a SEARCH keyword.

This keyword has the following format:

```
ENDSEARCH
```

Example:

When an MCS console is active and operating correctly, it displays a message similar to one of these:

```
IEE152 ENTER CANCEL D C,K
IEE612I CN=05 DEVNUM=B4C SYS=S028 CMDSYS=S028
```

Suppose that your script contains these statements:

```
ERROR=IGNORE
*
SEARCH=(IEE152I)
 KEY=(K S,DEL=N,SEG=20,MFORM=(T,J)@E)
ENDSEARCH
*
SEARCH=(IEE612I)
 KEY=(K S,DEL=N,SEG=20,MFORM=(T,J)@E)
ENDSEARCH
```

The statements take the following actions:

1. The first SEARCH statement tells CA Automation Point to search for the IEE152I message.
2. If the first SEARCH statement finds message IEE152I, the KEY statement executes, putting the console in nondelete mode, setting the segment length to 20 lines, and so on; if the SEARCH operation does *not* locate the IEE152I string, the ERROR statement tells CA Automation Point to continue processing.
3. The ENDSEARCH statement terminates the first search operation.
4. CA Automation Point executes the second SEARCH operation (for the IEE612I message) in the same way.

ERROR Keyword

The ERROR keyword lets you determine what happens if the following search fails to find the data. The ERROR keyword applies to all subsequent SEARCH statements until another ERROR statement replaces it.

This keyword has the following format:

ERROR=*action*

action

Valid values for *action* are:

ENDSCRIPT

If the search fails, this value tells CA Automation Point to stop processing the script. Use this value when you want the script to search the screen for error indicators. If none are found, the script is complete and can be terminated.

ENDSESSION

If a search fails, this value tells CA Automation Point to stop processing the script, stop controlling the session, stop monitoring messages, and begin action to clear a console error. Consequently, session automation is suspended for a period specified by the Post-Error Restart Automation setting for the session. After the wait period, CA Automation Point automatically restarts the session to restore session automation. You can use this value in console initialization scripts before searching for a text string that must be present to initialize the session.

IGNORE

If the search fails, this value tells CA Automation Point to ignore all subsequent statements in the current search until it finds the matching ENDSEARCH statement, and then to continue executing the remainder of the script.

Default: ENDSCRIPT

KEY Keyword

The KEY statement lets you send a *keystroke* string to the target session. To send normal text characters, it is not necessary to use special abbreviations; however, you must type the characters in the same case that you want CA Automation Point to send them.

This keyword has the following format:

KEY=(*keystrokes*)

keystrokes

Specifies the keystroke string to be sent to the target session. This value ends with @E (the ENTER operation), unless you are sending a PF or PA key.

Note: For a list of valid keyboard operations, see the *Administrator Guide*.

Example:

The following KEY statement sends a keystroke string to the session that deletes the display area on an MCS console:

```
KEY=(K A,NONE@E)
```

SEARCH Keyword

The SEARCH statement lets you search the entire screen for a specific string of data. Any statements that follow, up to the corresponding ENDSEARCH, execute only if the search is successful. You can nest SEARCH statements.

This keyword has the following format:

SEARCH=(*text*)

text

Specifies the string of data that CA Automation Point looks for on the screen

Example:

The following SEARCH statement tells CA Automation Point to search for message IEE152I:

```
SEARCH=(IEE152I)
```

WAIT Keyword

The WAIT keyword specifies the time, in seconds, to wait after processing a preceding statement.

This keyword has the following format:

WAIT=(*time*)

time

Specifies the number of seconds to wait. Specify 0 through 32767

Example:

The following is an example sets a wait time of 5 seconds.

```
WAIT=5
```

XKEY Keyword

The XKEY keyword lets you send a CA Automation Point operation instruction to a host session while that session is in X state (that is, unable to accept instructions entered from the keyboard).

This keyword has the following format:

XKEY=(*instruction*)

instruction

Specifies the name of the keyboard operation (for example, POWER_RESET) that you want the session to receive.

Notes:

- The @ character must always precede the keyboard operation name, and the name must be enclosed in quotation marks (@"*key_opt"*). The entire operation instruction, including the @ character, must be enclosed in parentheses.
- For a list of valid keyboard operations, see the *Administrator Guide*.

Example:

The following XKEY statement sends a RESET keystroke to a session that is currently in X-state.

```
XKEY=(@"RESET")
```

Chapter 4: ADDRESS AXC Commands

CA Automation Point has its own built-in command processors that you can issue to perform specific tasks, such as issuing z/OS commands to an automated session or sending messages from non-automated sessions through CA Automation Point rules.

ADDRESS AXC Command Summary

The following sections summarize CA Automation Point ADDRESS AXC commands.

Commands for Automation Processing Data

Use the following commands to get data for automation processing.

CLOSEBUF

Closes the internal host message stream buffer (created by a previously issued OPENBUF command) for a specified asynchronous session

GETSCRN

Captures screen images from a session and stores information about that session in variables

OPENBUF

Opens an internal host message stream buffer for a specified asynchronous session

READBUF

Reads a host message from the internal buffer of a specified asynchronous session

Commands for Automation Tasks

Use the following commands to perform automation tasks.

DELVAR

Deletes a variable from the CA Automation Point variable table.

GETMSGI

Fetches the number of messages in the CA Automation Point action message recall list, the internal pointer to each action message in the list, and the text of each action message.

GETREXXL

Returns a list of active and queued REXX programs.

GETVAR

Copies the value of a CA Automation Point status variable into a local REXX variable.

GETVARL

Returns the names of all CA Automation Point status variables that match a specified name mask.

LOADRULES

Enables a new CA Automation Point rules file dynamically. The dynamically enabled rules file replaces the current rules file.

MSG

Allows you to indirectly process messages through rules, even in sessions that you cannot normally automate with rules. The messages appear to come from a session that you specify.

SESSCNTL

Establishes or closes connections, pauses or restarts automation for a single specified session. This command can also control the frequency of background window updates and change the host to which an automated Telnet session is connected.

SETVAR

Sets the contents of a CA Automation Point status variable to a given value, changes the value of a variable, or creates a new variable

STOPREXX

Stops an executing REXX program or cancels execution of a queued REXX program.

WAIT

Causes a REXX program to wait for a specified number of seconds before processing the next REXX statement.

Commands for REXX-related Operations

Use the following commands for REXX-related operations.

REXX

Queues a REXX program for processing by CA Automation Point

SCRIPT

Invokes a script from a REXX program

SESSCMD

Sends a text string (such as a command) to a specified session and retrieves a screen image from that session after it has processed the text string

Commands for Utilities

Use the following commands to perform the corresponding utility tasks.

DOM

Deletes an action message from the action message area of the Merged Messages window and from the Action Message Recall window (by deleting it from the CA Automation Point action message recall list).

PLOT

Draws a line or bar graph of system resources.

SESSCONFIG

Populates a REXX stem variable with information about sessions configured in a particular session definition set.

SESSLIST

Populates a REXX stem variable with information about the current state of sessions running on the AP Desktop.

WTO

Displays a text string in the normal message area of the Merged Messages window and in the Normal Message Recall window.

WTOH

Displays a text string in the action message area of the Merged Messages window and in the Action Message Recall window.

WTXC

Displays a text string in the AP Messages window and in the AP Message Recall window. (CA Automation Point rules can process messages issued with this command.)

ADDRESS AXC Command Syntax

Follow these rules when invoking ADDRESS AXC commands:

- Use double ampersands (&&) when you include an ampersand in the text to be sent. To send the text ABC&D, issue the SESSCMD command processor as follows:

  ```
  "SESSCMD 'ABC&&D"'
  ```

- Use a single ampersand (&) before the name of a variable in your REXX program to instruct CA Automation Point to evaluate the variable. For example:

  ```
  myvar = "This is a test msg."
  "WTOH '&myvar"'
  ```

  displays the following message in the Action Message area:

  ```
  This is a test msg.
  ```

- The usual delimiter for the positional operands in the MSG, REXX, SESSCMD, WTO, WTOH, and WTXC command processors is the single quote ('). If the value that you are specifying contains single quotes, you can use any of the following delimiter characters:

  ```
  ~ ! @ # $ % ^ * _ - + = " < > , . / ] \
  ```

  The following command sends a message containing an embedded single quote:

  ```
  "WTXC #Time's up!# "
  ```

- Do not exceed the 256-character maximum line length when invoking a command processor. Be especially careful if you are using many variables or variables that have very long values.
- Operands shown in brackets ([ ]) are optional.

Return Codes from Command Processors

Some CA Automation Point command processors generate a return code when you issue them:

- A command processor executed from REXX assigns the return code value to the REXX variable RC.
- A command processor invoked from a rule places the return code value in the CA Automation Point variable AXCRESULT.
- If the return code is a negative number (such as -2), REXX traces the line in the REXX program that triggered the return code.

RC	Meaning
30	You have tried to address an invalid or inactive command environment (using an ADDRESS statement in your REXX program).
-8	There was not enough memory to process the command.
-7	The statement invoking the command processor contained an invalid session name or type, or the session does not support the requested operation.
-6	The statement invoking the command processor was missing a required argument.
-5	The statement invoking the command processor contained an invalid argument.
-4	The statement invoking the command processor contained too many arguments or mutually exclusive arguments.

RC	Meaning
-2	CA Automation Point encountered an invalid delimiter while parsing the command. The problem occurs when CA Automation Point expects to see a certain character as a delimiter and finds another character instead. For example, using this statement in a REXX program produces a return code of -2 because CA Automation Point expects parentheses (not single quotes) to enclose the text string. "SETVAR REXX_MESSAGE 'THIS IS A REXX MESSAGE'" CA Automation Point recognizes the following delimiters if you issue its commands from REXX programs: ■ Single quotation marks ■ Parentheses ■ Spaces not contained in quotation marks or parentheses ■ The end of the line This return code could also indicate that you have exceeded the maximum length of a variable name in a command processor statement.
-1	CA Automation Point does not recognize an operand in the command processor, probably because its name is misspelled. CA Automation Point also returns this code if you issue operating system commands without specifying CMD as the target environment. If you issue a command through a CA Automation Point REXX program, you must use the REXX ADDRESS command to target commands to other REXX environments because CA Automation Point becomes the default environment to REXX. For example, to use the operating system to copy a file, issue a command that looks like this: ADDRESS CMD COPY filename1 filename2
0	The command processor executed without errors.
3	An error occurred when processing the command: ■ The LOADRULES command generates a return code of 3 if the rule file cannot be opened. ■ READBUF, CLOSEBUF, and OPENBUF commands generate a return code of 3 if a buffer does not exist for the session that you specify, or if another REXX program currently owns the specified session's buffer. ■ The SESSCNTL command generates a return code of 3 if you attempt to restart automation in a session while global automation pause is in effect. ■ The SESSCONFIG command generates a return code of 3 if you specify an invalid session definition set name or the associated file cannot be opened.

RC	Meaning
9	The LOADRULES command processor did not enable the rules file that you specified because one or more rules in the file contained syntax errors.
32	The SESSCMD command processor timed out because the keyboard in the specified session was locked (in an X state) and did not clear soon enough.
36	A SESSCMD command locked the host keyboard and the keyboard did not clear soon enough; the host could not respond before SESSCMD timed out.
150	The PLOT command referenced an undefined graph. Be sure that you have specified the correct plot and that you have spelled it correctly.
151	The PLOT command attempted to define a line or bar that already exists.
152	The PLOT command attempted to delete a line or bar that does not exist.
153	You specified an invalid range in a PLOT command statement. For example, you may have specified a minimum value that exceeded a maximum value.
154	You tried to define more than eight lines in a PLOT command statement.
155	You tried to define more than one bar in a PLOT command statement.
156	The PLOT command referenced an invalid or undefined TICK value.
157	You defined an illegal tick mark.
158	You specified an invalid scale in a PLOT command statement. For example, you may have specified a start value that exceeded a stop value, or you may have tried to apply a TIME scale to an axis defined as NUMERIC.
159	The PLOT command exceeded the maximum number of label lines for an axis or title label.
303	An incorrect key operation was specified.

Commands for Automation Processing Data

The following sections describe the ADDRESS AXC commands used for automation data processing.

CLOSEBUF Command

The CLOSEBUF command closes an automated asynchronous session's host message stream buffer. (The OPENBUF command opens a buffer for an asynchronous session.)

This command has the following format:

"CLOSEBUF SESSION(*sessname*)"

SESSION

Specifies the name of a session (*sessname*) that has a message stream buffer that you want to terminate. The session must be an *automated* asynchronous session.

Usage Notes:

- Always issue a CLOSEBUF command to explicitly close a buffer that you opened with a previously issued OPENBUF command.
- If the specified session is automated-that is, if you chose to automate the session when you defined the session- any unread host messages in the buffer are lost when you issue the CLOSEBUF command.
- Only the REXX program that opened the buffer for the specified asynchronous session can close that session's buffer. If another REXX program tries to close the buffer, the CLOSEBUF command generates a return code of 3.
- If your REXX program issues a CALL statement to call another REXX program, the called program *can* close the buffer.

GETSCRN Command

The GETSCRN command captures the screen image associated with a session and places information about that session in variables.

This command has the following format:

"GETSCRN SESSION(*sessname*) [SCREEN(YES|NO)] [PREFIX(*prefix*|LINE|NO)]"

SESSION

Specifies the name of the session (*sessname*) from which you want to capture the screen image.

SCREEN

(Optional) Specifies whether to create the REXX variable SCREEN to store the character portion of the screen image. This operand has two options:

YES

Creates the SCREEN variable.

NO

Does not create the SCREEN variable. Specifying SCREEN(NO) saves memory if you do not intend to use the SCREEN variable.

Default: YES

PREFIX

(Optional) Creates a stem variable to store the lines of a screen image returned to CA Automation Point. CA Automation Point stores each line of the screen in a separate variable and numbers the variables consecutively (for example, LINE.1, LINE.2, and so on).

Note: The variable numbered 0 (for example, LINE.0) always contains the total number of elements in the stem variable.

Valid PREFIX values are:

prefix

The name that you want to assign to the variables set by the PREFIX operand. For example, specifying PREFIX(ROW) causes CA Automation Point to create stem variables named ROW.0, ROW.1, ROW.2, and so on through ROW.*n*.

LINE

Tells CA Automation Point to create REXX variables named LINE1 through LINE*n* to store lines of a screen image.

NO

Does not create the screen variable. Specifying PREFIX(NO) saves memory if you do not intend to use the variables.

Default: LINE

Usage Notes:

- Some asynchronous console sessions have a scrolling display and sometimes generate multiple screens of output. For these sessions, it is better to open an internal message stream buffer and redirect the incoming messages to the buffer; your REXX program can then read the messages in the buffer and act upon each one as necessary. For more information about message stream buffering, see the description of the OPENBUF command (see page 79).
- For compatibility with earlier versions of CA Automation Point, the GETSCRN command also creates the old-style, non-stem variables. For example, suppose that the PREFIX variable is LINE. In addition to the stem variables described above, GETSCRN also creates the non-stem variables LINE1, LINE2, and so on through LINE*n*. In this case, the variable LINE (equivalent to the stem variable LINE.0) contains the total number of elements in the variable.
- When you issue the GETSCRN command from a REXX program, CA Automation Point captures the screen image and places information about the screen into the following REXX variables:

  CSRCOL

  Specifies the column in which the cursor is positioned.

  CSRPOS

  Specifies the number of positions the cursor is away from the upper-left corner of the screen.

  CSRROW

  Specifies the row containing the cursor.

  OIA

  Specifies the operator information area (OIA) (also called the status line).

  SCREEN

  Creates a copy of the text of the screen image, with each row laid end to end. This copy does not include attributes such as colors or field positions.

  SCRLEN.

  Specifies the number of lines on the screen.

SCRSIZE

Specifies the number of characters the screen can contain.

SCRSTAT

Specifies the status of the screen:

LOCKED-The terminal emulator for the session is currently in an X-state and will not accept any input from the keyboard.

UNLOCKED-The terminal emulator for the session is not currently in an X-state.

SCRWIDTH

Specifies the number of columns on the screen.

- Each time you issue the GETSCRN command, CA Automation Point creates SCREEN and LINE.*n* variables automatically unless you specify otherwise.

Examples--capture screen image:

- Suppose that you want to use a REXX program to capture a screen image from an MCS console session and place information about that screen in REXX variables. You can place a statement like the following in the REXX program:

  "GETSCRN SESSION(S008)"

  In the sample statement, S008 is the name of an MCS console session. Because this statement does not specify the PREFIX operand, CA Automation Point automatically stores lines of the returned screen image in REXX variables named LINE1 through LINE*n*.

  The sample REXX program described next uses the CA Automation Point GETSCRN and WTO commands to capture the contents of a screen and place them in a file called SCREEN.cap. To invoke the REXX program, issue the command shown below either manually or using a CA Automation Point rule:

  SC *sessname filename* [DEBUG]

  The command has these components:

  - SC specifies the name (SC.cmd) of the REXX program.
  - The *sessname* value is the name of the session supplying the screen to be captured.
  - The *filename* specifies the file name to hold the captured screen image. (The default file name is SCREEN.cap.)
  - DEBUG is an optional operand that activates the REXX trace facility for debugging purposes.

- CA Automation Point provides a sample REXX program, SC.cmd, for capturing a screen image. The following example shows the text of the SC.cmd program. Besides the GETSCRN command, the program also uses the SESSCMD and WTO commands.

```
/* This REXX program captures a screen image. */
PARSE UPPER ARG . '"' TEXT '"' .   /* Get the message text.  */
IF DEBUG \= '' THEN TRACE R        /* Activate REXX trace.   */

PARSE UPPER ARG P1 P2 .           /* Parse off positional arguments */
IF P1='"' | P1='DEBUG'            /* If first arg not there,       */
THEN SESSNAME ='SESSION'          /* Then set default session name  */
ELSE SESSNAME = P1                /* Else, use 1st argument        */

IF P2='"' | P2="DEBUG"            /* If second argument not there   */
THEN FILENAME = 'SCREEN.CAP'      /* Then set default filename      */
ELSE FILENAME = P2                /* Else, use 2nd arg as filename  */

1  "WTO 'SC.CMD active, session= &SESSNAME filename= &FILENAME"'
"GETSCRN SESSION(&SESSNAME) PREFIX(LINE) SCREEN(NO)"
IF RC \= 0
THEN SAY 'GETSCRN FAILED, RC='RC
ELSE DO
  TITLE = "SCREEN FROM SESSION " SESSNAME
  CALL LINEOUT FILENAME, TITLE
    DO I = 1 TO SCRLEN            /* Loop through each row */
    CALL LINEOUT FILENAME, LINE.I   /* Write line to disk   */
    END
  CALL LINEOUT FILENAME           /* Close the file       */
  END

2  "WTO 'SC.CMD COMPLETE' "
EXIT
```

The WTO commands in SC.CMD issue write-to-operator messages as the GETSCRN command captures the screen image. The WTO command statement indicated by **1** in the example above alerts the operator when screen processing begins, and the statement indicated by **2** notifies the operator when SC.CMD finishes executing.

OPENBUF Command

The OPENBUF command opens an internal host message stream buffer for a specified automated asynchronous session and redirects the session's incoming messages to the buffer. The REXX program that opened the buffer can then read the messages from the buffer and act upon each message.

Note: Message stream buffering is useful for interpreting command responses from the asynchronous host console, especially those generating multiple screens of output.

This command has the following format:

"OPENBUF SESSION(*sessname*) [WAIT(*waittime*)]"

SESSION

Specifies the session (*sessname*) for which you want to create the message stream buffer. It must be an *automated* asynchronous session.

WAIT

(Optional) Specifies the number of seconds (*waittime*) -from 0 to 100000-that the OPENBUF command waits if another REXX program currently owns the buffer for the specified session.

If the *waittime* value expires before the OPENBUF command executes, it generates a return code of 3.

Default: 0

Usage Notes:

Keep the following items in mind when using the OPENBUF command:

- The REXX program issuing the OPENBUF command owns the specified session's buffer.
- Only the REXX program that opens the buffer can read or close the buffer. The READBUF command reads messages from the buffer and the CLOSEBUF command closes the buffer.
- If your REXX program issues a CALL statement to call another REXX program, the called program *can* read or close the buffer.
- If the session is automated-if you chose to automate the session when you defined the session-incoming host messages are no longer processed by CA Automation Point rules and do not appear in CA Automation Point function windows until your REXX program closes the buffer.
- Always issue a CLOSEBUF command to explicitly close a buffer that you opened with a previously issued OPENBUF command.

READBUF Command

The READBUF command reads the next message from an automated asynchronous session's internal message stream buffer (if one exists). Upon completion, the message is removed from the buffer. When the READBUF command returns a message with no lines, the buffer is either empty or the message has not yet arrived.

This command reads one message at a time from the session buffer. If a message exists in the session buffer when this command is issued, that message is placed into the Line.1 stem variable when using the default prefix of LINE.

This command has the following format:

"READBUF SESSION(*sessname*) [PREFIX(*prefix*|LINE|NO)] [WAIT(*waittime*)]"

SESSION

Specifies the session (*sessname*) that has a message buffer from which you want to read the host messages. The session must be an *automated* asynchronous session.

PREFIX

(Optional) Creates a stem variable to store the message read from the buffer. CA Automation Point stores each line of the message in a separate stem variable and numbers the variables consecutively (for example, LINE.1, LINE.2, and so on).

Note: The stem variable numbered 0 (for example, LINE.0) always contains the total number of message lines stored in the variable. (So if LINE.0=0, then the buffer was empty when your REXX program issued the READBUF command.)

Valid PREFIX values are:

prefix

Specifies the name that you want to assign to the stem variable. For example, specifying PREFIX(ROW) causes CA Automation Point to create a set of variables named ROW.0, ROW.1, ROW.2, and so on through ROW.*n*.

LINE

Tells CA Automation Point to create stem variables named LINE1 through LINE*n* to store lines of the buffer.

NO

Discards the message.

Default: LINE

WAIT

(Optional) Specifies the number of seconds (*waittime*)-from 0 to 100000-that the READBUF command waits if the buffer is empty.

If the *waittime* value expires before a message arrives in the buffer, the READBUF command generates a return code of 3.

Default: 0

Usage Notes:

Keep the following points in mind when using the READBUF command:

- Only the REXX program that opened the buffer can read from the buffer. (The OPENBUF command opens a buffer for an asynchronous session.)
- If your REXX program issues a CALL statement to call another REXX program, the called program *can* read from the buffer.
- If a buffer does not exist for the session that you specify, or if another REXX program currently owns the specified session's buffer, the READBUF command generates a return code of 3.
- You can direct messages that the REXX program reads from the buffer to CA Automation Point rules by issuing the MSG command.

Commands for Automation Tasks

The following sections describe the ADDRESS AXC commands used for automation tasks.

DELVAR Command

The DELVAR command deletes a status variable from the CA Automation Point variable group.

This command has the following format:

"DELVAR *varname*"

varname

Specifies the name of the CA Automation Point variable to be deleted.

Usage Notes:

- DELVAR status variable names are case-sensitive in CA Automation Point. Use uppercase variable names consistently.
- Deleting a non-existent variable (a variable that has not been previously set) is always considered successful with RC=0.

Example:

To delete the status variable MSG_STATUS from the variable table, issue this command:

```
"DELVAR MSG_STATUS"
```

GETMSGI Command

The GETMSGI command fetches the internal pointer to an action message and the actual message text.

This command has the following format:

```
"GETMSGI"
```

Usage Notes:

When you invoke the GETMSGI command, it populates the following REXX variables as shown:

REXX Variable	Contents After Invoking GETMSGI
DOMID.0	The number of messages in the CA Automation Point action message recall list. This variable contains a value from 0 through 100.
DOMID.*n*	The internal pointer to message number n in the action message recall list.
DOMID_MSG.*n*	The text for action message number n in the action message recall list.

After invoking the GETMSGI command, you can use the DOM command to delete specific action messages from the CA Automation Point action message recall list.

Example:

Suppose that you want to delete all of the action messages in the CA Automation Point action message recall list every night at midnight. You could set up a time rule to invoke a REXX program like the following one:

```
"GETMSGI"
DO CNT=1 TO DOMID.0 BY 1
  INTERPRET 'CUR_DOMID=DOMID.'CNT
  "DOM DOMID(&CUR_DOMID)"
END
```

GETREXXL Command

The GETREXXL command returns a list of active and queued REXX programs invoked from within CA Automation Point. (The GETREXXL command cannot detect REXX programs invoked *outside* CA Automation Point.)

This command has the following format:

```
"GETREXXL"
```

Usage Notes:

- Use the GETREXXL command with the STOPREXX command to stop active REXX programs or to prevent queued REXX programs from executing.
- The GETREXXL command stores the information it returns in the following REXX stem variables:

REXX Stem Variable	Contents After Invoking GETREXXL
GETREXXL.0	The number of active and queued REXX programs invoked from within CA Automation Point (in the default AXC environment).
GETREXXL.*n*	An internal pointer to REXX program number *n*, indicating the order in which the program was invoked in relation to the others.
GETREXXL_NAME.*n*	The name of REXX program *n* and its command line arguments.

Example:

Suppose that you want to stop all instances of a particular program. Your REXX code could look like this:

```
"GETREXXL"

/* GETREXXL populates GETREXXL.0 with the number   */
/* of active and queued REXX programs, the         */
/* GETREXXL_NAME.i variables with the name and     */
/* arguments of a program, and GETREXXL.i with the */
/* number used by STOPREXX to terminate the program*/

DO I = 1 TO GETREXXL.0
  parse upper value GETREXXL_NAME.i with rxname.
  IF rxname  = token THEN
  "STOPREXX "GETREXXL.I
END
```

GETVAR Command

The GETVAR command copies the value of a CA Automation Point status variable into a local REXX variable.

This command has the following format:

"GETVAR *varname* [*rexxvar*]"

varname

Specifies the name of the CA Automation Point variable to be copied.

rexxvar

(Optional) The name of the local REXX variable into which the CA Automation Point variable is copied. If you do not specify a REXX variable name, the value of the first CA Automation Point variable is copied into the REXX variable AXCRESULT.

Usage Note:

GETVAR status variable names are case-sensitive in CA Automation Point. Use uppercase variable names consistently.

Example:

To give the variable SCREEN_STATUS the same value as the status variable SCRSTAT, issue this command:

"GETVAR SCRSTAT SCREEN_STATUS"

Note: If you issue this command when the SCRSTAT status variable has the value LOCKED, the REXX variable SCREEN_STATUS is also set to LOCKED.

GETVARL Command

The GETVARL command returns the names and values of all CA Automation Point status variables in the variable pool matching a specified name mask. A REXX stem variable with a special prefix (GETVL.*n*) stores the returned status variable names.

This command has the following format:

"GETVARL *varnamemask*
[PREFIX(GETVL|*prefix*]
[MAX(99|*nnn*)]
[SORT(ASCEND|DESCEND)]
[TOKEN(0|&GETVLTK)]"

varnamemask

Specifies the variable name mask for the status variable name search. Follow these name mask guidelines:

- Name masks can contain up to 32 characters.
- The following wild card characters are valid:
  - ? or + replaces individual characters.
  - * replaces any number of trailing characters when placed as the last character in the name mask.

PREFIX

(Optional) Specifies the prefix of the REXX stem variable containing the status variable names that the GETVARL command returns.

For example, assume that the stem name is the default GETVL. Following standard stem-variable convention, the GETVL.0 variable contains the number of status variables returned. The variables GETVL.1 through GETVL.*n* each contain the name of a status variable.

Default: GETVL

MAX

(Optional) Specifies the maximum number of variables names to return. The maximum valid *nnn* value is 500.

Default: 99

SORT

(Optional) Sorts the returned status variable names alphabetically in ascending (ASCEND) or descending (DESCEND) order.

Default: ASCEND

TOKEN

(Optional) A marker that the GETVARL command uses when it cannot retrieve all of the status variables requested in a single operation (because the number of variable names specified by *varnamemask* exceeds the current MAX value).

If your GETVARL request cannot return all of the status variable names specified by *varnamemask* in a single operation, it sets the TOKEN variable &GETVLTK to mark the variable where the search should resume for the next GETVARL operation.

When you issue the GETVARL command for the first time, omit the TOKEN operand or set it to 0 (zero).

If a subsequent GETVARL operation is necessary, specify TOKEN(&GETVLTK) to retrieve the remaining status variable names.

Example:

The following REXX statements list every existing variable:

```
GETVLTK=0
DO FOREVER
 "GETVARL * PREFIX(GETVL) SORT(ASCEND) MAX(500) TOKEN(&GETVLTK)"
 IF GETVL.0=0 THEN LEAVE     /* No more variables */
 DO I = 1 TO GETVL.0
   SAY GETVL.I               /* Display variable names */
 END
END
```

Note: The DO FOREVER loop is needed to list more than 500 variables. Without it, this code only lists the first 500 variables because of the MAX(500) entry.

LOADRULES Command

The LOADRULES command enables a new CA Automation Point rules file dynamically (while CA Automation Point is running). A dynamically enabled rules file always *replaces* the current rules file.

This command has the following format:

"LOADRULES FILE(*rulesfilename*) [REPLACE({YES|NO|CLEAN})]"

FILE

Specifies the name of the CA Automation Point rules file (*rulesfilename*) that you want to enable.

REPLACE

(Optional) Specifies the condition under which you want CA Automation Point to enable the new rules file. Valid REPLACE values are:

YES

Loads, compiles, and enables the new rules file even if one or more rules in the new rules file contain syntax errors.

NO

Loads and compiles the new rules file, but *does not enable* the file. Specify NO to check for syntax errors in a new rules file.

CLEAN

Loads, compiles, and enables the new rules file *only* if all rules in the file are free of syntax errors.

Default: YES

Example:

Suppose that you want to enable a new rules file after 8:00 p.m. (when the second shift starts at your data center). Your REXX program may contain statements like these:

```
IF ((TIME('H')>15) & (2ND_RULES_LOADED=FALSE)) THEN
 DO
 "LOADRULES FILE(2NDSHIFT.RUL)"
 2ND_RULES_LOADED=TRUE
 1ST_RULES_LOADED=FALSE
 END
```

MSG Command

The MSG command allows you to *indirectly* process host messages through CA Automation Point rules, even in sessions where you cannot normally use rules (such as processor console or *any* sessions running full-screen applications). The MSG command creates a message block out of the passed text that, to CA Automation Point rules, *appears* to come from the session you specify.

This command has the following format:

"MSG '*message*' SESSION(*sessname*)
[ACTION(YES|NO]
[JOBID(*jobid*)]
[JOBNAME(*jobname*)]
[MONNAME(*sysname1*)]
[MONNUM(*sysnum*)]
[MONPRTY(*prioritylevel*)]
[MONTYPE(*alerttype*)]
[PREFIX(*prefix*)]
[REPLYID(*replynum*)]
[SYSNAME(*sysname2*)]
[TIME(*time*)]"

message

Specifies the text of the message that you want to send.

SESSION

Specifies the name of the session from which the message appears to originate. The session must be defined in the session definition set; you cannot use a dummy name.

Note: For CA Automation Point internal sessions, use the name that corresponds to appropriate CA Automation Point window:

AP Window	Internal Session Name
AP Message Recall	AXC
CA-OPS/MVS Messages	OPS
AP Notification Messages	VOX

ACTION

(Optional) Specifies whether the message sent is an action message.

Default: NO

JOBID

(Optional) Specifies the current JES job ID associated with the message. This operand is valid for session types ECS, MCS, RCS, and VM.

JOBNAME

(Optional) Specifies the job name of the address space that issued the message.

MONNAME

(Optional) Specifies the source system name for a DataFrame message.

MONNUM

(Optional) Specifies the source system number for a DataFrame message.

MONPRTY

(Optional) Specifies the priority level of the alert message by the DataFrame system.

MONTYPE

(Optional) Specifies the type of the alert message issued by the DataFrame system.

PREFIX

(Optional) Specifies a character string that controls the formatting of messages that appear in CA Automation Point function windows displaying message activity.

For more information, see the description of the Global Session Prefix field in the help for the Configuration Manager, Expert Interface, Automation, Session Definition Sets dialog.

REPLYID

(Optional) Specifies the reply number for the message (if the message to be sent is a WTOR message).

SYSNAME

(Optional) Specifies the source system name for a message.

TIME

(Optional) Specifies the current time for the workstation in the form *hhmmss*.

Usage Note:

The MSG command can send a message up to 494 characters long. A string longer than 494 characters causes a negative return code.

Examples:

- To send a test message that appears (to CA Automation Point rules) to come from session S028, issue the following command:

  "MSG 'AXC0001 THIS IS A TEST MESSAGE FROM S028' SESSION(SO28)"

- Suppose that you have defined a non-automated session named HP_01 and are controlling the session with REXX programs. Use the MSG command to send messages (appearing to originate from session HP_01) that CA Automation Point rules can process, as shown in these REXX program statements:

  "GETSCRN SESSION(HP_01)"
  "MSG '&LINE1' SESSION(HP_01) JOBNAME(SCREEN_4)"

  The example REXX fragment sends the first line of Session HP_01's current screen through rules for processing. The optional JOBNAME operand assigns a job name that your rules can use to further identify the message. For example, the following CA Automation Point rule recognizes the message with the SCREEN_4 job name and colors the message bright red in the Merged Messages window:

  MSGID(ALERT 10) WHEN(&JOBNAME EQ SCREEN_4) COLOR(BRIGHT RED)

SESSCNTL Command

The SESSCNTL command establishes or closes a connection, or pauses or restarts automation for a single specified session. This command can also control the frequency of background window updates and change the host to which an automated Telnet session is connected. This command cannot be used to control the Windows command prompt (VIO) or Event Traffic Controller (ETC) sessions.

This command has the following format:

"SESSCNTL {AUTOMATE(PAUSE|RESTART|STATUS) SESSION(*sessname*) |
CONNECTION (OPEN|CLOSE|STATUS) SESSION(*sessname*) |
BWUPDATE(*numsecs*) |
TELNETHOST(*hostname)* SESSION(*sessname*) }"

AUTOMATE

Specifies an automation pause or restart for the specified session. This operand has the following options:

PAUSE

Resumes automation for the specified session. This option has no effect if automation is already paused.

RESTART

Causes automation of the specified session. This option has no effect if:

- CA Automation Point is already automating the session
- CA Automation Point is globally paused
- CA Automation Point does not automate the session directly (that is, the session is non-automated because the session was not defined as automated in the session definition set)

STATUS

Returns the specified session's automation state. (The returned value is stored in the AXCRESULT variable.)

- YES: indicates that automation is active
- NO: indicates that automation is paused
- PAUSED: indicates that the session is automated and that automation is currently paused

CONNECTION

Issues a connect or disconnect for the specified session or query current connection status. This operation has the following options:

OPEN

Attempts to establish the connection to host for the specified session. Initiation of the connection is indicated by message AXC0556I. Connection behavior is dependent on the session type. Results of the connection for TN3270 or TN5250 sessions are reported by message AXC1800I. For other session types, you can query the status immediately by issuing SESSCNTL CONNECTION(STATUS).

CLOSE

Closes connection to the host for the specified session. Initiation of the disconnection is indicated by message AXC0557I. Disconnection behavior is dependent on the session type. Results of the disconnection for TN3270 or TN5250 sessions iare reported by message AXC1804W. For other session types, you can query the status immediately by issuing SESSCNTL CONNECTION(STATUS).

STATUS

Returns the specified session's connection state. (The returned value is stored in the AXCRESULT variable.)

YES: Indicates that session is connected

NO: Indicates that session is not connected

Default: There is no default.

BWUPDATE

Specifies the window update frequency of background windows. The value (*numsecs*), which is an integer between 1 and 9, specifies the number of seconds between background window updates.

A lower value increases the apparent speed of the background windows and uses more machine cycles.

For unattended workstations, set BWUPDATE to 9 for greater throughput.

For an operator workstation, you may want to add menu items to allow the operator to easily control the BWUPDATE file. The DEBUG.mnu file contains sample statements to add these items to the Cmdarea menu.

Default: 5

TELNETHOST

Specifies the name of the host for the Telnet connection in the specified session. For example:

```
ADDRESS AXC "SESSCNTL TELNETHOST(mf.ca.com) SESSION(TELNETA)"
```

SESSION

Used with the AUTOMATE operand, defines a session for which you want to pause or restart automation. Used with the TELNETHOST operand, defines a session that is to monitor the Telnet connection. This operand is not valid with the BWUPDATE operand.

Usage Notes:

- The operands AUTOMATE, CONNECTION, BWUPDATE, and TELNETHOST are mutually exclusive.
- We strongly recommend that you pause automated sessions prior to closing a connection. We also recommend that you logout from all server-side applications.

  Establishing session connection does not restart automation automatically, you have to use separate SESSCNTL AUTOMATE(RESTART) command. It is recommended to let the session settle after each connect or disconnect operation, before you issue new SESSCNTL CONNECTION command for that particular session.
- DHD (Defer Host Disconnect) technology on the host can also affect the behavior of sessions that are reconnected shortly after disconnect.

Example:

The following REXX statements determine the current automation state of session ASYNCH_1 and pause automation for that session if it is active:

```
'SESSCNTL AUTOMATE(STATUS) SESSION(ASYNCH_1)'
IF AXCRESULT = "YES" THEN
  'SESSCNTL AUTOMATE(PAUSE) SESSION(ASYNCH_1)'
```

SETVAR Command

The SETVAR command sets the contents of a CA Automation Point status variable to a given value, changes the value of a variable, or creates a new variable.

This command has the following format:

"SETVAR *varname* (*value*)"

varname

Specifies the name of the CA Automation Point variable to set.

value

Specifies the value to be assigned to the variable.

Usage Notes:

- SETVAR status variable names are case-sensitive in CA Automation Point. Use uppercase variable names consistently.
- When designing REXX programs that use the SETVAR command, split long data lines for storage in multiple status variables. The maximum length of a SETVAR command is 241 characters, *after* substituting values for & variables. If you exceed the 241-character limit, CA Automation Point truncates the extra characters and reports a return code.

Example:

Suppose that REXX is operating without errors and you want to create a CA Automation Point variable called REXX_STATUS with a value indicating the status of REXX processing. To do so, write this statement in your REXX program:

"SETVAR REXX_STATUS (NO ERRORS)"

When the SETVAR command executes, the value of the REXX_STATUS variable is set to NO ERRORS.

STOPREXX Command

The STOPREXX command stops an executing REXX program or cancels execution of a queued REXX program. The STOPREXX command can stop a REXX program only if the program is running within CA Automation Point.

Note: You must issue the GETREXXL command *before* issuing the STOPREXX command.

This command has the following format:

"STOPREXX *rexxprognum*"

rexxprognum

Specifies the internal REXX pointer for a specific REXX program (acquired by a previously issued GETREXXL command).

Usage Note:

It is possible for the STOPREXX command to stop the issuing REXX program, meaning that your REXX program could terminate itself. Be sure that your REXX code allows for such a scenario when checking the GETREXXL_NAME.*n* variables generated by the GETREXXL command.

Example:

Suppose that you want to stop all instances of a particular program. Your REXX statements could look like these:

```
"GETREXXL"

/* GETREXXL populates GETREXXL.0 with the number   */
/* of active and queued REXX programs, the         */
/* GETREXXL_NAME.i variables with the name and     */
/* arguments of a program, and GETREXXL.i with the */
/* number used by STOPREXX to terminate the program*/

DO I = 1 TO GETREXXL.0
  parse upper value GETREXXL_NAME.i with rxname.
  IF rxname  = token THEN
  "STOPREXX "GETREXXL.I
END
```

WAIT Command

The WAIT command causes the program to wait for a specified number of seconds (you can specify any decimal value from 0.01 to 600) before executing the next REXX statement. It works only from within a REXX program.

This command has the following format:

"WAIT *seconds*"

seconds

Specifies the number of seconds that an executing REXX program pauses before executing the next REXX statement. Specify any integer value from 1 to 600.

Example:

Use the WAIT command to give a mainframe session time to react to a SESSCMD command before checking the screen for a result. For example, suppose that:

- Your REXX program includes a SESSCMD statement that sends the text string @C to a session to clear the screen.
- You want the REXX program to stop executing for five seconds to give the SESSCMD command time to clear the session screen.

To cause the REXX program to wait for the SESSCMD command to execute, include this statement in the program:

"WAIT 5"

When the statement executes, the REXX program pauses for five seconds before executing the next instruction in the program.

Commands for REXX-related Operations

The following sections describe the ADDRESS AXC commands used for REXX-related operations.

REXX Command

The REXX command queues a REXX program for processing by CA Automation Point. You can also use the REXX command *within* a REXX program to start another, independent REXX program.

This command has the following format:

"REXX '*execname args*"'

execname

Specifies the name of the REXX program for CA Automation Point to execute.

args

Specifies the calling arguments for the specified REXX program.

Note: The length of the argument list (REXX program name and its arguments) is limited to 505 characters. An incoming argument list longer than 505 characters is truncated.

Example:

The following command invokes a REXX program called XCDEMO with the argument GETMSG:

"REXX 'XCDEMO GETMSG"'

SCRIPT Command

The SCRIPT command invokes a CA Automation Point script from a REXX program. The script executes immediately.

This command has the following format:

"SCRIPT *scriptname* SESSION(*sessname*)"

scriptname

Specifies the file name of the script to execute.

SESSION

Specifies the name (*sessname*) of the target session in which CA Automation Point should activate the script.

Example:

The SCRIPT command shown below invokes a script called PAUSE.SCR, which restores the MCS console for session S028 for manual use:

"SCRIPT PAUSE.SCR SESSION(S028)"

SESSCMD Command

The SESSCMD command sends a text string (such as a command) to a specified session, and then retrieves a screen image from that session after the text string has been processed. (To fetch the screen image, an executing SESSCMD command invokes the GETSCRN command automatically.)

This command has the following format:

"SESSCMD '*keystring*' SESSION(*sessname*)
[CMDWAIT(*nn*)]
[PREFIX(*prefix*|LINE|NO)]
[SCREEN(YES|NO)]"

keystring

Specifies the text of the command to be issued to the session. The string can contain up to 234 characters. If you execute the SESSCMD command from within a REXX program and the *keystring* contains quotation marks, specify the command by using different delimiter characters of your choice to enclose the text.

As part of the command text, you can include one of the following:

- A key abbreviation
- An operation instruction

Besides text, the string can also contain key abbreviations and keyboard operation instructions. A *key abbreviation* consists of the @ character followed by a letter (uppercase or lowercase), a digit, or the @ character itself. When specifying a key abbreviation, enter it exactly as shown in the key abbreviation table found in the appendix on customizing special CA Automation Point files in the *Administrator Guide*.

Note: The SESSCMD command automatically appends an ENTER keystroke (@E) to your *keystring* value.

SESSION

Specifies the name of the target session (*sessname*) to receive the command text.

CMDWAIT

(Optional) Specifies the number of seconds (up to 60) that CA Automation Point waits for the keyboard to unlock after the SESSCMD executes. After the specified waiting period, CA Automation Point then captures the current screen image of the current session.

If the time period you specified with the CMDWAIT operand expires before CA Automation Point issues the command, the SESSCMD command produces a return code of 32, meaning that the keyboard is locked. If CA Automation Point issues the command but the session does not respond to that command, the SESSCMD command generates a return code of 36.

Default: 15

PREFIX

(Optional) Creates a stem variable to store the lines of a screen image returned to CA Automation Point. CA Automation Point stores each line of the screen in a separate variable and numbers the variables consecutively (for example, LINE.1, LINE.2, and so on).

Note: The variable numbered 0-for example, LINE.0-always contains the total number of elements in the stem variable.

Valid PREFIX values are:

prefix

Specifies he name that you want to assign to the line variables. For example, specifying PREFIX(ROW) causes CA Automation Point to create a set of variables named ROW.0, ROW.1, ROW.2, and so on through ROW.*n*.

LINE

Tells CA Automation Point to create stem variables named LINE1 through LINE*n* to store lines of a screen image.

NO

Creates no variables. Specifying PREFIX(NO) saves memory if you do not intend to use the variables.

Default: LINE

Note: For compatibility with earlier versions of CA Automation Point, the SESSCMD command also creates the old-style, non-stem variables. For example, suppose that the PREFIX variable is LINE. In addition to the stem variables described above, SESSCMD also creates the nonstem variables LINE1, LINE2, and so on through LINE*n*. In this case, the variable LINE (equivalent to the stem variable LINE.0) contains the total number of elements in the variable.

Each time you issue the SESSCMD command, CA Automation Point creates SCREEN and LINE.*n* variables automatically unless you specify otherwise.

SCREEN

(Optional) Specifies whether CA Automation Point creates the variable SCREEN to store the character portion of the screen image. Valid SCREEN values are:

YES

Creates the SCREEN variable and other variables. For more information on those other variables, see the description of the GETSCRN command (see page 75).

NO

Creates no variables. This option saves memory if you do not intend to use the variables.

Default: YES

Usage Notes:

A keyboard operation instruction consists of the @ character followed by a keyboard operation name enclosed in quotation marks. For example, the instruction for the MODE SELECT key is @"MODE_SEL". The *Administrator Guide* lists valid operation names

- CA Automation Point appends an ENTER keystroke (@E) before it sends the string.
- You can use key abbreviations and operations only where appropriate. For example, you cannot send the abbreviation for the PA1 key (@x) to a VT100 session because PA1 operates only in 3270 sessions.
- When you want to send a text string to a session without appending an ENTER keystroke to it (such as sending strings to 3270 sessions that contain 3270 AID keys: Enter, Clear, PA1 through PA3, PF1 through PF24), you can invoke the CA Automation Point @"SEND" operation from the same SESSCMD command.
- When the text string contains single or double quotes, use a different delimiter (such as /) to delimit the text string. For a complete description of alternate delimiters, see the section ADDRESS AXC Command Syntax (see page 70).
- The SESSCMD and GETSCRN commands allow CA Automation Point to interact with 3270, 5250, and asynchronous consoles and screens generated by applications. CA Automation Point rules can also control some of those screens.
- For master console sessions, do not issue instructions that either change the state of the console or clear messages from the screen while automation is active.
- If you want to invoke the SESSCMD command from within a REXX program, use the following methods to prevent the SESSCMD command from interfering with message processing by rules:
  - Use a rule to capture the response of a SESSCMD command.

    When you issue a host command to an automated console session using the SESSCMD command in a REXX program, use CA Automation Point rules to capture data from the z/OS command response. In such a scenario, the rule stores the captured data in status variables; the REXX program retrieves the values from the status variables using the GETVAR command.

    Note: Rules process only the first line of a multi-line command response. To retrieve data from a multi-line response, invoke a REXX program or CLIST on the host to issue the host command, capture the command response data, and return the data to CA Automation Point as a set of single-line messages.
  - Use REXX programs instead of rules to control the console.

    TSO sessions are good candidates for REXX programs to control because you can use those sessions to communicate with your mainframe automation product.

    Note: If you use REXX programs to control a console, do not automate the session when you define it in the session definition set.

- If you issue an instruction to a session in one of the following ways, CA Automation Point sends the instruction as one complete unit:
  - Using a SESSCMD command
  - Using the KEY keyword for scripts
  - From the command line of the Command window, the Merged Messages window, or the AP Messages window
  - From rules clauses containing any of these keywords: OSCMD, REPLY, or SESSCMD

  However, a series of commands from any of those sources executes uninterrupted *only* if no other CA Automation Point facilities are using the session.
- **Special considerations for VT52, VT100 and VT320 sessions:** The CTRL+H key maps to an operation named DEL (represented by the ASCII character (ý) which has a decimal value of 127), a *nondestructive* backspace. The syntax for sending the nondestructive backspace from a REXX program is @"ý". (In most text editors, you can obtain the (ý) character by typing ALT+127. Press and hold the ALT key, type 127 on the numeric keypad, then release the ALT key.)

  The BACKSPACE key maps to the BACK_SPACE operation, a *destructive* backspace. When specifying a destructive backspace from a REXX program, you need to suppress the ENTER key. The syntax is:

  'SESSCMD /@"BACK_SPACE"@"SEND"/ SESSION(*sessname*)'

Examples:

- Suppose that you want a REXX program to issue the ISPF command =X (to take a TSO session out of ISPF) to a TSO session named S028. Insert the following statement into your REXX program:

  "SESSCMD /=X/ SESSION(S028)"
- Suppose that you want to send a NEWLINE keystroke to a non-automated, full-screen session named SESS_A, and you do not want the SESSCMD command to automatically append an ENTER keystroke. Insert the following statement into your REXX program:

  'SESSCMD /@N@"SEND"/ SESSION(SESS_A)'
- Suppose that you want to send a "power on reset" command to an asynchronous session called ASYNCH. Insert the following statement into your REXX program:

  'SESSCMD /@"POWER_RESET"/ SESSION(ASYNCH)'

Commands for Utilities

The following sections describe CA Automation Point utility commands.

DOM Command

The DOM command deletes an action message from the action message area of the Merged Messages window and from the Action Message Recall window (by deleting it from the CA Automation Point action message recall list).

Note: Invoke the DOM command only *after* you have invoked the GETMSGI command or the GETVAR command. The GETMSGI command returns the *domid* value. The GETVAR command retrieves a previously saved DOMID value.

This command has the following format:

"DOM DOMID(*domid*)"

DOMID

Specifies the internal pointer to a specific action message in the action message recall list.

Usage Note:

When you invoke the GETMSGI command, it stores the internal pointer value to an action message in the DOMID.*n* REXX variable. Before invoking the DOM command, assign the DOMID.*n* value to another variable name-a name that does not contain a period-and use the new variable name for *domid.*

Examples:

- Suppose that you want to delete all of the action messages containing the word completed. You could write a REXX program containing statements like the following:

```
"GETMSGI"
DO CNT=1 TO DOMID.0 BY 1
  INTERPRET 'CUR_ACTION_MSG=DOMID_MSG.'CNT
  IF POS('completed',CUR_ACTION_MSG) > 0 THEN DO
    INTERPRET 'CUR_DOMID=DOMID.'CNT
    "DOM DOMID(&CUR_DOMID)"
    END
END
```

- Suppose that you want to delete a specific action message. The first step is to save the internal pointer (stored in the &DOMID environmental variable) to a working variable. Assume that this rule exists in your rules file:

  ```
  MSGID(SAVEDOM), HILIGHT, SET(&SAVE_ID=&DOMID)
  ```

  Your REXX program could contain these statements:

  ```
  "GETVAR SAVE_ID CUR_DOMID"
  "DOM DOMID(&CUR_DOMID)"
  "SETVAR SAVE_ID (0)"
  ```

  The first statement invokes the GETVAR command, which fetches the internal pointer to the action message produced by the rule and stores it in the CUR_DOMID variable. The second statement invokes the DOM command, which deletes the action message that the CUR_DOMID value points to in The CA Automation Point action message recall list. The last statement invokes the SETVAR command, which resets the SAVE_ID working variable used by the rule to 0.

PLOT Command

The PLOT command displays a window containing system information shown as a graph. For a detailed description of the PLOT command, see the *Administrator* Guide.

SESSCONFIG Command

The SESSCONFIG command populates a REXX stem variable with configured session attributes for sessions selected according to specified criteria from a given session definition set. This command has the following format:

"SESSCONFIG [SESSTYPE({*sesstype*|ENABLED|DISABLED|ALL})]
[DEFSET(*sessiondefinitionset*|STARTED|SAVED)] [PREFIX(*sessconfig*)]"

SESSTYPE

(Optional) Specifies the type of session to be listed. This operand can have one of the following arguments:

sesstype

Lists attributes of all sessions in the selected session definition set with the console type of *sesstype*. Possible *sesstype* names are:

- ASYNCH
- DTX
- ISERIES
- JES3
- JES3MCS
- MCS
- RCS
- SYSPLEX
- TANDEM
- TANDEMALL
- VAX
- VAXALL
- VM
- VSE
- Default

ENABLED

Lists attributes of all sessions configured as enabled.

DISABLED

Lists attributes of all sessions configured as disabled.

ALL

Lists attributes of all sessions in the selected session definition set.

Default: ALL

DEFSET

(Optional) Specifies the which definition set from which to get information

sessiondefinitionset

Specifies the name of the session definition set from which to get information

STARTED

Uses the session definition set that is currently loaded in AP Desktop, which is unaffected by any configuration or runtime changes made after AP Desktop starts

SAVED

Uses the session definition set that is currently selected as active in Configuration Manager

Default: STARTED

PREFIX

(Optional) Specifies the REXX stem variable that stores the session attributes. This value can be a maximum of 15 charaters.

Default: SESSCONFIG

Output of the command follows:

***prefix*.DEFINITION_SET**

Contains the session definition set name

***prefix*.0**

Contains the total number of returned sessions

prefix.i

Contains session name, where *i* is 1 to *prefix*.0

prefix.i.attribute

Reports following attributes for each session, where *i* is 1 to *prefix*.0 and *attribute* is one of the following keywords:

AUTOMATED

Indicates the automation status:

- YES: Session is configured as automated.
- NO: Session is configured as non-automated.

CONNECTED

Indicates the connection status:

- YES: Session is configured to connect on startup.
- NO: Session is configured not to connect on startup.

CONSOLE_TYPE

Indicates the console type (Default, ASYNCH, MCS, TANDEMALL, DTX, RCS, VAX, JES3, SYSPLEX, VAXALL, JES3MCS, TANDEM, VM, VSE, ISERIES)

DEVICE_NAME

Indicates the LU name or Device name (applicable only for TN3270, TN5250)

DEVICE_TYPE

Indicates the communication device or protocol used to connect the session

- COM*n*: Serial communication port *n* (COM1, COM2, ...)
- MEMORY: Session that uses memory as communication device
- ETC: Session is controlled by the Event Traffic Controller
- VIO: Windows command prompt session
- TELNET, SSH, TN3270, TN5250 : Session is connected to a remote host using this protocol

ENABLED

Indicates if a session is to be loaded:

- YES: Session will be loaded on CA Automation Point startup
- NO: Session will not be loaded

HOST_NAME

Indicates the host name (applicable only for TELNET, SSH, TN3270, or TN5250)

HOST_PORT

Indicates the port (applicable only for TELNET, SSH, TN3270, or TN5250)

MENU

Indicates the custom menu name or Default

SESSION_NAME

Indicates the name of the session

SYSTEM_NAME

Indicates the system name

TERMINAL

Indicates the terminal type (3278, 3278_2, 3278_3, 3278_4, 3278_5, 3279, 3279_2, 3279_3, 3279_4, 3279_5, 3477, 5292, 6530, ASYNCH, AXC, VIO, VIO43, VIO50, VT52, VT100, VT320, VT420)

TITLE

Indicates the title

If an attribute is not applicable for particular session, the attribute value contains the string "N/A".

SESSLIST Command

The SESSLIST command populates a REXX stem variable with current session attributes for currently loaded sessions selected according to specified criteria. Disabled sessions as specified in the active session definition set are not returned. This command has the following format:

"SESSLIST [SESSTYPE({*sesstype*|AUTOMATED|PAUSED|CONNECTED|DISCONNECTED|ALL})] [PREFIX(*sessname*)]"

SESSTYPE

(Optional) Specifies the type of session to be listed. This operand can have one of the following arguments:

sesstype

Lists attributes of sessions in the active session definition set with the console type of *sesstype*. Possible *sesstype* names are:

- ASYNCH
- DTX
- ISERIES
- JES3
- JES3MCS
- MCS
- RCS
- SYSPLEX
- TANDEM
- TANDEMALL
- VAX
- VAXALL
- VM
- VSE
- Default

AUTOMATED

Lists attributes of all sessions currently being automated.

PAUSED

Lists attributes of all sessions for which automation is paused.

CONNECTED

Lists attributes of all sessions currently connected to a host.

DISCONNECTED

Lists attributes of all sessions currently not connected to a host.

ALL

Lists attributes of all enabled sessions in the active session definition set.

Default: CONNECTED

PREFIX

(Optional) Specifies the REXX stem variable in which to store the session attributes.This value can be a maximum of 15 characters.

Default: SESSNAME

Output of the command follows:

***prefix*.DEFINITION_SET**

Contains the session definition set name which is currently active in AP desktop

***prefix*.GLOBAL_PAUSE**

Indicates whether global automation pause is currently in effect

***prefix*.0**

Contains the total number of returned sessions

prefix.i

Contains session names (where *i* is 1 to *prefix*.0).

prefix.i.attribute

Reports following attributes for each session, where *i* is 1 to *prefix*.0 and *attribute* is one of the following keywords:

AUTOMATED

Indicates the automation status:

- YES: session is currently being automated
- PAUSED: automation is temporarily paused
- NO: session is non-automated

CONNECTED

Indicates the connection status:

- YES: session is currently connected
- NO: session is currently not connected

CONSOLE_TYPE

Indicates the console type (Default, ASYNCH, MCS, TANDEMALL, DTX, RCS, VAX, JES3, SYSPLEX, VAXALL, JES3MCS, TANDEM, VM, VSE, ISERIES)

DEVICE_NAME

Indicates the LU name or Device name (applicable only for TN3270, TN5250)

DEVICE_TYPE

Indicates the communication device or protocol used to connect the session

- COM*n*: Serial communication port *n* (COM1, COM2, ...)
- MEMORY: Session that uses memory as communication device
- ETC: Session is controlled by the Event Traffic Controller
- VIO: Windows command prompt session
- TELNET, SSH, TN3270, TN5250: Session is connected to a remote host using this protocol

HOST_NAME

Indicates the host name (applicable only for TELNET, SSH, TN3270, or TN5250)

HOST_PORT

Indicates the port (applicable only for TELNET, SSH, TN3270, or TN5250)

MENU

Indicates a custom menu name or Default

SESSION_NAME

Indicates the name of the session

SYSTEM_NAME

Indicates the system name

TERMINAL

Indicates the terminal type (3278, 3278_2, 3278_3, 3278_4, 3278_5, 3279, 3279_2, 3279_3, 3279_4, 3279_5, 3477, 5292, 6530, ASYNCH, AXC, VIO, VIO43, VIO50, VT52, VT100, VT320, VT420)

TITLE

Indicates the title

If an attribute is not applicable for particular session, the attribute value contains the string "N/A".

WTO Command

The WTO command displays a text string in the normal message area of the AP Merged Messages window and in the Normal Message Recall window.

This command has the following format:

"WTO '*text*"'

text

Specifies the text (up to 251 characters) to be displayed. The text must be enclosed in single quotation marks ('); if the text contains embedded single quotes, use one of the alternate delimiters described in ADDRESS AXC Command Syntax in this chapter.

Example:

By using the WTO command to send messages to the Normal Message Recall window or the Merged Messages window, you can notify operators of important events as they occur. For example, you may include this statement in a REXX program:

"WTO 'REXX finished processing"'

When the statement executes, this message appears in the normal (non-highlighted) message area of the Merged Messages window:

REXX finished processing

WTOH Command

The WTOH command displays a text string in the action message area of the Merged Messages window and in the Action Message Recall window.

By using the WTOH command to send messages to the Action Message Recall window or the Merged Messages window, you can alert operators to problems with REXX processing.

This command has the following format:

"WTOH '*text*"'

text

Specifies the text (up to 128 characters) to be displayed. The text must be enclosed in single quotation marks ('); if the text contains embedded single quotes, use one of the alternate delimiters described in the section ADDRESS AXC Command Syntax in this chapter.

Example:

Suppose that the session name VM10 identifies a VM session on your system and you want to notify the operator when a severe error occurs in that session. To do so, place this statement in a REXX program:

"WTOH 'REXX detects severe error in session VM10"'

WTXC Command

The WTXC command displays a specified text string in the AP Messages window and the AP Message Recall window.

Note: CA Automation Point rules can process messages that the WTXC command issues from within a REXX program.

This command has the following format:

"WTXC '*text*"'

text

Specifies the text (up to 128 characters) to be displayed. The text must be enclosed in single quotation marks ('); if the text contains embedded single quotes, use one of the alternate delimiters described in the section ADDRESS AXC Command Syntax in this chapter.

Usage Note:

By using the WTXC command to send messages to the AP Message Recall window and the AP Messages window, you can keep track of how CA Automation Point interacts with REXX.

Example:

Suppose that you want the AP Messages Window to alert the operator when REXX takes some action (such as notifying a system programmer) when an error occurs. To do so, include this statement in a REXX program:

```
"WTXC 'REXX has notified system programmer of errors"'
```

Chapter 5: ADDRESS PPQ Commands

Program-to-program (PPQ) commands are commands that you can issue through the ADDRESS PPQ statement in a REXX program. For details about the CA Automation Point PPQ environment, see the chapter on using program-to-program queues in the *Administrator Guide.*

ADDRESS PPQ Command Summary

The following sections summarize CA Automation Point ADDRESS PPQ commands.

PPQ Setup Command

Use the following command for PPQ setup.

CREATE

Creates a new queue

PPQ Operations Commands

Use the following commands for PPQ operations.

LOCK

Prevents access to a queue by REXX programs other than the current one.

READ

Reads (accesses) one or more items from a specified queue.

UNLOCK

Restores access to a queue locked by a previously issued LOCK command.

WRITE

Writes (inserts) one or more items to a specified queue.

PPQ Dismantling Commands

Use the following commands to get data for dismantling PPQs.

DELETE

Deletes all elements in a specified queue and releases the memory allocated for the queue.

DISCONNECT

Breaks the connection with a specified remote queue and closes all sessions with the remote queue.

Special PPQ Commands

The following commands are used for special purposes.

COUNT

Counts the number of items in a specified queue.

DEBUG

A diagnostic tool that controls debugging trace output. This command is useful for providing diagnostic information to Technical Support.

LIST

Lists information about one or more queues residing on the local or remote machine, or both.

TRANSTATUS

A diagnostic tool that returns a table of transport-specific information. This command is useful for providing diagnostic information to Technical Support.

VER

Provides the version number and configuration of CA Automation Point PPQs.

ADDRESS PPQ Command Syntax

Issue a PPQ command from within a REXX program by specifying an ADDRESS PPQ statement, as shown:

ADDRESS PPQ '*ppqcommand operand(s)*'

Follow these guidelines when issuing a PPQ command:

- When specifying a required or optional operand with a PPQ command, use parentheses to pass values. For example**:**

  COUNT(*numitems*)

- Uppercase or lowercase characters are valid. For example:

  QUEUE(QUE_1)
  queue(que_1)

- Leading and trailing blanks are ignored. For example:

  ITEM(*item*)
  ITEM (*item*)

- Single and double quotes are supported. For example:

  CMDRESP(*destination*)
  CMDRESP('*destination*')
  CMDRESP("*destination*")

- When creating REXX programs that issue PPQ commands, do not name your variables with names reserved for PPQ commands. Also, do not include the following characters: ( ) ' " < > |, as they are interpreted by the REXX interpreter as REXX delimiters.

- Operands shown in brackets ([ ]) are optional.

ADDRESS PPQ Return Information

This section discusses the data returned by PPQ commands.

The RC Variable

RC is the REXX variable that contains the return codes from the ADDRESS PPQ environment. RC is set by every command and should be programmatically checked for acceptable results (usually a zero value) after each command executes.

The PPQ.ERROR Variable

If a PPQ command does not execute successfully-returning a non-zero RC value-it generates an error message and stores the message in the special REXX variable called PPQ.ERROR.

The PPQ error message ID begins with the prefix “PPQ” followed by a four-digit number (corresponding to the RC return code value) and a letter indicating the message type. See the *Message Reference Guide* for a more detailed description of the error message.

For example, if RC=4005, then the error message contained in the PPQ.ERROR variable is:

```
PPQ4005E  Insufficient memory.
```

Additional Return Information

The following PPQ commands, if executed successfully, return additional information beyond a return code (RC) value:

- COUNT
- LIST
- READ
- TRANSTATUS
- VER

Return information is stored in the special variable PPQ.*ppqcommand*. (The *ppqcommand* portion of the stem variable represents the name of the PPQ command.)

The PPQ.*ppqcommand*.0 variable contains the number of lines of information returned (that is, the number of elements in the PPQ.*ppqcommand* variable). The variables PPQ.*ppqcommand*.1 through PPQ.*ppqcommand*.*n* each contain a line of information. (The *n* value represents the last line of return information.)

Note: The PPQ.*ppqcommand* variable contains the same value stored in the PPQ.*ppqcommand*.1 variable (that is, the first, and sometimes only, return information line).

Change the Default Variable with PREFIX

You can direct return information to a variable other than the default PPQ.*ppqcommand* by specifying the PREFIX operand as follows:

PREFIX(*newvarname*)

newvarname

Specifies the name of the variable to replace the default.

Note: The PREFIX operand is valid only if the destination of the return information is REXX. For information about specifying other destinations, see the following section.

Change the Default Return Destination with CMDRESP

The CMDRESP operand shown directs the return information from a PPQ command to a specific destination:

CMDRESP(*destination*)

The following are valid values for destination:

REXX

Directs return information to a REXX variable. For more information, see the chapter "Using Program-to-Program Queues" in the *Administrator Guide*.

Note: The optional PREFIX operand is valid only if the destination of the return information is REXX.

XDQ

Directs return information to the external data queue.

TERMINAL

Directs return information to the terminal. This form uses the same output mechanism used by the REXX SAY command.

NOWHERE

Directs return information to the “bit bucket.” This value discards the return information.

Default: REXX

PPQ Setup Command

The following section describes the ADDRESS PPQ command used for setup.

CREATE Command

The create command creates a new queue.

This command has the following format:

```
ADDRESS PPQ 'CREATE QUEUE(qname) [DROP(NO|YES)]
            [SHARE(NO|YES)]
            [MAXITEMS(maxitems)]
            [PERSISTENCE(PERMANENT|TEMPORARY)]
            [TRANSPORT(network-transport-name)]'
```

QUEUE

Specifies the 1- to 16-character name of the queue (*qname*). The queue name can contain alphanumeric characters and any of these special characters: ! @ # $ _

The queue name cannot contain blanks.

Note: The qname cannot take the value of the computer name of any of the systems in your PPQ network.

DROP

(Optional) Causes the first (oldest) item in the queue to be dropped from the queue when an attempt is made to write to a full queue.

Default: NO

SHARE

(Optional) Specifies whether the queue is network-shared on the allowed transport.

For more information about network-shared versus local queues, see the chapter on using program-to-program queues in the *Administrator Guide*

Default: NO

MAXITEMS

(Optional) Specifies the maximum number of items (1 to 10,000) the queue can contain.

Default: 100

PERSISTENCE

(Optional) Specifies whether a queue should be deleted by the REXX program that created it when that program terminates. Valid PERSISTENCE values are:

PERMANENT

The queue remains (persists), even if the creating REXX program terminates (unless the PPQs services terminate or you restart your workstation).

TEMPORARY

Specifies that the queue is to be deleted when the creating REXX program terminates.

Default: PERMANENT

TRANSPORT

(Optional) Specifies the transport a queue is to be created on. This operand is valid only when SHARE(YES) is specified; it is otherwise ignored.

Specifies the *network-transport-name* is the network transport specified on the TRANSPORT() startup initialization parameter.

Valid *network-transport-name* is:

PPQTCP

Specifies the network transport is TCP/IP.

Note: This keyword will be removed in a future release.

Usage Note:

You should not create multiple queues with the same name in different transports within one PPQs network. (A *transport* is the mechanism that moves data from one location to another, such as shared memory on your workstation or TCP/IP.

Example:

```
'CREATE QUEUE(PROD1) SHARE(YES)'
```

PPQ Operations Commands

The following sections describe the ADDRESS PPQ commands used for PPQ operations.

LOCK Command

The LOCK command prevents access to a queue by REXX programs other than the current one. It provides a means for handling problems that can arise from the multiple access situations created by the multitasking environment. Using LOCK is necessary only when you are writing multiple items that must be sequential in the queue.

This command has the following format:

ADDRESS PPQ 'LOCK QUEUE(*qname*) [WAIT(NO|YES|*waittime*)]'

QUEUE

Specifies the 1- to 16-character name of the queue (*qname*). The queue name can contain alphanumeric characters and any of these special characters: ! @ # $ _

The queue name cannot contain blanks.

WAIT

(Optional) Specifies what the command does if it cannot execute immediately. For example, the queue may be in use or locked by another REXX program. Valid WAIT values are:

NO

Returns a nonzero return code (RC) value immediately. For information about what happens when a command does not execute successfully, see the chapter on using program-to-program queues in the *Administrator Guide*.

YES

Waits indefinitely while blocking the current REXX program until the queue becomes available.

waittime

Specifies the number of seconds to wait (up to 100000) while blocking the current REXX program. Fractional values (such as 1.5) are valid.

Default: NO

Usage Notes:

- Unlock a queue with the UNLOCK command as soon as possible after you lock it to prevent unnecessary delays in other REXX programs.
- Locks are nested. For every LOCK command issued for a queue, an equal number of UNLOCK commands are required to completely remove the lock. Be sure that each LOCK command in your REXX program has a corresponding UNLOCK command.

  Note: You can retrieve the lock nesting count using the LIST command.
- When a queue is in use by another REXX program, the LOCK command does not take effect until the current operation has completed. If the delay is prolonged or if the queue is locked by another REXX program, the LOCK command waits for the amount of time specified by the optional WAIT operand.

- You can lock a remote queue.
- It is good programming practice for user-written programs to eventually unlock every queue that they have locked.

READ Command

The READ command reads one or more items from a specified queue.

This command has the following format:

```
ADDRESS PPQ 'READ QUEUE(qname) [COUNT(numitems)]
            [CMDRESP(destination)]
            [ITEMNUM(FIRST|LAST|itemnum)]
            [PREFIX(newvarname)]
            [REMOVE(YES|NO)]
            [WAIT(NO|YES|waittime)]'
```

QUEUE

Specifies the 1- to 16-character name of the queue (*qname*). The queue name can contain alphanumeric characters and any of these special characters: ! @ # $ _

The queue name cannot contain blanks.

CMDRESP

(Optional) Directs return information, if any, to a specific destination. For a list of valid *destination* values, see the section Change the Default Return Destination with CMDRESP (see page 117).

Default: REXX

COUNT

(Optional) Specifies the number of items (*numitems*) to read from the queue.

Default: 1

ITEMNUM

(Optional) Specifies the starting item to read from the queue. The starting item that you specify must exist; the READ command cannot wait for an item that does not yet exist. Valid ITEMNUM values are:

FIRST

Starts the read operation from the first item in the queue.

LAST

Starts the read operation from the last item in the queue.

itemnum

Starts the read operation from an item number that you specify. (A value of 1 is the same as a value of FIRST.)

Default: FIRST

PREFIX

(Optional) Directs return information to a stem variable name (*newvarname*) other than the default. For more information, see ADDRESS PPQ Return Information (see page 115).

Note: The PREFIX operand is only valid if REXX is the destination of the return information.

Default: PPQ.READ

REMOVE

(Optional) Specifies whether the read operation is destructive or nondestructive (that is, whether the READ command removes an item from the queue after it has read the item). Valid values are:

YES

Removes the item from the queue after reading it.

NO

Does not remove the item from the queue after reading it.

Default: YES

WAIT

(Optional) Specifies what the command does if it cannot execute immediately. For example, the queue may be in use or locked by another REXX program. Valid WAIT values are:

NO

Returns a nonzero return code (RC) value immediately. For information about what happens when a command does not execute successfully, see the chapter on using program-to-program queues in the *Administrator Guide*.

YES

Waits indefinitely while blocking the current REXX program until the queue becomes available and is not empty.

waittime

Specifies the number of seconds to wait (up to 100000) for the queue to become available while blocking the current REXX program. Fractional values (such as 1.5) are valid.

Default: NO

Usage Notes:

- The ITEMNUM operand and the WAIT operand are mutually exclusive.
- When reading from a remote queue, especially using the TCP/IP transport, it is possible to receive return code 4520 (queue not found) indicating errors when the queue exists on the remote machine. These errors can occur if remote host network response times are slow or inconsistent. To alleviate this condition, use the following sample to code a function around the read request to detect the problem and recover from it:

```
READRMTPPQ:
/*
*********************************************************************
*    THIS FUNCTION WILL READ A PPQ REMOTE QUEUE WITH TIMEOUT.     *
*********************************************************************
* EXAMPLE CALL: RC = READPPQ('RMTQ', 4*60, 'REMOVE(NO) COUNT(2)')  *
* RETURNS: RC FROM PPQREAD                                          *
* NOTES: FOR LOCAL QUEUE, USE PPQREAD WAIT() PARAMETER IN PLACE OF *
*       THIS ROUTINE.                                               *
*********************************************************************
**
*/
QUEUE     = ARG(1)  /* QUEUE NAME         */
TIMEOUT   = ARG(2)  /* TIME OUT IN SECONDS */
READ_PARMS = ARG(3)  /* OTHER PPQREAD PARMS */

/* DEFAULT 3 MINUTE TIME OUT */
IF TIMEOUT = '' THEN TIMEOUT = 3*60

/* ENSURE WAIT PARM */
IF POS('WAIT(',READ_PARMS) = 0
  THEN READ_PARMS = 'WAIT(1)' READ_PARMS

/* START TIMER */
X = TIME('R')

/* LOOP UNTIL QUEUE FOUND, TIME OUT, OR ERROR */
DO UNTIL TIME('E') > TIMEOUT
  ADDRESS 'PPQ' 'READ QUEUE('QUEUE')' READ_PARMS
  IF RC <> 4520 THEN LEAVE
END
RETURN RC
```

UNLOCK Command

The UNLOCK command restores access to a queue locked by a previously issued LOCK command.

This command has the following format:

```
ADDRESS PPQ 'UNLOCK QUEUE(qname)'
```

QUEUE

Specifies the 1- to 16-character name of the queue (*qname*). The queue name can contain alphanumeric characters and any of these special characters: ! @ # $ _

The queue name cannot contain blanks.

Usage Notes:

- Unlock a queue as soon as possible after you lock it to prevent unnecessary delays in other REXX programs.
- Locks are nested. For every LOCK command issued for a queue, an equal number of UNLOCK commands is required to completely remove the lock.

  You can retrieve the lock nesting count using the LIST command.
- It is good programming practice for user-written programs to eventually unlock every queue that they have locked.

WRITE Command

The WRITE command writes one or more items to a specified queue.

This command has the following format:

```
ADDRESS PPQ 'WRITE QUEUE(qname) ITEM(item)|VAR(varname)
            [DATATYPE(BINARY|TEXT)]
            [ITEMNUM(LAST|FIRST|itemnum)]
            [WAIT(NO|YES|waittime)]'
```

QUEUE

Specifies the 1- to 16-character name of the queue (*qname*). The queue name can contain alphanumeric characters and any of these special characters: ! @ # $ _

The queue name cannot contain blanks.

ITEM

Specifies the item to write to the queue. The *item* value can be either a literal string (such as "this is an item") or a simple variable name (not enclosed in quotation marks so that REXX can evaluate it).

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

VAR

Specifies the name of a simple or compound variable (*varname*) containing an item or items to write to the queue.

A trailing dot after the *varname* value (as in *varname*.) specifies a compound variable, where *varname*.0 contains the number of items to write, and *varname*.1, *varname*.2, and so on, each contain an item to write. If you specify a compound variable containing more than one item, the WRITE command locks the specified queue temporarily, writes all items to the queue contiguously, and then unlocks the queue.

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

DATATYPE

(Optional) Specifies which type of data translation should be used. Valid DATATYPE values are:

TEXT

Automatically translates the data between EBCDIC and ASCII to match the requirements of the receiving system. This value is most useful when you want to send text strings between z/OS and Windows.

BINARY

Specifies that no data translation should occur.

Note: The BINARY format is faster and should be used for all data, including text, if all systems that communicate through PPQs use the same character set.

Default: BINARY

ITEMNUM

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

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

LAST

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

FIRST

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

itemnum

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

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

Default: LAST

WAIT

(Optional) Specifies what the command does if it cannot execute immediately. For example, the queue may be in use or locked by another REXX program. Valid WAIT values are:

NO

Returns a nonzero return code (RC) value immediately. For information about what happens when a command does not execute successfully, see the chapter on using program-to-program queues in the *Administrator Guide*.

YES

Waits indefinitely while blocking the current REXX program until the queue becomes available.

waittime

Specifies the number of seconds to wait (up to 100000) while blocking the current REXX program. Fractional values (such as 1.5) are valid.

Default: NO

Usage Note:

When writing to a remote queue, especially using the TCP/IP transport, it is possible to receive return code 4520 (queue not found), indicating errors when the queue exists on the remote machine. These errors can occur if remote host network response times are slow or inconsistent. To alleviate this condition, use the following sample to code a function around the read request to detect the problem and recover from it:

```
WRITERMTPPQ:
/*
*********************************************************************
*    THIS FUNCTION WILL WRITE A PPQ REMOTE QUEUE WITH TIMEOUT.    *
*********************************************************************
* EXAMPLE CALL: RC = WRITEPPQ('RMTQ', 4*60, 'DATATYPE(BINARY)')    *
* RETURNS: RC FROM PPQWRITE                                        *
* NOTES: FOR LOCAL QUEUE, USE PPQWRITE WAIT() PARAMETER IN PLACE OF*
*       THIS ROUTINE.                                              *
*********************************************************************
*/
QUEUE      = ARG(1)  /* QUEUE NAME          */
TIMEOUT    = ARG(2)  /* TIME OUT IN SECONDS  */
WRITE_PARMS = ARG(3)  /* OTHER PPQWRITE PARMS */

/* DEFAULT 3 MINUTE TIME OUT */
IF TIMEOUT = '' THEN TIMEOUT = 3*60

/* ENSURE WAIT PARM */
IF POS('WAIT(',WRITE_PARMS) = 0
  THEN WRITE_PARMS = 'WAIT(1)' WRITE_PARMS

/* ENSURE DATATYPE PARM */
IF POS('DATATYPE(',WRITE_PARMS) = 0
  THEN WRITE_PARMS = 'DATATYPE(TEXT)' WRITE_PARMS

 /* START TIMER */
X = TIME('R')

/* LOOP UNTIL QUEUE FOUND, TIME OUT, OR ERROR */
DO UNTIL TIME('E') > TIMEOUT
  ADDRESS 'PPQ' 'WRITE QUEUE('QUEUE')' WRITE_PARMS
  IF RC <> 4520 THEN LEAVE
END
RETURN RC
```

PPQ Dismantling Commands

The following sections describe the ADDRESS PPQ commands used for dismantling queues.

DELETE Command

The DELETE command deletes all elements in the specified queue and releases the memory allocated for the queue.

This command has the following format:

ADDRESS PPQ 'DELETE QUEUE(*qname*) [WAIT({NO|YES|*waittime*})]'

QUEUE

Specifies the 1- to 16-character name of the queue (*qname*). The queue name can contain alphanumeric characters and any of these special characters: ! @ # $ _

The queue name cannot contain blanks.

WAIT

(Optional) Specifies what the command does if it cannot execute immediately. For example, the queue may be in use or locked by another REXX program. Valid WAIT values are:

NO

Returns a nonzero return code (RC) value immediately. (For information about what happens when a command does not execute successfully, see the chapter on using program-to-program queues in the *Administrator Guide*.

YES

Waits indefinitely while blocking the current REXX program until the queue becomes available.

waittime

Specifies the number of seconds to wait (up to 100000) while blocking the current REXX program. Fractional values (such as 1.5) are valid.

Default: NO

Usage Notes:

The following restrictions apply to the DELETE command:

- If the queue you want to delete is a network-shared queue--that is, if it was created with the CREATE command's SHARE(YES) option--the DELETE command first makes the queue local (unshared) so that it can delete the queue.
- You can delete only local queues. The only way to delete a remote queue is through a REXX program on the machine where the queue resides.
- You cannot delete a locked queue unless the calling REXX program is also the lock holder.
- A temporary queue--created with the CREATE command PERSISTENCE(TEMPORARY) option--can be deleted only by the REXX program that created it.

DISCONNECT Command

The DISCONNECT command breaks the connection with the specified remote queue. (This command is nondestructive; it does not delete the remote queue.)

When the local workstation has no more use for a remote queue, issue the DISCONNECT command to free transport resources. (A *transport* is the mechanism that moves data from one location to another, such as shared memory on your workstation or TCP/IP.)

If a REXX program disconnects from a remote queue that other REXX programs on the same workstation are still using, an automatic reconnection occurs transparently. An error occurs if the queue is currently locked by a program other than the calling program.

This command has the following format:

```
ADDRESS PPQ 'DISCONNECT QUEUE(qname)'
```

QUEUE

Specifies the 1- to 16-character name of the queue (*qname*). The queue name can contain alphanumeric characters and any of these special characters: ! @ # $ _

The queue name cannot contain blanks.

Special PPQ Commands

The following sections describe the ADDRESS PPQ commands used for special purposes.

COUNT Command

The COUNT command counts the number of items in a specified queue. The return information is a positive integer.

This command has the following format:

```
ADDRESS PPQ 'COUNT QUEUE(qname) [CMDRESP(destination)]
            [PREFIX(newvarname)]'
```

QUEUE

Specifies the 1- to 16-character name of the queue (*qname*). The queue name can contain alphanumeric characters and any of these special characters: ! @ # $ _

The queue name cannot contain blanks.

CMDRESP

(Optional) Directs return information, if any, to a specific destination. For a list of valid *destination* values, see the section Change the Default Return Destination with CMDRESP (see page 117).

Default: REXX

PREFIX

(Optional) Directs return information to a stem variable name (*newvarname*) other than the default. For more information, see ADDRESS PPQ Return Information (see page 115).

Note: The PREFIX operand is only valid if REXX is the destination of the return information.

Note: The PREFIX operand is only valid if REXX is the destination of the return information.

Default: PPQ.COUNT

DEBUG Command

The DEBUG command controls debugging trace output. You can prevent messages below a specified severity level from being traced, and you can specify what types of messages that you want to trace. Like the TRANSTATUS command, the DEBUG command is a diagnostic tool.

This command has the following format:

ADDRESS PPQ 'DEBUG [CATEGORY(*hexbitfield*)]
[CMDRESP (*destination*)]
[SEVERITY(*severitylevel*)] '

CATEGORY

(Optional) Specifies a mask for the category of message to be traced. Specify the *hexbitfield* value as a 32-bit hexadecimal string. Each bit position containing a 1 allows the trace utility (ASOTRACE.exe) to trace the corresponding message category.

Default: FFFFFFFF

This parameter specifies the bit assignments are as follows:

General Hex Code	**Trace Type**
FFFFFFFF	All
00000001	Startup
00000002	Initialization and termination
00000004	Manager thread
00000008	API entry point
00000010	Status
Hex Code	**Trace Type**
00000020	Control block
00000040	Packet
00000080	Memory allocation
00000100	High volume routine
Application Hex Code	**Trace Type**
00100000	Utility routine
00200000	PPQ
00800000	VOX

01000000	OCF
00200000	GLV

CMDRESP

(Optional) Directs return information, if any, to a specific destination. For a list of valid *destination* values, see the section Change the Default Return Destination with CMDRESP (see page 117).

Default: REXX

SEVERITY

(Optional) Prevents tracing of messages with severity levels below the level specified for *severitylevel*. Valid severity levels are:

1

Error

2

Warning

3

Informational

4-8

Debug levels

Note: CA Technical Support specifies the appropriate debug trace level when necessary.

Default: 1

LIST Command

The LIST command lists information about one or more queues residing on the local or remote workstation, or both. The optional QUEUE and SCOPE operands allow you to specify search criteria to target specific queues. This command is useful for troubleshooting TCP/IP problems.

Each line of information that the LIST command returns represents one queue. Each field in a line contains a specific type of information about the queue, as follows:

Field	Description
1	The name of the queue
2	The number of items in the queue
3	The maximum number of items allowed in the queue

Field	Description
4	The maximum allowed length of each item
5	The lock nesting counter
6	The name of the transport where the queue was found
7	The scope of the queue (local, remote, or shared)
8	The name of the transport the queue uses
9	The system name of the creator of the queue

This command has the following format:

```
ADDRESS PPQ 'LIST [CMDRESP(destination)]
            [PREFIX(newvarname)]
            [QUEUE(qname)]
            [SCOPE(ALL|LOCAL|REMOTE)]
            [TRANSPORT(network-transport-name)]'
```

CMDRESP

(Optional) Directs return information, if any, to a specific destination. For a list of valid *destination* values, see the section Change the Default Return Destination with CMDRESP (see page 117).

Default: REXX

PREFIX

(Optional) Directs return information to a stem variable name (*newvarname*) other than the default. For more information, see ADDRESS PPQ Return Information (see page 115).

Note: The PREFIX operand is only valid if REXX is the destination of the return information.

Default: PPQ.LIST

QUEUE

(Optional) Specifies the 1- to 16-character name of the queue (*qname*). The queue name can contain alphanumeric characters and any of these special characters: ! @ # $ _

The queue name cannot contain blanks.

Note: Wildcard characters are not valid in this case.

SCOPE

(Optional) Specifies queues on the local or remote workstation, as follows:

ALL

Lists queues residing on the local workstation *and* queues on remote workstations that REXX programs on the local workstation have accessed.

LOCAL

Lists only those queues residing on the local workstation.

REMOTE

Lists only those queues residing on remote workstations that REXX programs on the local workstation have accessed.

Default: ALL

TRANSPORT

(Optional) Specifies a valid transport.

The *network-transport-name* is the network transport specified on the TRANSPORT() startup initialization parameter.

Valid network transport names are:

PPQTCP

The network transport is TCP/IP.

When TRANSPORT(PPQTCP) is specified, the LIST command displays the current state of each TCP/IP connection to the remote hosts listed under TCP/IP Host Names in the Configuration Manager Program-to-Program Queues dialog.

'LIST TRANSPORT(PPQTCP)' displays one of the following forms of output:

If the local machine is configured only with IPv4 under TCP/IP:

Session *nnnnnnn*: IP: *iphost1* State: *host1state* Host '*host1*'
Session *nnnnnnn*: IP: *iphost2* State: *host2state* Host '*host2*'

I**f the local machine is configured to support both IPv4 and IPv6 under TCP/IP:**

Session nnnnnnn: State: host1state
IP: *iphost1*
Host '*host1*'
Session *nnnnnnn*: State: *host2state*
IP: *iphost2*
Host '*host2*'

nnnnnnn -- Specifies the internal session identifier

iphostn- -- Specifies the IPv4 or IPv6 address for the host named *hostn*; an IPv6 address will be enclosed in square brackets

hostnstate -- Specifies the state of the session that is connected to the host named *hostn*. Values are UP, DOWN, STARTING, LISTEN, BROKEN, and UNKNOWN.

Notes:

- The old IPv4 format is deprecated; it may be removed in a future release in favor of the new IPv4/IPv6 format.
- PPQTCP is the only valid value for TRANSPORT.

TRANSTATUS Command

The TRANSTATUS command returns a table of transport-specific information. (A *transport* is the mechanism that moves data from one location to another, such as shared memory on your workstation or TCP/IP.) Like the DEBUG command, the TRANSTATUS command is a diagnostic tool. This command is useful for providing diagnostic information to Technical Support.

Each line of information that the TRANSTATUS command returns represents one transport. Each field in a line contains a specific type of information about the transport, as follows:

Field	Description
1	The name of the transport (such as SharedMemory or TCP/IP)
2	The version number (for example, 11.3.0.0 Rev=99999)
3	The state of the transport (such as UNKNOWN, INITIALIZATION, OPERATIONAL, or TERMINATION)

The following is an example of return information lines:

```
SHAREDMEMORY 11.3.0.0 Rev=99999 OPERATIONAL
TCP/IP       11.3.0.0 Rev=99999 OPERATIONAL
```

This command has the following format:

```
ADDRESS PPQ 'TRANSTATUS [CMDRESP(destination)]
            [PREFIX(newvarname)]'
```

CMDRESP

(Optional) Directs return information, if any, to a specific destination. For a list of valid *destination* values, see the section Change the Default Return Destination with CMDRESP (see page 117).

Default: REXX

PREFIX

(Optional) Directs return information to a stem variable name (*newvarname*) other than the default. For more information, see ADDRESS PPQ Return Information (see page 115).

Note: The PREFIX operand is only valid if REXX is the destination of the return information.

Default: PPQ.TRANSTATUS

VER Command

The VER command provides the version number and configuration of the CA Automation Point PPQs on your workstation.

The VER command returns one line of information containing four fields. The fields contain the following information:

Field	Description
1	The PPQ service name (for example, PPQ)
2	The version number (for example, 11.3.0.0 Rev=99999)
3	The Build date of the current PPQ service in *mmm dd yyyy* format (for example, Jul 5 2010)
4	The system name for the PPQ service on this workstation as defined during configuration (for example, PPQSYS)

The following is an example of return information line:

```
PPQ 11.3.0.0 Rev=99999 Jul 5 2010 PPQSYSTEM
```

This command has the following format:

```
ADDRESS PPQ 'VER [CMDRESP(destination)] [PREFIX(newvarname)]'
```

CMDRESP

(Optional) Directs return information, if any, to a specific destination. For a list of valid *destination* values, see the section Change the Default Return Destination with CMDRESP (see page 117).

Default: REXX

PREFIX

(Optional) Directs return information to a stem variable name (*newvarname*) other than the default. For more information, see ADDRESS PPQ Return Information (see page 115).

Note: The PREFIX operand is only valid if REXX is the destination of the return information.

Default: PPQ.VER

Chapter 6: ADDRESS GLV Commands

You issue GLV commands through the ADDRESS GLV statement in a REXX program. For details about The CA Automation Point global variable environment, see the chapter on configuring and writing REXX in the *Administrator Guide*.

ADDRESS GLV Command Summary

The following list summarizes CA Automation Point ADDRESS GLV commands.

GET

Retrieves the current value of a status or GLV variable and assigns that value to a local REXX variable of the same name.

GRPLIST

Places a list of all variable groups into the external data queue.

GRPLISTV

Places a list of all variable groups directly into a REXX variable.

LIST

Places the name and value of a variable into the external data queue.

LISTV

Places the name and value of a variable directly into the REXX variable.

PURGE

Deletes all volatile variables in a specified variable group.

PUT

Assigns the value of a local REXX variable to a volatile status or GLV variable of the same name.

PUTP

Assigns the value of a local REXX variable to a nonvolatile status or GLV variable of the same name. (This command creates or updates a .GLV file.)

SELECT

Selects a particular variable group for succeeding GLV commands to act upon. This command is valid globally as a stand-alone command or locally as an operand in most other GLV commands.

SET

Assigns values to one or more volatile variables.

SETL

Assigns a literal value, which can contain blanks, to a single volatile variable.

SETLP

Assigns a literal value, which can contain blanks, to a single nonvolatile variable. (This command creates or updates a .GLV file.)

SETP

Assigns values to one or more nonvolatile variables. (This command creates or updates a .GLV file.)

VER

Provides the version number and maintenance level of GLV services on your workstation to the external data queue.

VERV

Provides the version number and maintenance level of GLV services on your workstation into a REXX variable.

ADDRESS GLV Command Syntax

Issue a GLV command from within a REXX program by specifying an ADDRESS GLV statement, as shown:

ADDRESS GLV '*glvcommand operand(s)*'

Operands shown in brackets ([ ]) are optional.

Return Codes for GLV Commands

After a GLV command executes, it sets the special REXX return code variable RC. The RC variable contains a value of 0 (zero) if the command executed successfully; otherwise, the RC variable contains one of these values:

RC	Description
-86	Unrecognized command name
-84	Unrecognized command operand
-82	Command operand is missing

If the RC variable contains any other value, contact CA Technical Support at http://ca.com/support.

ADDRESS GLV Command Descriptions

The following sections describe the ADDRESS GLV commands.

GET Command

The GET command retrieves the value of any variable (that is, any status or GLV variable whether volatile or nonvolatile) and assigns that value to a local REXX variable of the same name.

This command has the following format:

ADDRESS GLV '[SELECT *vargroup*] GET *varname* . . . '

SELECT

(Optional) Specifies the name of a variable group (*vargroup*) to which the specified variable (*varname*) belongs.

For more information, see the description of the SELECT (see page 148) command.

GET

Specifies the name of the variable whose value you want to retrieve. (CA Automation Point places the contents of the specified variable into a local REXX variable of the same name.)

You can specify one or more *varname* variable names.

Examples:

- The following REXX statement assigns the values of two nonvolatile status variables (AXCDISK_DVOL and AXCDISK_ERRORS) from the variable group AXC to local REXX variables of the same names:

  ADDRESS GLV 'SELECT AXC GET AXCDISK_DVOL AXCDISK_ERRORS'

- The following REXX statement assigns the value of VOXCALLS (a GLV variable that could be volatile or nonvolatile in this example) to a local REXX variable of the same name. (Notice that the VOXCALLS variable is a member of the VOXGROUP variable group.)

  ADDRESS GLV 'SELECT VOXGROUP GET VOXCALLS'

GRPLIST Command

The GRPLIST command places a list of all variable group names into the external data queue, making that information available to any REXX program.

This command has the following format:

```
ADDRESS GLV 'GRPLIST'
```

Example:

The following REXX statements list the variable groups in use:

```
SAY 'These are the variable groups currently in use:'
ADDRESS GLV 'GRPLIST'
do while queued() > 0
  parse pull line
  say line
end
```

GRPLISTV Command

The GRPLISTV command places a list of all variable group names directly into a REXX variable.

This command has the following format:

```
ADDRESS GLV 'GRPLISTV rexxstem'
```

GRPLISTV

Specifies the name of the REXX stem variable (*rexxstem*) that will contain the return information for the command.

Example:

The following example lists the variable groups in use:

```
ADDRESS GLV 'GRPLISTV rexxstem'
say 'These are the variable groups currently in use:'
do i=1 to rexxstem.0
  say rexxstem.i
end
```

LIST Command

The LIST command places the name and value of any variable (that is, any status or GLV variable whether volatile or nonvolatile) into the external data queue, making that value available to any REXX program.

This command has the following format:

```
ADDRESS GLV '[SELECT vargroup] LIST [varname . . .] '
```

SELECT

(Optional) Specifies the name of a variable group (*vargroup*) to which the specified variable (*varname*) belongs.

For more information, see the description of the SELECT (see page 148) command.

varname

(Optional) Specifies the name of the variable whose name and value you want to place into the external data queue. You can specify one or more varname variable names.

The following wildcard characters are valid for variable name masking:

? or +

Replaces individual characters.

*****

Replaces any number of trailing characters when placed as the last character in the name mask.

If you do not specify a varname value, the LIST command writes all variables (in the selected or default variable group) to the external data queue.

Example:

The following REXX statement places the name and value of the variable LASTUSER into the external data queue:

```
ADDRESS GLV 'SELECT USER_GRP LIST LASTUSER'
```

LISTV Command

The LISTV command places the name and value of any variable, that is, any status or GLV variable whether volatile or nonvolatile, into a REXX variable.

This command has the following format:

ADDRESS GLV '[SELECT *vargroup*] LISTV *rexxstem* [*varname* . . .] '

SELECT

(Optional) Specifies the name of a variable group (*vargroup*) to which the specified variable (*varname*) belongs.

For more information, see the description of the SELECT (see page 148) command.

rexxstem

Specifies the name of the REXX stem variable that will contain the return information for the command.

varname

(Optional) Specifies the name of the variable whose name and value you want to place into the selected stem variable. You can specify one or more varname variable names.

The following wildcard characters are valid for variable name masking:

? or +

Replaces individual characters.

*

Replaces any number of trailing characters when placed as the last character in the name mask.

If you do not specify a *varname* value, the LIST command writes all variables (in the selected or default variable group) to the selected stem variable.

Example:

The following REXX example places the name and value of the variable(s) starting with D into the REXX variable:

```
ADDRESS GLV 'SELECT UNNAMED LISTV REXXSTEM D*'
say 'These are all the variables in the UNNAMED group that start with "D":'
do i=1 to rexxstem.0
  say rexxstem.i
end
```

PURGE Command

The PURGE command deletes all volatile variables within a specified variable group. (The PURGE command has no effect on nonvolatile variables.)

This command has the following format:

```
ADDRESS GLV '[SELECT vargroup] PURGE'
```

SELECT

(Optional) Specifies the name of a variable group (*vargroup*) to which the specified variable (*varname*) belongs.

For more information, see the description of the SELECT (see page 148) command.

If you do not select a variable group, the PURGE command erases all volatile variables in the group selected by the previously issued SELECT command. If you have not yet selected a group, the PURGE command erases all volatile variables in the UNNAMED group.

Examples:

- The following REXX statement erases the volatile variables in the variable group RELAYGRP:

  ```
  ADDRESS GLV 'SELECT RELAYGRP PURGE'
  ```

- The following REXX statement erases all volatile variables in the default variable group (either UNNAMED or another group specified by the previously issued SELECT command):

  ```
  ADDRESS GLV 'PURGE'
  ```

PUT Command

The PUT command assigns the value of a local REXX variable to a *volatile* status or GLV variable of the same name.

This command has the following format:

ADDRESS GLV '[SELECT *vargroup*] PUT *varname* . . .'

SELECT

(Optional) Specifies the name of a variable group (*vargroup*) to which the specified variable (*varname*) belongs.

For more information, see the description of the SELECT (see page 148) command.

varname

Specifies the name of a volatile status or GLV variable. You can specify one or more varname variable names.

Note: If you assign a value to a volatile variable, and a nonvolatile variable of the same name already exists, the nonvolatile variable will be erased.

Example:

Assume that the following PUT statement follows a REXX program that has just finished processing, and that the program has assigned values to the local SUBTOTAL and TOTAL variables. The statement shown below assigns those values to volatile variables of the same name in the variable group STATS_GROUP:

```
ADDRESS GLV 'SELECT STATS_GROUP PUT SUBTOTAL TOTAL'
```

PUTP Command

The PUTP command assigns the value of a local REXX variable to a *nonvolatile* status or GLV variable of the same name. This command updates or creates a .glv file. (A .glv file contains nonvolatile variables.)

This command has the following format:

ADDRESS GLV '[SELECT *vargroup*] PUTP *varname* . . .'

SELECT

(Optional) Specifies the name of a variable group (*vargroup*) to which the specified variable (*varname*) belongs.

For more information, see the description of the SELECT (see page 148) command.

varname

Specifies the name of a nonvolatile status or GLV variable. You can specify one or more *varname* variable names.

Note: If you assign a value to a nonvolatile variable, and a volatile variable of the same name already exists, the volatile variable will be erased.

Example:

Assume that the following PUTP statement follows a REXX program that has just finished processing, and that the program has assigned values to the local SUBTOTAL and TOTAL variables. The statement shown below assigns those values to nonvolatile variables of the same name in the variable group STATS_GROUP:

```
ADDRESS GLV 'SELECT STATS_GROUP PUTP SUBTOTAL TOTAL'
```

SELECT Command

The SELECT command selects a default variable group for subsequent GLV commands to act upon.

This command has the following format:

```
ADDRESS GLV 'SELECT vargroup [glvcommand]'
```

vargroup

Specifies the name of a variable group that you want to select as the default group for subsequent GLV commands.

Default: UNNAMED

glvcommand

(Optional) One of the following GLV commands:

GET	PUTP	SETL
LIST	PURGE	SETLP
PUT	SET	SETP

Note: If you specify SELECT as an operand in one of the GLV commands listed here, the selected variable group applies to *that command only* and does not change the default variable group.

Usage Notes:

- The selected variable group becomes the default group for subsequent GLV commands until you select another variable group.
- If the variable group that you select does not exist, the SELECT command creates it.
- If you do not issue the SELECT command to select a default variable group, GLV commands assume the default group is UNNAMED.

Example:

In the following REXX code, assume that the default variable group is UNNAMED because you have not issued a SELECT command to select another default variable group.

The first LIST command requests the value of the variable LAST_USER (set by either CA Automation Point or another REXX program). The second LIST command (specifying SELECT locally as an operand) selects the variable group TESTER as the group to which the variable named NEW_USER belongs.

```
ADDRESS GLV 'LIST LAST_USER'
PULL LAST_USER.UNNAMED
ADDRESS GLV 'SELECT TESTER LIST NEW_USER'
PULL NEW_USER.TESTER
```

SET Command

The SET command assigns values to one or more volatile status or GLV variables.

This command has the following format:

```
ADDRESS GLV '[SELECT vargroup] SET varname value [varname value . . .]'
```

SELECT

(Optional) Specifies the name of a variable group (*vargroup*) to which the specified variable (*varname*) belongs.

For more information, see the description of the SELECT (see page 148) command.

varname

Specifies the name of a volatile status or GLV variable.

Note: If you assign a value to a volatile variable, and a nonvolatile variable of the same name already exists, the nonvolatile variable will be erased.

You can specify one or more *varname* variable names, but each *varname* that you specify must have a matching *value*.

value

Specifies the value to assign to the variable specified by *varname*. The valid value is any character string that does not contain blanks.

Example:

The following REXX statement assigns the literal value SYS1_IPL to the EVENT variable and the literal value DATE_AND_TIME to the EVENT_TIME variable:

```
ADDRESS GLV 'SET EVENT SYS1_IPL EVENT_TIME DATE_AND_TIME'
```

SETL Command

The SETL command assigns a literal value, which can contain blanks, to a single volatile status or GLV variable.

This command has the following format:

ADDRESS GLV '[SELECT *vargroup*] SETL *varname value*'

SELECT

(Optional) Specifies the name of a variable group (*vargroup*) to which the specified variable (*varname*) belongs.

For more information, see the description of the SELECT (see page 148) command.

varname

Specifies the name of a volatile status or GLV variable.

Note: If you assign a value to a volatile variable, and a nonvolatile variable of the same name already exists, the nonvolatile variable will be erased.

value

Specifies the value to assign to the variable specified by varname. Valid values are literal strings. The value can contain blanks.

Example:

The following REXX statement assigns a literal value to the volatile variable MESSAGE:

```
ADDRESS GLV 'SETL MESSAGE This is a sentence in one variable'
```

SETLP Command

The SETLP command assigns a literal value, which can contain blanks, to a single nonvolatile status or GLV variable. This command updates or creates a .glv file. (A .glv file contains nonvolatile variables.)

This command has the following format:

ADDRESS GLV '[SELECT *vargroup*] SETLP *varname value*'

SELECT

(Optional) Specifies the name of a variable group (*vargroup*) to which the specified variable (*varname*) belongs.

For more information, see the description of the SELECT (see page 148) command.

varname

Specifies the name of a nonvolatile status or GLV variable.

Note: If you assign a value to a nonvolatile variable, and a volatile variable of the same name already exists, the volatile variable will be erased.

value

Specifies the value to assign to the variable specified by *varname*. Valid values are literal strings. The value can contain blanks.

Example:

The following REXX statement assigns a literal value to the nonvolatile status variable AXCDISK_MESSAGE:

```
ADDRESS GLV 'SETLP AXCDISK_MESSAGE This is a sentence in one variable'
```

SETP Command

The SETP command assigns values to one or more *nonvolatile* status or GLV variables. This command updates or creates a .GLV file. (A .GLV file contains nonvolatile variables.)

This command has the following format:

ADDRESS GLV '[SELECT *vargroup*] SETP *varname value* [*varname value* . . .]'

SELECT

(Optional) Specifies the name of a variable group (*vargroup*) to which the specified variable (*varname*) belongs.

For more information, see the description of the SELECT (see page 148) command.

varname

Specifies the name of a nonvolatile status or GLV variable.

Note: If you assign a value to a nonvolatile variable, and a volatile variable of the same name already exists, the volatile variable will be erased.

You can specify one or more *varname* variable names, but each *varname* that you specify must have a matching *value*.

value

Specifies the value to assign to the variable specified by *varname*. The valid value is any character string that does not contain blanks.

Example:

The following REXX statement assigns the literal value SYS1_IPL to the EVENT variable, and the literal value DATE_AND_TIME to the EVENT_TIME variable:

```
ADDRESS GLV 'SETP EVENT SYS1_IPL EVENT_TIME DATE_AND_TIME'
```

VER Command

The VER command provides the version number, maintenance level, and build date of GLV services on your workstation.

This command has the following format:

```
ADDRESS GLV 'VER'
```

The VER command returns one line of information containing three fields to the external data queue. The fields contain the following information:

1

Contains the GLV service name (for example, GLV)

2

Contains the version number with service pack and patch (for example, 11.3.0.0 Rev=99999)

3

Contains the build date of the current GLV service in *mmm dd yyyy* format (for example, Jan 25 2010).

The following is an example of a return information line:

```
GLV 11.3.0.0 Rev=99999 Jan 25 2010
```

VERV Command

The VERV command provides the version number, maintenance level, and build date of GLV services on your workstation.

The VERV command returns one line of information containing three fields to a REXX variable.

This command has the following format:

ADDRESS GLV 'VERV *rexxstem*'

rexxstem

Specifies the name of the REXX stem variable that will contain the return information for the command.

Example:

The following is an example of the VERV command:

```
ADDRESS GLV "VERV LINE"
say 'The current GLV Manager version is: 'line.1'
```

Chapter 7: ADDRESS VOX Commands

CA Automation Point VOX commands are issued through the ADDRESS VOX statement in a REXX program. For details about the CA Automation Point VOX command environment, see the chapter on using notification services in the *Administrator Guide.*

ADDRESS VOX Command Summary

The following sections summarize CA Automation Point ADDRESS VOX commands.

Notification Manager Database Maintenance Commands

Use the following commands to manage the Notification Manager database.

ALTERENTITY

Alters the characteristics of an entity.

ALTERMETHOD

Alters the characteristics of a method.

ALTERPARM

Alters the characteristics of a parameter.

ALTERTIME

Alters the characteristics of a time block.

CREATEENTITY

Creates a new entity.

CREATELOGIN

Creates a new login.

CREATEMETHOD

Creates a new method.

CREATEPARM

Creates a new parameter.

CREATETIME

Creates a new time block.

DESTROYENTITY

Destroys an existing entity.

DESTROYLOGIN

Destroys an existing login.

DESTROYMETHOD

Destroys an existing method.

DESTROYPARM

Destroys an existing parameter.

DESTROYTIME

Destroys an existing time block.

EPWCHECK

Checks the password for an entity.

LISTENTITY

Lists all the data for one or more entities (except the password).

LISTFIND

Lists the time blocks that are active for one or more entities at a given date and time. The list is a tree that can contain any level of nesting.

LISTFORTO

Lists all the time blocks that forward to a given entity (no matter what the time and date). The list is a tree that can contain any level of nesting.

LISTLOGIN

Lists all the data for one or more logins.

LISTMETHOD

Lists all the data for one or more methods.

LISTPARM

Lists all of the data for one or more parameters.

LISTPERGRPS

Lists all personal groups for a specified contact.

LISTTIME

Lists all the data for one or more time blocks.

NMDBMERGE

Merges a previously exported copy of a database into the current Notification Manager database.

NMEXPORT

Creates a copy of the current Notification Manager database in a format that can be easily moved to another system.

NMIADDCALLER

Adds an entity to the list of entities that are allowed to call in on an item.

NMIANSWER

Sets or retrieves the answer for an item in the Notification Manager database.

NMICHECKCALLER

Checks to see whether an entity has been permitted to listen to/answer this item (by a call to NMIADDCALLER).

NMIGETITEM

Retrieves the ask text and tell text for an item from the Notification Manager database.

NMIMPORT

Imports a copy of a Notification Manager database.

Notification Commands

Use the following commands for notification tasks.

PAGE

Issues an alphanumeric page to designated personnel using a modem-based alphanumeric paging service.

PAGE2WAY

Issues an alphanumeric page to designated personnel using an Internet-based alphanumeric paging service.

SENDMAIL

Generates an e-mail message to designated personnel.

Voice Processing Commands

Use the following commands for voice processing tasks.

ANSWER

Waits for an incoming telephone call on any available voice channel, any voice channel within a group, or a specific voice channel only.

ANSWERPLAY

Waits for an incoming telephone call (in the same way as the ANSWER command) and plays one or more prerecorded voice messages after answering.

CALL

Initiates a telephone call using an open, available voice channel.

CALLPLAY

Initiates a telephone call on an available voice channel and plays one or more prerecorded voice messages.

CLEAR

Clears the digit buffer of a voice channel or its call progress analysis (CPA) parameters.

GETCHANNEL

Serializes I/O activity on a voice channel by marking an available channel as in-use. The calling REXX program then has exclusive access to the channel.

GETCHANNELNUM

Identifies the physical voice channel number associated with a channel handle.

GETDIGITS

Retrieves tone digits-such as menu selections or access codes that a remote party enters from a telephone keypad in response to a voice prompt-from the digit buffer of a voice channel.

GETGROUP

Returns a text string containing a group name and a list of all physical voice channel numbers associated with the group.

GETSTATUS

Returns the current status of a voice channel.

GETSYSNAMES

Retrieves the system names of all connected notification servers in a distributed environment.

LOAD

Loads a voice file or voice word library index file into main memory for faster access.

PLAY

Plays a prerecorded voice message through a specified voice channel.

PLAYGETDIGITS

Plays a prerecorded voice message through a specified voice channel, and then retrieves tone digits that the remote party enters from the telephone keypad.

RECORDFILE

Records a voice message from the remote party and stores it in a disk file.

RELEASECHANNEL

Resets a voice channel marked as in-use by the GETCHANNEL command, making the channel available to other REXX programs issuing a GETCHANNEL command. Used with the GETCHANNEL command to serialize I/O activity on a channel.

SENDTONES

Sends tone digits through an already-open voice channel, useful for sending additional tones after CA Automation Point has successfully called and connected to the remote party.

SETGROUP

Associates a group name with one or more voice channels.

SETHOOK

Sets the hook state of a voice channel to on-hook or off-hook, useful in special circumstances such as multiple CALL operations within a single REXX program.

SETVOLUME

Adjusts the volume for current and subsequent PLAY operations on a specified voice channel.

STOP

Terminates an active I/O operation on a voice channel.

VERIFYUSER

Checks the validity of the user ID and password combination of the remote party.

WINK

Sends a brief "handshaking" protocol signal through a voice channel.

Utility Commands

The following commands are ADDRESS VOX utility commands.

GETTAPIDEVICELIST

Lists all the TAPI devices that are installed under Windows.

SETENGINE

Allows you to modify notification server settings.

SETMSGSTREAM

Duplicates the message stream from CA Automation Point to another queue that is accessible through PPQs, local or remote.

SETTRACE

Start and stop trace logging.

SLEEP

Causes the issuing REXX EXEC to enter a system sleep state.

STARTREXX

Starts another REXX program.

VER

Provides the version number and configuration information of the CA Automation Point notification services at your site

ADDRESS VOX Command Syntax

To issue a VOX command, use an ADDRESS statement in your REXX program to access the CA Automation Point VOX command environment, as shown:

ADDRESS VOX "*voxcommand* [*operands...*]"

Follow these guidelines when issuing a VOX command:

- Use parentheses to pass required or optional operand values. For example:

  TIMEOUT(100)

- Uppercase or lowercase values are valid. For example:

  COUNT(5) or count(5)

- A VOX command ignores leading and trailing blanks. For example:

  LINE(15)

- A VOX command supports single and double quotes. For example:

  SYSTEM('VOX1') or SYSTEM("VOX1")

- Operands shown in brackets ([ ]) are optional.

ADDRESS VOX Return Information

This section discusses ADDRESS VOX Return variables.

The RC Variable

RC is the REXX variable that contains the return codes from the ADDRESS VOX environment. RC is set by every command and should be programmatically checked for acceptable results (usually a zero value) after each command executes.

The VOX.ERROR Variable

If a VOX command does not execute successfully (returning a nonzero RC value) it generates an error message and stores the message in the special REXX variable called VOX.ERROR.

A VOX error message ID begins with the prefix VOX followed by a four-digit number (corresponding to the RC return code value), and a letter indicating the message severity. See the *Message Reference Guide* for a more detailed description of the error message.

For example, if RC=5204, then the error message ID and message text contained in the VOX.ERROR variable is:

VOX5204E Invalid channel handle.

The VOX.voxcommand Variable

For the following commands, return information is stored in the special variable *VOX.voxcommand*:

- ANSWER
- CALL
- CALLPLAY
- GETCHANNEL
- GETCHANNELNUM
- GETDIGITS
- GETGROUP
- GETSTATUS
- GETSYSNAMES
- GETTAPIDEVICELIST
- LISTFIND
- LISTFORTO
- LISTLOGIN
- LISTMETHOD
- LISTPARM
- LISTTIME
- NMIGETITEM
- NMILISTANSWERS
- NMILISTCALLERS
- NMILISTITEMS
- PAGE2WAY
- PLAYGETDIGITS
- VER

(The *voxcommand* portion of the stem variable represents the name of the VOX command.)

The VOX.*voxcommand*.0 variable contains the number of lines of information returned (that is, the number of elements in the VOX.*voxcommand* variable). The variables VOX.*voxcommand*.1 through VOX.*voxcommand*.*n* each contain a line of information. (The *n* value represents the last line of return information.)

Note: The VOX.*voxcommand* variable contains the same value stored in the VOX.*voxcommand*.1 variable (that is, the first-and sometimes only-return information line).

Change the Default Variable with PREFIX

You can direct return information to a variable other than the default VOX.*voxcommand* by specifying the PREFIX operand.

To direct return information to another variable, enter the following command:

PREFIX(*newvarname*)

newvarname

Specifies the name of the variable to replace the default.

Note: The PREFIX operand is valid only if the destination of the return information is REXX. For information about specifying other destinations, see the following section.

Change the Default Return Destination with CMDRESP

To direct the return information from a VOX command to a specific destination, use the CMDRESP operand :

CMDRESP(*destination*)

The following are valid values for *destination*:

REXX

Directs return information to a REXX variable. For more information, see the preceding section.

Note: The optional PREFIX operand is valid only if the destination of the return information is REXX.

XDQ

Directs return information to the external data queue.

TERMINAL

Directs return information to the terminal. This form uses the same output mechanism used by the REXX command SAY.

NOWHERE

Directs return information to the “bit bucket.” This value discards the return information.

Default: REXX

Notification Manager Database Maintenance Commands

The following sections describe the CA Automation Point commands for managing the Notification Manager database.

ALTERENTITY Command

The ALTERENTITY command changes the characteristics of an entity.

This command has the following format:

```
ADDRESS VOX "ALTERENTITY KEY(key)
 [AVAIL(YES|NO)]
 [BROADCAST(YES|NO)]
 [ESCTO( key|name|0)]
 [NAME( name)]
 [PASSWORD( password)]
 [UNAVAILDESC(desc)]"
```

KEY

Specifies the key of the entity whose characteristics are to be altered.

AVAIL

(Optional) Specifies if the entity is available to be contacted. If the value is set to NO, no attempt will be made to contact this entity. Until the AVAIL is set to YES, Notification Manager will not attempt to contact this entity.

Note: This is not a valid operand for Groups.

Default: YES

BROADCAST

(Optional) Values are:

YES

Perform all times and methods that are active at the current time.

NO

Terminate processing for this entity as soon as one time or method succeeds.

ESCTO

(Optional) If this entity cannot be found, escalates to the entity whose *key* or *name* is specified. If *0* is specified, escalation does not take place.

NAME

(Optional) Specifies the new name to be assigned to this entity. Names can only contain uppercase and lowercase letters, numbers, blanks, and the following special characters: - _ @ # . $ % / ; : \

PASSWORD

(Optional) Changes the password for this entity to *password*. Passwords are between four and eight numeric digits.

UNAVAILDESC

(Optional) Defines the description of why an entity is unavailable.

Note: This operand is not valid for Groups.

Return Information:

After ALTERENTITY executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Example:

The following example changes the escalation entity for the entity with key 100001009 to Mary Jones:

```
ADDRESS VOX "AlterEntity KEY(100001009) ESCTO(Mary Jones)"
```

ALTERMETHOD Command

The ALTERMETHOD command changes the characteristics of a method.

This command has the following format:

```
ADDRESS VOX "ALTERMETHOD KEY(key)
 [INVOKE(invocation string)]
 [NAME(name)]
 [TYP(B|C|D|...|W)]"
```

KEY

Specifies the *key* of the method whose characteristics are to be altered.

INVOKE

(Optional) Specifies the string that is used to invoke this method.

The invocation string must begin with the name of the command or REXX script to be invoked. Parameters that never change can follow. (Parameters can also be added to the invocation by using the PARAMETERS feature of this database.)

NAME

(Optional) Specifies the new name to be assigned to the method. Names can only contain uppercase and lowercase letters, numbers, blanks, and the following special characters: - _ @ # . $ % / ; : \

TYP

(Optional) Specifies that the method type code for the method. This value is a one-letter designation. When NMFIND is invoked with the MTUP parameter and a TYP character is a member of the MTUP list, then the method is attempted.

Return Information:

After ALTERMETHOD executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Example:

The following example changes the invocation string for the method with key 200003001 to PAGE2:

```
ADDRESS VOX "AlterMethod KEY(200003001) INVOKE(PAGE2)"
```

ALTERPARM Command

The ALTERPARM command changes the characteristics of a parameter.

This command has the following format:

```
ADDRESS VOX "ALTERPARM KEY(parm key)
 [DESC(desc)]
 [VALUE(value)]"
```

KEY

Specifies the *key* of the parameter whose characteristics are to be altered.

DESC

(Optional) Describes the usage and meaning of this parameter. This data is not passed to the methods. This keyword may only be specified when the parameter is a method level parameter. The maximum length of this operand is 1000 characters.

VALUE

(Optional) Specifies the actual value of this parameter for the specified method and item. This data is passed to the methods. The maximum length of this operand is 240 characters.

Return Information:

After ALTERPARM executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Example:

The following example changes the value for parameter 300001039 to 68:

```
ADDRESS VOX "AlterParm KEY(300001039) VALUE(68)"
```

ALTERTIME Command

The ALTERTIME command changes the characteristics of a time block.

This command has the following format:

```
ADDRESS VOX "ALTERTIME KEY(parm key)
    [ASK(method key|name)]
    [BGNDATE(mm/dd)]
    [BGNTIME(hh:mm)]
    [DOW(SUN|MON|...|SAT)]
    [ENDDATE(mm/dd)]
    [ENDTIME(hh:mm)]
    [FORTO(entity name|key)]
    [NAME(name)]
    [PRIORITY(n)]
    [TELL(method key|name)]"
```

KEY

Specifies the *key* of the time block whose characteristics are to be altered.

ASK

(Optional) Specifies the *name* of the ask method to use during this time block.

If you specify the FORTO operand, you must specify a value of ForwardTo for this operand. If you specify a value other than ForwardTo, you cannot specify the FORTO operand.

BGNDATE

(Optional) Specifies the first date (no year is specified) on which this time block will be active. The period is active starting on this date every year. If you specify this operand, you can specify the ENDDATE operand, but not the DOW operand.

BGNTIME

(Optional) Specifies the time, in military time format (00:00 to 23:59), at which this time block becomes active on the specified dates or days of the week.

If you specify this operand, you must specify the ENDTIME operand.

DOW

(Optional) Specifies the days of the week on which this time block is active.

Days are specified as the first three characters of their English names. You can specify any number of days in any order. If you specify this operand, you cannot specify the BGNDATE and ENDDATE operands.

Separate the days of the week with spaces.

ENDDATE

(Optional) Specifies the last date (no year is specified) on which this time block becomes active. The period becomes inactive after this date every year. If you specify this operand, you can specify the BGNDATE operand, but not the DOW operand.

ENDTIME

(Optional) Specifies the time, in military time format (00:00 to 23:59), at which this time block becomes inactive on the specified dates or days of the week.

If you specify this operand, you must specify the BGNTIME operand.

Note: To make a time block active from 10:00 a.m. to 1:00 p.m., you must specify the ENDTIME as 13:00. If you specify BGNTIME(10:00) ENDTIME(12:59), the minute from 12:59 to 13:00 is not covered by the time block.

FORTO

(Optional) Specifies the *name* or *key* of the entity to forward to during this time block.

If you specify this operand, you cannot specify a value of ForwardTo for both the TELL and ASK operands.

NAME

(Optional) Change the name of this time block to *name*. Names can only contain uppercase and lowercase letters, numbers, blanks, and the following special characters: - _ @ # . $ % / ; : \

PRIORITY

(Optional) If multiple time blocks apply to a particular time of day, a certain algorithm is used to determine the order in which they will be attempted. This operand controls rule two of that ordering. The default ordering algorithm is as follows:

1. Time blocks with a BGNDATE and ENDDATE are performed before time blocks with a DOW.
2. Time periods with a higher priority are performed sooner.
3. If two time blocks with a BGNDATE and ENDDATE apply, the one whose BGNDATE is closest to the current date is performed first.
4. If two time blocks with a DOW apply, the one whose first active day of the week is closest to the current day of the week is performed first.
5. If there is still a tie, the time block whose start time is closest to the current time is performed first.
6. If there is still a tie, the time block whose end time is closest to the current time is performed first.
7. If there is still a tie, the order is random/undefined.

TELL

(Optional) Specifies the *name* of the tell method to use during this time block.

If you specify the FORTO operand, you must specify a value of ForwardTo for this operand. If you specify a value other than ForwardTo, you cannot specify the FORTO operand.

Return Information:

After ALTERTIME executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Example:

The following example changes the beginning date of time block 400001034 to 07/20:

```
ADDRESS VOX "AlterTime KEY(400001034) BGNDATE(07/20)"
```

CREATEENTITY Command

The CREATEENTITY command creates a new entity.

This command has the following format:

```
ADDRESS VOX "CREATEENTITY NAME(name)
  [AVAIL(YES|NO)]
  [BROADCAST(YES|NO)]
  [ESCTO(key|name)]
  [KEY(key)]
  [NMGROUP(YES|NO)]
  [PASSWORD(password)]
  [UNAVAILDESC(desc)]"
```

NAME

The name of the entity to be created. Names can only contain uppercase and lowercase letters, numbers, blanks, and the following special characters: - _ @ # . $ % / : ; \

AVAIL

(Optional) This operand defines if the entity is available to be contacted. Values are:

YES

Notification Manager will attempt to contact this entity

NO

No attempt will be made to contact this entity.

Until the AVAIL is set to YES, Notification Manager will not attempt to contact this entity

Note: This is not a valid operand for Groups.

Default: YES

BROADCAST

(Optional) Values are:

YES

Perform all methods that are active at the current time

NO

Terminate processing for this entity as soon as one method succeeds.

Default: NO

ESCTO

(Optional) If this entity cannot be found, escalates to the entity whose *key* or *name* is specified. If *0* is specified, escalation does not take place.

KEY

(Optional) Specifies the numeric ID to be associated with this entity. The ID must be between 100001001 and 199999999.

NMGROUP

(Optional) This operand determines whether the entity being created will be designated as an individual or a group. If this is set to YES, the entity created will be a group.

Default: NO

PASSWORD

(Optional) Specifies the password for this entity. Passwords are between four and eight numeric digits.

UNAVAILDESC

(Optional) This operand is used to define the description of why an entity is unavailable.

Note: This is not a valid operand for Groups.

Return Information:

After the CREATEENTITY command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, The RC variable contains a value of zero. The REXX variable vox.CreateEntity.NewKey contains the key of the newly created entity.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. The VOX.ERROR variable also contains the complete VOX error message, which includes the four digit return code and error message text.

Example:

The following example creates an entry in the database for Joe:

```
ADDRESS VOX "CreateEntity NAME(Joe) BROADCAST(no) PASSWORD(1111) ESCTO(Frank)"
```

CREATELOGIN Command

The CREATELOGIN command creates a new login.

This command has the following format:

```
ADDRESS VOX "CREATELOGIN USERNAME(name)
 [ENTITY(key|name)]"
```

USERNAME

Specifies the name of the login to be created. Usernames can contain uppercase and lowercase letters, numbers, blanks, periods, and the following special characters: - _ @ # $ % \ . :

ENTITY

(Optional) Specifies the entity to which the login is to be associated. If no value is given, the login is created without an associated entity. This can either be a key or a name. Each entity is only allowed one login.

Note: The key or name given cannot be a Notification group.

Default: 0

Return Information:

After the CREATELOGIN command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. The REXX variable vox.CreateLogin.NewKey contains the key of the newly created login.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. The VOX.ERROR variable also contains the complete VOX error message, which includes the four digit return code and error message text.

Usage Notes:

Privileges for a login cannot be set from the REXX interface. Each login created from REXX is assigned the same privileges as the Guest login account.

Example:

The following example creates an entry in the database for login SMIJO21:

```
ADDRESS VOX "CreateLogin USERNAME(SMIJ021) ENTITY(John Smith) "
```

CREATEMETHOD Command

The CREATEMETHOD command creates a new method.

This command has the following format:

```
ADDRESS VOX "CREATEMETHOD "NAME(name)
 INVOKE(invocation string)
 USE(TELL|ASK|BOTH)
 [TYP(B|C|D|...|W)]"
```

NAME

Specifies the name of the entity to be created.

Note: Names can only contain uppercase and lowercase letters, numbers, blanks, and the following special characters: - _ @ # . $ % / ; : \

INVOKE

Specifies the string that is used to invoke this method.

The invocation string must begin with the name of the command or REXX script to be invoked. Parameters that never change can follow. (Parameters can also be added to the invocation by using the PARMS features of this database.)

USE

Specifies what this method is used for. Values are:

TELL

The method may only be used to pass data to the entity.

ASK

The method may only be used to receive data from the entity.

Note: The ASK value on the USE operand is not applicable if you are using the CA Automation Point DBMS.

BOTH

The method may be used to ask, tell, or both.

TYP

(Optional) A one-letter designation that specifies the method type code for the method. When NMFIND is invoked with the MTUP parameter and if the TYP character is a member of the MTUP list, then the method is attempted.

Return Information:

After CREATEMETHOD executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. The REXX variable vox.CreateMethod.NewKey contains the key of the newly created method.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. The VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Example:

The following example creates an entry in the database for method MyMethod:

```
ADDRESS VOX "CreateMethod NAME(MyMethod) INVOKE(XREXX1) USE(both)"
```

CREATEPARM Command

The CREATEPARM command creates a new parameter.

This command has the following format:

```
ADDRESS VOX "CREATEPARM NAME(parm name) METHOD(method name or key)
  ITEM(time key|entity name or key) VALUE(value) [DESC(desc)]"
```

NAME

Specifies the name of the parameter to be created.

Names can only contain uppercase and lowercase letters, numbers, and the following special characters: - _ @ # . $ % / ; : \

METHOD

Specifies the *name* or *key* of the method that owns this parameter.

ITEM

Specifies the *name* or *key* of the entity that owns this parameter or the key of the time block that owns it.

You specify this parameter when you want to create a parameter at the entity or time block level. Do not specify this parameter if you want to create a parameter at the method level.

Note: To create a parameter at the entity or time-block level, you must first create a parameter with the same name at the method level.

VALUE

Specifies the actual value of this parameter for the specified method and item. This data is passed to the methods. The maximum length of this operand is 240 characters.

DESC

(Optional) Describes the use and meaning of this parameter.

When you create a parameter at the method level, you must specify a description of the parameter. When you create a parameter at the entity or time block level, you cannot specify a description of the parameter. This data is not passed to the methods. The maximum length of this operand is 1000 characters.

Usage Notes:

Parameter entries are divided into three increasingly specific classes:

- **The method class** - Those entries that apply any time the method is used.
- **The entity class** - Those entries that apply any time a method is used for a particular entity.
- **The time class** - Those entries that apply only for a particular time block, which implies a specific entity and method.

You cannot create (specify) a parameter value at the entity or time block level unless you first create a parameter with the same name at the method level. For instance, if the REXX program that performs the MyPage method has a PagerID parameter and you want to specify the pager ID for Chris, you must issue the following commands:

```
CreateParm Name(PagerID) Method(MyPage) Value() Desc(Pager ID)
CreateParm Name(PagerID) Method(MyPage) Item(Chris) Value(123)
```

Return Information:

After CREATEPARM executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. The REXX variable vox.CreateParm.NewKey contains the key of the newly created parameter.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Example:

The following example creates an entry in the database that defines 131 as the value for the parameter WAIT whenever the MyMethod method is used to tell Joe something:

```
ADDRESS VOX "CreateParm Method(MyMethod) ITEM(Joe) NAME(WAIT) VALUE(131)"
```

CREATETIME Command

The CREATETIME command creates a new time block.

This command has the following format:

```
ADDRESS VOX "CREATETIME ENTITY(key or name)
  DOW(SUN|MON|...|SAT)|BGNDATE(mm/dd)[ENDDATE(mm/dd)]
  BGNTIME(hh:mm)
  ENDTIME(hh:mm)
 {TELL(method key or name) ASK(method key or name)|FORTO(entity name or key)}
 [NAME(name)]
 [PRIORITY(n)]"
```

ENTITY

Specifies the *key* or *name* of the entity for this time block.

DOW

Specifies the days of the week on which this time block is active.

Days are specified as the first three characters of their English names. You can specify any number of days in any order. If you specify this operand, you cannot specify the BGNDATE and ENDDATE operands.

Separate the days of the week with spaces.

BGNDATE

Specifies the first date (no year is specified) on which this time block is active. The period is active starting on this date every year. If you specify this operand, you can specify the ENDDATE operand, but not the DOW operand.

ENDDATE

(Optional) Specifies the last date (no year is specified) on which this time block becomes active. The period becomes inactive after this date every year. If you specify this operand, you must specify the BGNDATE operand, but not the DOW operand. If not specified, the default value for ENDDATE will be the same value that is specified for BGNDATE.

BGNTIME

Specifies the time, in military time format (00:00 to 23:59), at which this time block becomes active on the specified dates or days of the week.

ENDTIME

Specifies the time, in military time format (00:00 to 23:59), at which this time block becomes inactive on the specified dates or days of the week.

Note: To have a time block active from 10:00 a.m. to 1:00 p.m., you must specify the ENDTIME as 13:00. If you specify BGNTIME(10:00) ENDTIME(12:59), the minute from 12:59 to 13:00 is not covered by the time block.

TELL

Specifies the *name* of the tell method to use during this time block.

If you specify this operand, you cannot specify the FORTO operand.

ASK

Specifies the *name* of the ask method to use during this time block. If you specify this operand, you cannot specify the FORTO operand.

FORTO

Specifies the *name* or *key* of the entity to forward to during this time block.

If you specify this operand, you cannot specify the TELL or ASK operand.

NAME

(Optional) Specifies the *name* to assign to this time block. Names can only contain uppercase and lowercase letters, numbers, blanks, and the following special characters: - _ @ # . $ % / ; : \

PRIORITY

(Optional) If multiple time blocks apply to a particular time of day, an algorithm is used to determine the order in which they are attempted.

This operand controls rule two of that ordering. The default ordering algorithm is as follows:

1. Time blocks with a BGNDATE and ENDDATE are always performed before time blocks with a DOW.
2. Time blocks with a higher priority are performed sooner.
3. If two time blocks with a BGNDATE and ENDDATE apply, then the one whose BGNDATE is closest to the current date is performed first.
4. If two time blocks with a DOW apply, the one whose first active day of the week is closest to the current day of the week is performed first.
5. If there is still a tie, the time block whose start time is closest to the current time is performed first.
6. If there is still a tie, the time block whose end time is closest to the current time is performed first.
7. If there is still a tie, the order is random/undefined.

Note: The priority operand overrides all of these rules except the first.

Return Information:

After CREATETIME executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. The REXX variable vox.CreateTime.NewKey contains the key of the newly created time block.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Example:

The following example creates an entry in the database indicating that from 8:00 a.m. to 9:00 a.m. between the dates of 07/20 and 07/24 each year, the MyMethod method can be used to tell Joe.

```
ADDRESS VOX "CreateTime ENTITY(Joe) BGNDATE(07/20) ENDDATE(07/24) BGNTIME(08:00)
ENDTIME(09:00) TELL(MyMethod)"
```

DESTROYENTITY Command

The DESTROYENTITY command destroys an existing entity. However, you cannot destroy an entity if it contains associated data or if other entities forward or escalate to this entity.

Associated data includes method level parameters, parameter overrides at the entity level or time block level, and the time blocks themselves. Delete associated data in this order:

- All parameter overrides
- Method level parameters
- All time blocks

You can change the destination of or delete those entities that forward or escalate to this entity. For more information, see the descriptions of the DESTROYPARM (see page 184) and DESTROYTIME (see page 185) commands.

This command has the following format:

ADDRESS VOX "DESTROYENTITY KEY(*key*)
[CASCADE(YES|NO)]"

KEY

Specifies the *key* of the entity to be deleted.

CASCADE

(Optional) Values are:

YES

Destroy all other items in the database that refer to this item.

NO

Do **not** destroy any other items in the database that refer to this item.

Default: NO

For example, if you specify CASCADE(YES) when deleting an entity, all of the data associated with the entity (time blocks, methods, personal and time block-specific parameter overrides) are deleted. If you specify CASCADE(NO) for an entity that has associated data, the command fails.

Return Information:

After DESTROYENTITY executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Usage Notes:

For additional information about obtaining associated data and entities that forward and escalate to this entity, see the following REXX programs that are located in the CA Automation Point installation directory \sample\nm:

- listent.rex
- listparm.rex
- lstforto.rex
- listtime.rex

Example:

Assume that Joe's entity key is 100001111. The following example removes the entry for entity Joe from the database:

```
ADDRESS VOX "DestroyEntity KEY(100001111)"
```

DESTROYLOGIN Command

The DESTROYLOGIN command destroys an existing login.

This command has the following format:

```
ADDRESS VOX "DESTROYLOGIN KEY(key)"
```

KEY

Specifies the *key* of the login to be deleted.

Return Information:

After DESTROYLOGIN executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Example:

The following example removes the login JSMITH from the database:

```
ADDRESS VOX "DESTROYLOGIN KEY(800001006)"
```

DESTROYMETHOD Command

The DESTROYMETHOD command destroys an existing method. However, you cannot destroy a method if it has associated data, such as time blocks, method level parameters, or parameter overrides, at either the entity level or the time block level. Delete the associated data in this order:

1. All parameter overrides
2. Method level parameters
3. All time blocks

Note: For more information, see the descriptions of the DESTROYPARM (see page 184) and DESTROYTIME (see page 185) commands.

This command has the following format:

```
ADDRESS VOX "DESTROYMETHOD KEY(key)
[CASCADE(YES|NO)]"
```

KEY

Specifies the *key* of the method to be deleted.

CASCADE

(Optional) Values are:

YES

Destroy all other items in the database that refer to this item.

NO

Do *not* destroy any other items in the database that refer to this item.

Default: NO

For example, if you specify CASCADE(YES) when deleting a method, all associated parameters and time blocks are deleted. If you specify CASCADE(NO) for a method that has associated parameters and time blocks, the command fails.

Return Information:

After DESTROYMETHOD executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Usage Notes:

For more information about obtaining associated data, see the following REXX programs located in the CA Automation Point installation directory \sample\nm:

- listparm.rex
- listtime.rex

Example:

The following example removes the entry method with key 200002222 from the database:

```
ADDRESS VOX "DestroyMethod KEY(200002222)"
```

DESTROYPARM Command

The DESTROYPARM command destroys an existing parameter. However, you cannot destroy a method parameter if it has overrides at either the entity level or the time block level. You must first delete all parameter overrides.

This command has the following format:

```
ADDRESS VOX "DESTROYPARM KEY(key)
[CASCADE(YES|NO)]"
```

KEY

Specifies the *key* of the parameter to be deleted.

CASCADE

(Optional) Valid values are:

YES

Destroy all other items in the database that refer to this item.

NO

Do *not* destroy any other items in the database that refer to this item.

Default: NO

For example, if you specify CASCADE(YES) when deleting a parameter, all of the personal and time block-specific parameter overrides are deleted. If you specify CASCADE(NO) for a parameter that has personal and time block-specific parameter overrides, the command fails.

Return Information:

After DESTROYPARM executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Usage Notes:

For more information about obtaining associated data, see the listparm.rex REXX program.

Example:

The following example removes the entry for the parameter with key 300001030 from the database:

```
ADDRESS VOX "DestroyParm KEY(300001030)"
```

DESTROYTIME Command

The DESTROYTIME command destroys an existing time block. However, you cannot destroy a time block if it has a parameter override. First you must delete the parameter override at the time block level.

This command has the following format:

ADDRESS VOX "DESTROYTIME KEY(*key*) [CASCADE(YES|NO)]"

KEY

Specifies the *key* of the time entry to be removed.

CASCADE

(Optional) Values are:

YES

Destroy all other items in the database that refer to this item.

NO

Do *not* destroy any other items in the database that refer to this item.

Default: NO

For example, if you specify CASCADE(YES) when deleting a time block, all of the time block-specific parameter overrides are deleted. If you specify CASCADE(NO) for a time block that has time block- specific parameter overrides, the command fails.

Return Information:

After DESTROYTIME executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Usage Notes:

For more information on obtaining associated data, see the listtime.rex REXX program.

Example:

The following example removes the entry for the time with key 400002267 from the database:

```
ADDRESS VOX "DestroyTime KEY(400002267)"
```

EPWCHECK Command

The EPWCHECK command checks the password for an entity.

This command has the following format:

ADDRESS VOX "EPWCHECK NAME(*name*)|KEY(*key*) EPW(*current password for entity*)"

NAME

Specifies the *name* of the entity to be checked.

KEY

Specifies the *key* of the entity to be checked.

EPW

Specifies the password to validate.

Return Information:

After EPWCHECK executes, it sets the special REXX return code variable RC.

- **If the command executes successfully,** the RC variable contains a value of zero.
- **If the command does not execute successfully,** the RC variable contains a nonzero value. Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Example:

The following example checks to see whether 1034 is the password for Fred:

ADDRESS VOX "EPWCheck NAME(Fred) EPW(1034)"

LISTENTITY Command

The LISTENTITY command lists all the data for one or more entities (except the password).

This command has the following format:

```
ADDRESS VOX "LISTENTITY NAME(name)|KEY(key)
 [CMDRESP(destination)]
 [PREFIX(varname)]"
```

NAME

Specifies the *name* of the entity to be listed. * indicates that you want data for all entities to be listed.

KEY

Specifies the *key* of the entity to be listed.

CMDRESP

(Optional) Directs return information, if any, to a specific destination. For a list of valid *destination* values, see the section ADDRESS VOX Return Information (see page 161).

Default: REXX

PREFIX

(Optional) Specifies the name of the REXX stem variable (other than the default name) that contains the return information for the command.

For information about changing the default variable name, see the section ADDRESS VOX Return Information (see page 161).

Default: VOX.LISTENTITY

Return Information:

After LISTENTITY executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. The VOX.listentity.0 variable contains the number of lines of information returned. Each variable from VOX.listentity.1 to VOX.listentity.*n* contains a line of information. *n* represents the last line of return information.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. The VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Usage Note:

For more information, see the sample REXX program listent.rex located in the CA Automation Point installation directory \sample\nm.

LISTFIND Command

The LISTFIND command lists the time blocks that are active for one or more entities at a given date and time. The list is actually a tree and can contain any level of nesting.

This command has the following format:

```
ADDRESS VOX "LISTFIND NAME(name)|KEY(key) DOW(dow)|DATE(mm/dd) TIME(hh:mm)
  [CMDRESP(destination)]
  [MTUP (A|profile)]
  [PREFIX(varname)]"
```

NAME

Specifies the *name* of the entity for which a find tree is to be created.

KEY

Specifies the *key* of the entity for which a find tree is to be created.

DOW

Specifies the day for which you want to search.

Days are specified as the first three characters of their English names. You can specify only one day. If you specify this operand, you cannot specify the DATE operand.

DATE

Specifies the date (no year is specified) for which you want to search.

If you specify this operand, you cannot specify the DOW operand.

TIME

Specifies, in military time format (00:00 to 23:59), the time for which you want to search.

CMDRESP

(Optional) Directs return information, if any, to a specific destination. For a list of valid *destination* values, see the section ADDRESS VOX Return Information (see page 161).

Default: REXX

MTUP

(Optional) Use the Methods to Use Profile (MTUP) operand to specify which methods are attempted for a particular instance of an NMFIND notification request. The method type code for each method is defined in the Notification Manager database under the TYP parameter for the method. Before attempting to notify a contact using a scheduled method, Notification Manager compares the value its method type code with the profile specified on the MTUP operand. If the method type code is not part of the MTUP profile, notification is not attempted, and the next scheduled method is compared against the MTUP profile. If the method type code is part of the MTUP profile, then the notification is attempted using that method.

The values for this operand are:

profile

Any combination of method type codes B through W.

A

All method types specified for all active schedules. No comparisons are made, and all methods are attempted.

Default: A

PREFIX

(Optional) Specifies the name of the REXX stem variable (other than the default name) that contains the return information for the command.

For information about changing the default variable name, see the section ADDRESS VOX Return Information (see page 161).

Default: VOX.LISTFIND

Return Information:

After LISTFIND executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. The VOX.listfind.0 variable contains the number of lines of information returned. Each variable from VOX.listfind.1 to VOX.listfind.*n* contains a line of information. *n* represents the last line of return information. Consider the following REXX variables:

  VOX.ListFind.BgnTime

  The latest start time of any time block in the call tree

  VOX.ListFind.EndTime

  The earliest end time of any time block in the call tree. This allows the caller to see the range of time when this tree is valid.

- **If the command does not execute successfully**, the RC variable contains a nonzero value. The VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Usage Note:

For further details, refer to the sample REXX programs listfind.rex and listmtup.rex located in the CA Automation Point installation directory \sample\nm.

Example:

The following example checks the tree for Fran Smith at 10:00 a.m. on October 11:

```
ADDRESS VOX "LISTFIND NAME(Fran Smith) TIME(10:00) DATE(10/11)"
```

LISTFORTO Command

The LISTFORTO command lists all the time blocks that forward to a given entity (regardless of the time and date). The list is actually a tree and can contain any level of nesting.

This command has the following format:

```
ADDRESS VOX "LISTFORTO NAME(name)|KEY(key)
 [CMDRESP(destination)]
 [PREFIX(varname)]"
```

NAME

Specifies the *name* of the entity for which the tree is to be created.

KEY

Specifies the *key* of the entity for which a tree is to be created.

CMDRESP

(Optional) Directs return information, if any, to a specific destination. For a list of valid *destination* values, see the section ADDRESS VOX Return Information (see page 161).

Default: REXX

PREFIX

(Optional) Specifies the name of the REXX stem variable (other than the default name) that contains the return information for the command.

For information about changing the default variable name, see the section ADDRESS VOX Return Information (see page 161).

Default: VOX.LISTFORTO

Return Information:

After LISTFORTO executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. The VOX.listforto.0 variable contains the number of lines of information returned. Each variable from VOX.listforto.1 to VOX.listforto.*n* contains a line of information. *n* represents the last line of return information.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. The VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Usage Note:

For further details, see the sample REXX program lstforto.rex, located in the CA Automation Point installation directory \sample\nm.

Example:

The following example checks the tree for Fran Smith at 10:00 a.m. on October 11:

```
ADDRESS VOX "LISTFORTO NAME(Fran Smith)"
```

LISTLOGIN Command

The LISTLOGIN command lists all the data for one or more logins.

This command has the following format:

```
ADDRESS VOX "LISTLOGIN USERNAME(name)|KEY(key)
 [CMDRESP(destination)]
 [PREFIX(varname)]"
```

USERNAME

Specifies the *name* of the login to be listed. * indicates that you want data for all logins to be listed.

KEY

Specifies the *key* of the login to be listed.

CMDRESP

(Optional) Directs return information, if any, to a specific destination. For a list of valid *destination* values, see the section ADDRESS VOX Return Information (see page 161).

Default: REXX

PREFIX

(Optional) Specifies the name of the REXX stem variable (other than the default name) that contains the return information for the command.

For information about changing the default variable name, see the section ADDRESS VOX Return Information (see page 161).

Default: VOX.LISTLOGIN

Return Information:

After LISTLOGIN executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. The VOX.listlogin.0 variable contains the number of lines of information returned. Each variable from VOX.listlogin.1 to VOX.listlogin.*n* contains a line of information. *n* represents the last line of return information.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. The VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Usage Notes:

For further details, refer to the sample REXX program listlogin.rex located in the CA Automation Point installation directory \sample\nm.

LISTMETHOD Command

The LISTMETHOD command lists all the data for one or more methods.

This command has the following format:

ADDRESS VOX "LISTMETHOD NAME(*name*)|KEY(*key*) [CMDRESP(*destination*)]
[PREFIX(*varname*)]"

NAME

Specifies the *name* of the method to be listed. An asterisk (*) indicates that you want data for all methods to be listed.

KEY

Specifies the *key* of the method to be listed.

CMDRESP

(Optional) Directs return information, if any, to a specific destination. For a list of valid *destination* values, see the section ADDRESS VOX Return Information (see page 161).

Default: REXX

PREFIX

(Optional) Specifies the name of the REXX stem variable (other than the default name) that contains the return information for the command.

For information about changing the default variable name, see the section ADDRESS VOX Return Information (see page 161).

Default: VOX.LISTMETHOD

Return Information:

After the LISTMETHOD command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. The VOX.listmethod.0 variable contains the number of lines of information returned. Each variable from VOX.listmethod.1 to VOX.listmethod.*n* contains a line of information. *n* represents the last line of return information.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. The VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Usage Note:

For further details, refer to the sample REXX program listmeth.rex located in the CA Automation Point installation directory \sample\nm.

LISTPARM Command

The LISTPARM command lists all the data for one or more parameters. Specify at least one of the NAME, KEY, METHOD, and ITEM operands. If you specify more than one operand, the search becomes more restrictive.

This command has the following format:

ADDRESS VOX "LISTPARM NAME(*parm name*)|KEY(*parm key*) METHOD(method *name* or *key*) ITEM(*entity name or entity key*) [CMDRESP(*destination*)] [PREFIX(*varname*)]"

NAME

Specifies the *name* of the parameter to be listed. An asterisk (*) indicates that you want data for all parameters to be listed.

KEY

Specifies the *key* of the parameter to be listed.

METHOD

Specifies the name or the key of the method to which the parameters are to belong.

ITEM

Specifies the *key* or *name* of the entity to which the parameters are to belong.

CMDRESP

(Optional) Directs return information, if any, to a specific destination. For a list of valid *destination* values, see the section ADDRESS VOX Return Information (see page 161).

Default: REXX

PREFIX

(Optional) Specifies the name of the REXX stem variable (other than the default name) that contains the return information for the command.

For information about changing the default variable name, see the section ADDRESS VOX Return Information (see page 161).

Default: VOX.LISTPARM

Usage Notes:

Parameter entries are divided into three increasingly specific classes:

- **The method class**-Those entries that apply any time the method is used.
- **The entity class**-Those entries that apply any time a method is used for a particular entity.
- **The time class**-Those entries that apply only for a particular time block, which implies a specific entity and method.

Here are some procedures for retrieving data from LISTPARM:

- If you specify a method and nothing else, you will receive *all* entries in all three classes for that method.
- If you want just the entries in the method class for the method, specify ITEM(-1).
- If you specify a method and an entity, you receive all entries in the entity and time classes.
- If you want just the entries in the entity class, specify the negation of the key of the entity in the ITEM() operand.
- If the method key is negative, the item key is for a time block, and the parameter name is *, the output will be the format that NMFIND needs (all time block-level entries, then all the entity-level entries for the entity that owns the time block, then all the method-level entries for the method).

Return Information:

After the LISTPARM command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. The VOX.listparm.0 variable contains the number of lines of information returned. Each variable from VOX.listparm.1 to VOX.listparm.*n* contains a line of information. *n* represents the last line of return information.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. The VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Example:

The following example lists the data for PAGERID for VOICE1 for Fran Smith:

```
ADDRESS VOX "LISTPARM NAME(PAGERID) METHOD(VOICE1) ITEM(FRAN SMITH)"
```

For further details, refer to the sample REXX program listparm.rex, located in the CA Automation Point installation directory \sample\nm.

LISTPERGRPS Command

The LISTPERGRPS command lists the personal groups for a contact. A personal group is defined as a group of which the contact is a member. This command can also be used to list all the members of a specific group.

This command has the following format:

```
ADDRESS VOX "LISTPERGRPS NAME{({name|*}) | KEY(key)}
 [GRP(*|name|key)]
 [PREFIX(varname)]"
```

NAME

Specifies the *name* of the entity whose personal groups are to be listed.

An asterisk (*) returns a list of contacts that are members of the specified group.

Note: An asterisk (*) is valid only when the GRP operand is defined and the GRP operand is not defined as the wildcard character.

KEY

Specifies the *key* of the entity whose groups are to be listed.

GRP

(Optional) Specifies the group name or key to check for the membership of contacts.

An asterisk (*) returns a list of all groups of which the contact specified by NAME is a member.

Note: An asterisk (*) is valid only when the NAME operand is not defined as the wildcard character.

Default: *

CMDRESP

(Optional) Directs return information, if any, to a specific destination. For a list of valid *destination* values, see the section ADDRESS VOX Return Information (see page 161).

Default: REXX

PREFIX

(Optional) Specifies the name of the REXX stem variable (other than the default name) that contains the return information for the command.

For information about changing the default variable name, see the section ADDRESS VOX Return Information (see page 161).

Default: VOX.LISTPERGRPS

Usage Note:

For more information on this command, see the sample REXX program listpergrps.rex in the CA Automation Point installation folder \sample\nm.

Return Information:

After the LISTPERGRPS command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. The VOX.listpergrps.0 variable contains the number of lines of information returned. Each variable from VOX.listpergrps.1 to VOX.listpergrps.*n* contains a line of information. *n* represents the last line of return information.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. The VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Examples:

- The following command lists any group of which Chris Smith is a member:

  ADDRESS VOX "LISTPERGRPS NAME(Chris Smith)"

- The following command lists all the members of the Weekend Group:

  ADDRESS VOX "LISTPERGRPS NAME(*) GRP(Weekend Group)"

LISTTIME Command

The LISTTIME command lists all of the data for one or more time blocks. Specify one of the NAME or KEY operands. If the entity key or name is provided, all entries for the entity are listed. If the times table key is provided, just that entry is listed.

This command has the following format:

ADDRESS VOX "LISTTIME NAME(entity *name*)|KEY(*entity*|*time key*)
 [CMDRESP(*destination*)]
 [PREFIX(*varname*)]"

NAME

Specifies the *name* of the entity whose time entries are to be listed. An asterisk (*) indicates that you want all times for all entities to be listed.

KEY

Specifies the *key* of the entity for whom all times are to be listed or the key of the time that is to be listed.

CMDRESP

(Optional) Directs return information, if any, to a specific destination. For a list of valid *destination* values, see the section ADDRESS VOX Return Information (see page 161).

Default: REXX

PREFIX

(Optional) Specifies the name of the REXX stem variable (other than the default name) that contains the return information for the command.

For information about changing the default variable name, see the section ADDRESS VOX Return Information (see page 161).

Default: VOX.LISTTIME

Return Information:

After the LISTTIME command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. The VOX.listtime.0 variable contains the number of lines of information returned. Each variable from VOX.listtime.1 to VOX.listtime.*n* contains a line of information. *n* represents the last line of return information.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. The VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Usage Note:

For further details, refer to the sample REXX program listtime.rex, located in the CA Automation Point installation directory \sample\nm.

NMDBMERGE Command

The NMDBMERGE command merges a previously exported copy of a database into the current Notification Manager database. Names (entity name, method name, time block name, and so on) are used to check for duplicate data. If a duplicate is found, the original data is replaced with the imported data. Otherwise, a new entry is created in the database. The NMDBMERGE command merges existing data with imported data.

This command has the following format:

```
ADDRESS VOX "NMDBMERGE PATH(path)"
```

PATH

Specifies the name of the path from which the exported database is retrieved.

Return Information:

After the NMDBMERGE command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains a non-zero value. Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Usage Notes:

- Do not use the database when issuing the NMDBMERGE command. To guarantee that the database is not being used, shut down the CA-AP NM Gateway Server service before running the NMDBMERGE command.
- CA does not support importing a Notification Manager database that has been exported from a newer release of CA Automation Point into an older release of CA Automation Point.

Example:

The following example imports a copy of an exported database that resides in the G:\TEMP_EXP directory:

```
ADDRESS VOX "NMDBMERGE PATH(G:\TEMP_EXP)"
```

NMEXPORT Command

The NMEXPORT command creates a backup copy of a Notification Manager database in a format that you can easily move or copy to another system.

This command has the following format:

```
ADDRESS VOX "NMEXPORT PATH(path) [CREATE(YES|NO)]"
```

PATH

Specifies the name of the path on which the exported database is stored.

CREATE

(Optional) Values are:

YES

Create the specified directory if it does not exist.

NO

Do not create the specified directory if it does not exist.

Default: NO

Return Information:

After the NMEXPORT command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully,** the RC variable contains a value of zero.
- **If the command does not execute successfully,** the RC variable contains a non-zero value. Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Usage Note:

It is important that the database is not being used when issuing the NMEXPORT command. To guarantee this, shut down the CA Automation Point NM Gateway Server service before running the NMEXPORT command.

Example:

The following example exports a copy of the current database into the G:\TEMP_EXP directory:

```
ADDRESS VOX "NMEXPORT PATH(G:\TEMP_EXP)"
```

NMIADDCALLER Command

This command adds an entity to the list of entities that are allowed to call in on an item.

This command has the following format:

ADDRESS VOX "NMIADDCALLER ENTITY(*key*) ITEM(*item*)"

ENTITY

Specifies the *key* of the entity.

ITEM

Specifies the *item* number.

Return Information:

After the NMIADDCALLER command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. The VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Example:

The following example tells the entity with key 100001000 to call in on item 1234:

```
ADDRESS VOX "NmiAddCaller ENTITY(100001000) ITEM(1234)"
```

NMIANSWER Command

This command sets or retrieves the answer for an item in the Notification Manager database.

This command has the following format:

ADDRESS VOX "NMIANSWER ITEM(*item*) ANSWER(*ansnum*) [ANSENTITY(key)]"

ITEM

Specifies the *item* number.

ANSWER

Specifies a number that is the answer to the item, from 0 to 9. To set the answer for the item, specify a value from 1 to 9. To retrieve the answer for the item, specify the value 0. The answer for the problem is returned in the return code. If the problem has not been answered, the answer is 0. If the problem has already been answered, the new answer is rejected and the value of the existing answer is returned.

ANSENTITY

(Optional) Specifies the key of the answering entity. This key is used to record the entity that provided the answer to the notification request.

Return Information:

After the NMIANSWER command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains the answer to the specified item (a numeric digit from 1 to 9). If RC=0, the specified item has not been answered.
- **If the command does not execute successfully**, the VOX.ERROR variable contains the complete VOX error message. This message includes the four-digit return code and error message text

Example:

The following example answers item 1234, and then checks to see whether it receives the same answer on its second call. (If the SAY statement runs, there is a problem).

```
ADDRESS VOX "NMIANSWER ITEM(1234) ANSWER(5) ANSENTITY(100001001)"
answer = rc
ADDRESS VOX "NMIANSWER ITEM(1234) ANSWER(0)"
IF answer<>rc THEN SAY "Expected" answer "received" rc
```

NMICHECKCALLER Command

This command checks to see whether an entity has been permitted to listen to or answer this item by a call to NMIADDCALLER. This sets the return code to 0 if the entity is authorized.

This command has the following format:

```
ADDRESS VOX "NMICHECKCALLER ENTITY(key) ITEM(item)"
```

ENTITY

Specifies the *key* of the entity.

ITEM

Specifies the *item* number.

Return Information:

After the NMICHECKCALLER command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. The entity is permitted to answer this item.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. The VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Example:

The following example tests whether the entity with key 100001000 is permitted to call in on item 1234:

```
ADDRESS VOX "NMICHECKCALLER ENTITY(100001000) ITEM(1234)"
IF rc=0 THEN SAY "Congratulations, you are authorized."
```

NMIGETITEM Command

This command retrieves the ask text and tell text for an item from the Notification Manager database.

This command has the following format:

ADDRESS VOX "NMIGETITEM ITEM(*item*) [CMDRESP(*destination*)] [PREFIX(*varname*)]"

ITEM

Specifies the *item* number.

CMDRESP

(Optional) Directs return information, if any, to a specific destination. For a list of valid *destination* values, see ADDRESS VOX Return Information in this chapter.

Default: REXX

PREFIX

(Optional) Specifies the name of the REXX stem variable (other than the default name) that contains the return information for the command.

For information about changing the default variable name, see ADDRESS VOX Return Information in this chapter.

Default: VOX.NMIGETITEM

Return Information:

After the NMIGETITEM command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. The VOX.NMIGETITEM variable contains the tell and ask information for the specified item.
- **If the command does not execute successfully**, the RC variable contains a nonzero value. The VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and error message text.

Example:

The following example retrieves the values of the TELL and ASK parameters for item 1234:

```
ADDRESS VOX "NMIGETITEM ITEM(1234)"
PARSE VAR vox.nmigetitem y 2 tellmsg (y) askmsgs (y).
Say "The TELL text for this item is" tellmsg
Say "The ASK text for this item is" askmsgs
```

NMILISTANSWERS Command

This command lists the answers that are specified using the NMFIND ASK parameter (if any) for the specified notification request.

This command has the following format:

```
ADDRESS VOX "NMILISTANSWERS ITEM(item) [CMDRESP(destination)] [PREFIX(varname)]"
```

ITEM

Specifies the item number.

CMDRESP

(Optional) Directs return information, if any, to a specific destination. For a list of valid destination values, see ADDRESS VOX Return Information (see page 161).

Default: REXX

PREFIX

(Optional) Specifies the name of the REXX stem variable (other than the default name) that contains the return information for the command.

For information about changing the default variable name, see ADDRESS VOX Return Information (see page 161).

Default: VOX.NMILISTANSWERS

Usage Note:

For more information on this command, see the sample REXX program listanswers.rex in the CA Automation Point installation folder \sample\nm.

Return Information:

After the NMILISTANSWERS command executes, it sets the special REXX return code variable RC.

- If the command executes successfully, the RC variable contains a value of zero. The VOX.nmilistanswers.0 variable contains the number of lines of information returned. Each variable from VOX.nmilistanswers.1 to VOX.nmilistanswers.n contains a line of information. n represents the last line of return information.
- If the command does not execute successfully, the RC variable contains a nonzero value. The VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and error message text.

Example:

The following example retrieves the list of all answers that are specified for notification request item number 1234:

```
ADDRESS VOX “NMILISTANSWERS ITEM(1234)”
```

NMILISTCALLERS Command

This command lists status information about every notification attempt that is made during the processing of the specified notification request.

This command has the following format:

```
ADDRESS VOX "NMILISTCALLERS ITEM(item) [CMDRESP(destination)] [PREFIX(varname)]"
```

ITEM

Specifies the item number.

CMDRESP

(Optional) Directs return information, if any, to a specific destination. For a list of valid destination values, see ADDRESS VOX Return Information (see page 161).

Default: REXX

PREFIX

(Optional) Specifies the name of the REXX stem variable (other than the default name) that contains the return information for the command.

For information about changing the default variable name, see ADDRESS VOX Return Information (see page 161).

Default: VOX.NMILISTCALLERS

Usage Note:

For more information on this command, see the sample REXX program listcallers.rex in the CA Automation Point installation folder \sample\nm.

Return Information:

After the NMILISTCALLERS command executes, it sets the special REXX return code variable RC.

- If the command executes successfully, the RC variable contains a value of zero. The VOX.nmilistcallers.0 variable contains the number of lines of information returned. Each variable from VOX. nmilistcallers.1 to VOX. nmilistcallers.n contains a line of information. n represents the last line of return information.
- If the command does not execute successfully, the RC variable contains a nonzero value. The VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and error message text.

Example:

The following example retrieves the list of all notification attempts made for notification request item number 1234:

```
ADDRESS VOX “NMILISTCALLERS ITEM(1234)”
```

NMILISTITEMS Command

This command lists detailed status information about previously issued notification requests. Each command operand is optional, and the combination of operands is used to define and restrict the items the query returns.

This command has the following format:

```
ADDRESS VOX "NMILISTITEMS
 [ITEM(item)]
 [ITEMSTATUS(status)]
 [NOTIFIER(name)]
 [RECENTITY(name)]
 [ANSENTITY(name)]
 [BGNTST(timestamp)]
 [ENDTST(timestamp)]
 [MAXITEMS(number)]
 [CMDRESP(destination)]
 [PREFIX(varname)]"
```

ITEM

(Optional) Specifies the item number.

ITEMSTATUS

(Optional) This operand is used to specify the notification request status that is used to restrict the query results. By specifying an item status, only those notification requests that are currently in the specified status are returned by this query. This operand accepts the following status values:

0

Initializing status

1

Sending status

2

Sent status

3

Awaiting Response status

4

Responded status

5

Late Response status

6

No Response status

7

Failed status

NOTIFIER

(Optional) This operand is used to specify the name of the login that is used to initiate the notification request. By specifying a value for this operand, the query is restricted to show only those notification requests initiated by the specified login. You can specify one or more wildcard characters to match more than one login name.

RECENTITY

(Optional) This operand is used to specify the name of the contact that is used as the target for the notification request. By specifying a value for this operand, the query is restricted to show only those notification requests for which the specified contact was the initial recipient. You can specify one or more wildcard characters to match more than one contact name.

ANSENTITY

(Optional) This operand is used to specify the exact name of the contact that has permission to provide an answer to a notification request. By specifying a value for this operand, the query is restricted to show only those notification requests that the specified contact has permission to answer.

BGNTST

(Optional) This operand is used to specify the beginning timestamp for retrieving notification requests. By specifying a value for this operand, the query is restricted to show only those notification requests initiated after the date and time reflected by the specified timestamp. The timestamp value reflects the number of seconds elapsed since midnight, January 1, 1970.

ENDTST

(Optional) This operand is used to specify the ending timestamp for retrieving notification requests. By specifying a value for this operand, the query is restricted to show only those notification requests initiated before the date and time reflected by the specified timestamp. The timestamp value reflects the number of seconds elapsed since midnight, January 1, 1970.

MAXITEMS

(Optional) This operand is used to specify the maximum number of notification requests to retrieve from the database. By specifying a value for this operand, the query is restricted to show no more than the number of items that are reflected by the value of this operand.

CMDRESP

(Optional) Directs return information, if any, to a specific destination. For a list of valid destination values, see ADDRESS VOX Return Information in this chapter.

Default: REXX

PREFIX

(Optional) Specifies the name of the REXX stem variable (other than the default name) that contains the return information for the command.

For information about changing the default variable name, see ADDRESS VOX Return Information (see page 161).

Default: VOX.NMILISTITEMS

Usage Note:

For more information on this command, see the sample REXX program listitems.rex in the CA Automation Point installation folder \sample\nm.

Return Information:

After the NMILISTITEMS command executes, it sets the special REXX return code variable RC.

- If the command executes successfully, the RC variable contains a value of zero. The VOX.nmilistitems.0 variable contains the number of lines of information returned. Each variable from VOX. nmilistitems.1 to VOX. nmilistitems.n contains a line of information. n represents the last line of return information.
- If the command does not execute successfully, the RC variable contains a nonzero value. The VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and error message text.

Example:

This example shows all notification requests currently in the Awaiting Response state that were sent to the contact “Joe”:

```
ADDRESS VOX “NMILISTITEMS ITEMSTATUS(3) RECENTITY(Joe)"
```

NMIMPORT Command

The NMIMPORT command imports an exported copy of a Notification Manager database.

Important: This command destroys all existing data in the database before importing new data.

This command has the following format:

```
ADDRESS VOX "NMIMPORT PATH(path)"
```

PATH

Specifies the name of the path from which the exported database is retrieved.

Return Information:

After the NMIMPORT command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully,** the RC variable contains a value of zero.
- **If the command does not execute successfully,** the RC variable contains a non-zero value. Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Usage Notes:

- It is important that the database is not being used when issuing the NMIMPORT command. To guarantee this, shut down the CA-AP NM Gateway Server service before running the NMIMPORT command.
- CA does not support importing a Notification Manager database that has been exported from a newer release of CA Automation Point into an older release of CA Automation Point.

Example:

The following example imports a copy of an exported database that resides in the G:\TEMP_EXP directory:

```
ADDRESS VOX "NMIMPORT PATH(G:\TEMP_EXP)"
```

Notification Commands

The following sections describe the CA Automation Point commands for use with notification tasks.

PAGE Command

The PAGE command is a notification command that issues an alphanumeric page to designated personnel using a dial-up modem connection to an alphanumeric paging service, according to TAP protocol.

This command has the following format:

ADDRESS VOX "PAGE PAGERID(*number*) TONESTRING(*phonenumber*) MESSAGE(*message*)
[BATCH(YES|NO)]
[BAUDRATE(*baudrate*)]
[COMPORT(*portnumber*)|TAPIDEVICEID(*numeric tapi deviceid*)]
[DATABITS(*databits*)]
[INITSTRING(*modeminitstring*)]
[PAGERPW(*password*)]
[PARITY(NONE|EVEN|ODD|MARK|SPACE)]
[STOPBITS(*stopbits*)]
[SYSTEM(*sysname*)]"

PAGERID

Specifies the pager ID number of the remote party. This number must contain between four and ten tone digits. Valid characters are: 0 1 2 3 4 5 6 7 8 9 -

Default: There is no default.

TONESTRING

Specifies the telephone number or other special digits (*tonestring*) to dial. Valid characters are: 0 1 2 3 4 5 6 7 8 9 * # a b c d & - , T P M I X

Default: There is no default.

MESSAGE

Specifies the alphanumeric message to display on the beeper of the recipient. The maximum length of the pager ID plus the message must be less than 240 characters.

Default: There is no default.

COMPORT

(Optional) Specifies the serial communications port to use, regardless of an existing connection on another device.

Normally, if there is already an established connection to the specified pager service, notification server issues a new page over the existing connection to enhance performance. Specifying the COMPORT parameter overrides existing port connections.

Use any valid communications port name (COM1, COM2, COM3, and so on). COMPORT and TAPIDEVICEID are mutually exclusive.

Default: Selects the first available communications port enabled for use by the notification server.

Note: To view the list of available communications ports, see the Alphanumeric Paging Options dialog in Configuration Manager.

BATCH

(Optional) Valid values are:

YES

Executes the command by writing the command to an internal notification server input queue where it is processed in batch mode.

NO

Specifies that the issuing process is to wait for the return code and responses before returning to the issuer.

Note: If issued with BATCH(YES), the return code you receive indicates only whether the command was properly queued for batch mode execution-it is not the completion code for the command.

Default: NO

BAUDRATE

(Optional) Specifies the baud rate that your modem uses to connect to your paging service. Valid values are: 300, 1200, 2400, 4800, and 9600.

Default: 9600

DATABITS

(Optional) Specifies the number of data bits per character used by the modem. The TAP protocol specifies that this must be set to 7. Valid values are: 4, 5, 6, 7, and 8.

Default: 7

INITSTRING

(Optional) This modem initialization command string should include the default modem initialization strings and any additional modem commands.

Default: ATZ; AT&C1&D2; ATV1Q0X4; ATS0=0S2=128S7=55

PAGERPW

(Optional) Specifies the alphanumeric character password of the remote party. This six-character password is an access code for the pager service.

PARITY

(Optional) A method used by the modem for error checking. The TAP protocol specifies that this must be set to E. Valid values are: N (None), E (Even), O (Odd), M (Mark), S (Space)

Default: E

STOPBITS

(Optional) A number that represents the time between transmitted characters used by the modem. The TAP protocol specifies that this must be set to 1. Valid values are: 0 (for 1.5), 1, and 2.

Default: 1

SYSTEM

(Optional) Specifies the name of the system that is running the notification server to which you want to direct the command. The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name

TAPIDEVICEID

(Optional) Specifies the numeric ID of the Telephony Application Programming Interface (TAPI) device or modem, (installed on the notification server) you want to use. This parameter overrides existing connections on another device.

Normally, if there is already an established connection to the specified pager service, notification server issues a new page over the existing connection to enhance performance.

Note: To view all of the Telephony Application Programming Interface (TAPI) devices that are installed under Windows on the Notification Server, issue the GETTAPIDEVICELIST command.

TAPIDEVICEID and COMPORT are mutually exclusive.

Default: The first available TAPI device

Notes:

- The modem must be properly installed within Windows. See the *Administrator Guide* for details.
- When you specify TAPIDEVICEID, the PAGE command overrides default settings configured in the Alphanumeric Paging Options dialog. It then initiates the page using the specified TAPI device ID, and uses TAPI to initialize the modem.
- When neither COMPORT nor TAPIDEVICEID is specified, the next available communications port is used to initiate a page, according to settings in the Alphanumeric Paging Options dialog.

Return Information:

After the PAGE command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command executes successfully but the RC variable contains a non-zero value of either 5331 or 5332**, the paging service final confirmation sequence did not conform to TAP protocol.

  Note: If the page completes successfully for a given paging service, these non-zero return codes can be disregarded.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5309, 5310, 5311, 5312, 5313, 5314, 5315, 5316, 5317, 5318, 5319, 5320, 5321, 5322, 5323, 5324, 5325, 5326, 5327, 5331, 5332. (See the *Message Reference Guide* for further information.) Additionally, VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Usage Note:

You can send pages to numeric pagers with the PAGE command if your paging service provider supplies a telephone number to support a TAP protocol dial-up.

Examples:

- The following code demonstrates how to repeatedly call a pager service as long as the reason that the PAGE request failed is either no answer from the pager service or a BUSY signal:

```
DO FOREVER
 ADDRESS VOX "PAGE baudrate(1200) comport(com2) pagerid(123-4567)"||,
          "tonestring(9-999-9999) message(Please call 555-1212)"
 IF rc=5322 |, /* BUSY signal */
   rc=5323    /* NO ANSWER   */
 THEN ITERATE
 ELSE LEAVE
END
IF rc=0 THEN SAY "Page got through."
ELSE SAY "Page failed with rc="rc "message="vox.error
```

- The following example demonstrates how you can find the first available TAPI device from the TAPI device list and send a page using its device ID.

```
/* Find first available tapi device from list and send page using
   its deviceID                                          */
 address VOX "GETTAPIDEVICELIST"
 if rc = 0 then
  if datatype(VOX.GETTAPIDEVICELIST.0) = "NUM" then
  do
   do i = 1 to VOX.GETTAPIDEVICELIST.0
    parse var VOX.GETTAPIDEVICELIST.i deviceid":"modemdescription
    if datatype(deviceid) = "NUM" then
     do
      say "Sending page using modem:"modemdescription

      /* Send a TAPI page with a specific deviceID */
      address VOX "PAGE MESSAGE(SAMPLE)" ,
          "PAGERID(111111) TONESTRING(5550987)" ,
          "TAPIDEVICEID("deviceid")"
      if rc <> 0 then say vox.error
      return
     end
   end
  end

 say "No TAPI devices available to send PAGE"
 exit
```

PAGE2WAY Command

The PAGE2WAY command is a notification command that issues an alphanumeric page to a designated page device using the Internet to relay these requests to the appropriate paging service. To receive page requests using this command, the paging service must support either the SNPP (Simple Network Paging Protocol) or WCTP (Wireless Communication Transfer Protocol) protocols.

This command has the following format:

```
ADDRESS VOX "PAGE2WAY
 PAGERID(number)
 PROVIDER(paging service name)
 {MESSAGE(message) | STATUS(message ID) | EXPIRE(message ID)}
 [ PAGERPW(password) ]
 [ MCRESPONSE(rsp1;rsp2;rsp3 ...) ]
 [ WAITSTATUS(QUEUED | DELIVERED | READ | REPLIED) ]
 [ WAIT(statuswait) ]
 [ SYSTEM(sysname) ]
 [ PREFIX(varname) ]
 [ CMDRESP(destination) ]"
```

PAGERID

Specifies the pager ID number (up to 15 digits).

PROVIDER

Specifies the user-defined name of the paging service (up to 256 characters). Paging service names are defined using the 2-Way Paging Setup dialog (located inside Configuration Manager) that will associate the paging service name with the required parameters used to connect to the paging service gateway system.

MESSAGE

Specifies the text message to be displayed on the 2-way pager device. Although this message can be up to 500 characters long, the message is still subject to message length limitations imposed by the pager service. Contact your pager service provider for details.

The MESSAGE, STATUS, and EXPIRE keywords are mutually exclusive.

STATUS

A text message ID assigned by the paging service (up to 128 characters). This message ID is returned with the initial MESSAGE request in the VOX.PAGE2WAY variable (by default) and is used to query the paging service for the status of a previously issued page request. The results of this STATUS request will also be placed in the VOX.PAGE2WAY variable (by default) and will be one of the following four values: QUEUED, DELIVERED, READ, or REPLIED <*Full Text Response*>. The <*Full Text Response*> section of the REPLIED value will be the text of the response sent from the 2-way device that corresponds to the page request. The MESSAGE, STATUS, and EXPIRE keywords are mutually exclusive.

Note: The list of supported status indicators may vary by pager service provider. Contact your pager service provider for details.

EXPIRE

A text message ID assigned by the paging service (up to 128 characters). This message ID is returned with the initial MESSAGE request in the VOX.PAGE2WAY variable (by default) and is used to remove the associated page item from the page item list maintained by the Notification Server. The Notification Server will periodically query the defined paging services for status updates to the previously issued page requests contained within the page item list. If no further status updates are required for a specific 2-way page request, you may use this keyword to manually remove (or expire) the associated page item. If this keyword is not used, the page item will remain active until either the reply to the page request has been returned in response to a STATUS request or the age of the page item exceeds the expire time. The MESSAGE, STATUS, and EXPIRE keywords are mutually exclusive.

PAGERPW

(Optional) Specifies the alphanumeric access code provided by the paging service, if required (up to 15 characters). Contact your paging service provider for details.

This operand is valid only in combination with the MESSAGE operand.

MCRESPONSE

(Optional) An optional list of alphanumeric multiple-choice responses (MCR) that, if specified, will be sent to the 2-way device in addition to the message text. The recipient can then choose one of these pre-programmed responses when sending a reply back to the paging service. The responses specified using this operand must be separated using a semicolon (";"), and the total length of the value of this operand must not exceed 512 characters (including separation characters). This operand is valid only in combination with the MESSAGE operand. It is at the discretion of the paging service to determine how many MCR responses are allowed, if the paging service supports this functionality.

WAITSTATUS

(Optional) Specifies the status text used to determine when the associated STATUS request should return with a page status value. This operand can be used to delay execution of your REXX program until either the status of the specified page request matches the status text specified for this operand, or the wait interval specified in the WAIT operand expires. This operand is only valid in combination with the STATUS operand.

Note: The list of supported status indicators may vary by pager service provider. Contact your pager service provider for details.

WAIT

(Optional) Specifies the amount of time to wait, in 1/10-second intervals, for the status of the specified page request to match the status specified by the WAITSTATUS operand. This operand is only valid in combination with both the WAITSTATUS operand and the STATUS operand.

Default: WAIT(6000) (Ten minutes)

SYSTEM

(Optional) Specifies the alphanumeric name of the system that is running the Notification Server to which you want to direct the command (up to 8 characters).

Default: The local system name.

PREFIX

(Optional) Specifies the name of a REXX stem variable (other than the default name) that contains the return information for the command.

Default: VOX.PAGE2WAY

CMDRESP

(Optional) Directs return information, if any, to a specific destination. For a list of valid *destination* values, see ADDRESS VOX Return Information in this chapter.

Default: REXX

Return Information:

After the PAGE2WAY command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**:
  - The RC variable contains a value of zero.
  - If the MESSAGE operand was used, the VOX.PAGE2WAY variable contains the message ID assigned by the paging service for the submitted page request. This message ID can be used later to query for status updates using the STATUS operand. If the STATUS operand was used, the VOX.PAGE2WAY variable will contain the current status of the page request (QUEUED, DELIVERED, READ, REPLIED <full text response>). If the EXPIRE operand was used, the VOX.PAGE2WAY variable will contain the previous status of the page request (QUEUED, DELIVERED, READ, REPLIED <full text response>).
- **If the command did not execute successfully**:
  - The RC variable contains one of the following values: 4150, 5104, 5126, 5127, 5220, 5222, 5223, 5224, 5225, 5226, 5227, 5228, 5229, 5230, 5231, 5232, 5233, 5234, and 5237. (See the *CA Automation Point Message Reference Guide* for more information.)
  - The VOX.ERROR variable contains the complete VOX error message, which includes the four digit return code and complete error message text.
  - The VOX.PAGE2WAY variable contains the last return code received from the paging service during the page submit attempt. This value may be useful in troubleshooting the cause of the failed page request. If the paging service is defined to use the SNPP communications protocol, see RFC1861 for a description of all possible return codes. If the provider is defined to use the WCTP protocol, point your Web browser to http://www.wctp.org/ and download the WCTP Specification v1.1 to view all the possible WCTP return codes.

Usage Notes:

The CA Automation Point server machine must have the ability to access the Internet in order to submit 2-way page requests to the configured paging service providers.

Examples:

- The following code demonstrates how to initiate a 2-way page request. In case of an error, this example also reports both the VOX error code and the last return code reported by the paging service.

```
ADDRESS VOX "PAGE2WAY PAGERID(5551212) PROVIDER(ABC Wireless) "||

        "MESSAGE('Hello from ABC Wireless ') "||
        "MCRESPONSE('Hello;Goodbye')"


IF rc = 0 THEN msgid = VOX.PAGE2WAY
ELSE
DO
  SAY "Page failed. VOX RC = "||rc||", Provider RC = "||VOX.PAGE2WAY||"."
  SAY "Error Msg = "||VOX.ERROR
END
```

- The following example demonstrates how to query the paging service for status updates. In this case, the REXX program will wait until the paging service reports that the page request has been delivered to the specified device (or 5 minutes, whichever occurs first).

```
ADDRESS VOX "PAGE2WAY PAGERID(5551212) PROVIDER(ABC Wireless) "||

        "MESSAGE('JES is down on SYS4')"


IF rc = 0 THEN
DO
  msgid = VOX.PAGE2WAY

  ADDRESS VOX "PAGE2WAY PAGERID(5551212) PROVIDER(ABC Wireless) "||
              "STATUS("||msgid||") WAITSTATUS(DELIVERED) WAIT(3000)"

  IF rc = 0 THEN
    SAY "The current status of message "||msgid||" is: "||VOX.PAGE2WAY
  ELSE
    SAY "Unable to query for page status updates. RC = "||rc||"."

END
ELSE SAY "Unable to submit page request. RC = "||rc||"."
```

SENDMAIL Command

The SENDMAIL command is a notification command that generates an e-mail message to designated personnel.

This command has the following format:

```
ADDRESS VOX "SENDMAIL TO(recipientlist) {MESSAGE(text)|VAR(rexxvariable)}
 [SUBJECT(text)]
 [SYSTEM(sysname)]
 [CC(recipientlist)]
 [BATCH(YES|NO)]
 [MAILID(text)]
 [ATTACHMENT(filename)]"
```

TO

Specifies the primary recipients of the mail message.

A *recipient list* is a list of one or more mail recipient names. The recipient names are text strings that the notification server attempts to resolve into e-mail addresses. If more than one name is listed, the names must be separated by a semicolon (;).

MESSAGE

Specifies the text (or body) of the mail message. The maximum length allowed is 240 characters. MESSAGE and VAR are mutually exclusive.

Line control characters can be inserted into the text for customized message viewing.

VAR

An optional method to specify the text (or body) of the mail message. The *rexxvariable* specified can be either a REXX variable or a stem variable. The REXX variable referenced by this VAR parameter can hold a maximum of 30,000 characters. Text from stem variables is concatenated (with one intervening blank) to form the mail text. MESSAGE and VAR are mutually exclusive. BATCH(YES) and VAR are mutually exclusive.

Line control characters can be inserted into the text assigned to the specified REXX variable for customized message viewing.

SUBJECT

(Optional) Subject of the mail message. The maximum length is 240 characters.

Default: There is no default.

SYSTEM

(Optional) Specifies the name of the system that is running the notification server to which you want to direct the command.

The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name

CC

(Optional) List of recipient names to receive a copy of the mail message.

If more than one name is listed, the names must be separated by a semicolon (;).

BATCH

(Optional) Valid values are:

YES

Executes the command by writing the command to an internal notification server input queue where it will be processed in batch mode.

NO

Specifies that the issuing process waits for the return code and responses before returning to the issuer.

Notes:

- If issued with BATCH(YES), the return code you receive indicates only whether the command was properly queued for batch mode execution-it is not the completion code for the command.
- BATCH(YES) and VAR are mutually exclusive.

Default: NO

MAILID

(Optional) User-supplied text to identify or track the mail message. This text is included in the last line of the identification section (generated by CA Automation Point) that is appended to the mail body. The maximum length is 40 characters.

Note: To use the MAILID option, you must enable Append Identification section to mail body on the Configure Mail dialog.

Default: There is no default.

ATTACHMENT

(Optional) User-supplied file which is to be attached to the mail message. The file name specified must be fully qualified and accessible from the Notification Server which is issuing the SENDMAIL command. Only one file can be specified per mail request. The maximum length of the file name including path is 512 characters.

Default: There is no default.

Return Information:

After the SENDMAIL command executes, it sets the special REXX return code variable RC.

- **If a command executes successfully**, the RC variable contains a value of zero.
- **If a command does not execute successfully**, the RC variable contains one of the following values: 5341, 5344, 5346, 5347, 5348. (See the *Message Reference Guide* for more information.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Usage Note:

The SENDMAIL command displays the 5345I and 5349W messages in the window with the default name of AP Notification Messages.

Examples:

- The following example illustrates how to use the SENDMAIL command:

```
msgvar = "This email is to inform you that z/OS system Y has been IPLed"
file="C:\TEMP\YourData.dat"
 address VOX SENDMAIL TO(John Smith) CC(Jane Jones) SUBJECT(Notification from AP) VAR(msgvar)
ATTACHMENT(path)
```

- The following example illustrates how to programmatically add line control characters to customize message viewing:

```
/* This is a sample program to take a screen   */
/* dump from session SYSA and send an email to */
/* support mailbox with the screen output.     */


/* NOTE: This message must be viewed by the    */
/* recipient using a fixed pitch font.         */

session_name= 'SYSA'
recipientlist='support'

ADDRESS AXC "GETSCRN SESSION("session_name") SCREEN(YES) PREFIX(LINE)"
IF rc <> 0 THEN DO
        SAY 'GETSCRN failed with rc=' rc
        EXIT
        END

/* Add a line feed character (hex 0a) at the end of each line on the screen dump   */
/* Prefix each line with a tab character (hex 09)                                  */

line_feed= '0a'x
tab= '09'x

DO i=1 to line.0
line.i=tab||line.i||line_feed
end

ADDRESS VOX "SENDMAIL TO("recipientlist") VAR(line.) SUBJECT(Screen dump from Session
"session_name")"
IF RC <> 0 THEN SAY 'SENDMAIL with screen dump failed with rc=' rc
```

Voice Commands

The following sections describe CA Automation Point ADDRESS VOX voice commands.

Valid Dialing Characters

The following ADDRESS VOX commands dial or send telephone keypad digits entered from your workstation's keyboard that you specify in the command statement.

- CALL
- CALLPLAY
- PLAYGETDIGITS
- SENDTONES

The following table lists the valid DTMF characters and their special functions (if any):

DTMF Character	Special Function (If applicable)
“0” through “9”	---
“*”	---
“#”	---
“a”	---
“b”	---
“c”	---
“d”	---
“-”	Ignored by command
“,”	Pause
“&”	Flash
“T”	DTMF mode
“P”	Pulse mode
“M”	MF mode
“L”	Wait for local dial tone
“I”	Wait for international dial tone
“X”	Wait for a special (or “extra”) dial tone

ANSWER Command

The ANSWER command waits for a period specified by the TIMEOUT parameter for an incoming telephone call on any one of a designated set of voice channels available at the time the command was invoked. The ANSWER command will not wait for incoming calls on voice channels that are marked as in-use at the time the command is invoked.

Furthermore, when the ANSWER command is invoked with the GROUP operand, any one channel satisfies the command and returns control to the caller with the handle to the answered channel. This answered channel must be released when you are done with the call (see the description of the RELEASECHANNEL command). This channel cannot reenter the set of channels on which any outstanding ANSWER command is waiting.

The ANSWER command can answer a call on:

- Any available voice channel
- Any available voice channel within a specified channel group
- A single, specific voice channel

When answering an incoming call, the ANSWER command:

- Sets the hook state of the receiving voice channel to off-hook
- Resets the play volume
- Clears the following channel attributes:
  - The digit buffer
  - The call progress analysis (CPA) parameters
  - The history of loop drop, silence-on, and silence-off events

This command has the following format:

```
ADDRESS VOX "ANSWER {CHANNEL(channelhandle)|GROUP(groupname|ALL)}
            [SYSTEM(sysname)]
            [ANSRING(ringnumber)]
            [TIMEOUT(waittime|0|-1)]
            [HOOKSTATE(hookstate)]
            [PREFIX(varname)]
            [CMDRESP(destination)]"
```

CHANNEL

Specifies the channel handle (*channelhandle*)-identifies a physical channel-that the GETCHANNEL command returns.

GROUP

A group name defining a group of specific, physical channel numbers that the ANSWER command monitors for an incoming call. (For more information about assigning channels to a group, see the description of the SETGROUP command.)

ALL enables the ANSWER command to monitor all groups (and, therefore, all channels) for a call.

SYSTEM

(Optional) Specifies the name of the system running the notification server to which you want to direct the command.

The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name

ANSRING

(Optional) Specifies the ring number on which to answer an incoming call.

Default: 1

TIMEOUT

(Optional) Specifies the maximum amount of time to wait for rings (in 1/10-second units). Or, you can specify 0 (zero) to return immediately if no ring exists or -1 to wait indefinitely.

Default: -1

HOOKSTATE

(Optional) Specifies the desired hook state after CA Automation Point detects a ring. Valid values are OFFHOOK and ONHOOK.

Default: OFFHOOK

PREFIX

(Optional) Specifies the name of a REXX stem variable (other than the default name) that contains the return information for the command.

For information about changing the default variable name, see ADDRESS VOX Return Information in this chapter.

Default: VOX.ANSWER

CMDRESP

(Optional) Directs return information to a specific destination (*destination*). For a list of valid destination values, see ADDRESS VOX Return Information in this chapter.

Return Information:

After the ANSWER command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. If the GROUP operand was specified, the REXX variable VOX.ANSWER contains the returned channel handle.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5105, 5112, 5117, 5155, 5199, 5203, 5204, 5206. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Examples:

- The following REXX code illustrates how to use the ANSWER command in a program that pages a user, and then answers when the user calls the CA Automation Point system in response to the page:

```
numpagerservice = '9,555-1900'
numtocall = '9,555-1234'

/* Acquire exclusive access to a voice channel. */

address vox 'getchannel channelnum(1) prefix(handle)'

/* Dial the digital pager service and leave number to call. */

address vox 'call channel('handle') tonestring('numpagerservice')'
address vox 'sendtones tonestring('numtocall')'


/* Wait up to 10 minutes for an incoming call. */

address vox 'answer channel('handle') timeout(6000)'

/* Get the callers User Identification number. */

address vox 'playgetdigits channel('handle') file(entruid.vox) count(8) singlestring(yes) prefix(userid)'

/* Get the caller's Personal Identification Number (PIN). */

address vox 'playgetdigits channel('handle') file(entrpin.vox) count(8) singlestring(yes) prefix(pin)'
...
```

```
/* Verify the remote user has entered a valid user-ID/pin combination.*/

address vox 'verifyuser userid('userid') pin('pin')'

if RC == 0 then do
  /* Explain the problem and list the options */
  /* available for resolving the problem.     */

  address vox 'playgetdigits channel('handle') file(problem.vox options.vox) prefix(response)'
  ...
end

/* Release the voice channel. */
address vox 'releasechannelchannel('handle')'
```

- The following REXX code illustrates how you can use the ANSWER command in a program that processes incoming calls destined for a help desk. When CA Automation Point receives a call, the HDCTRL.CMD program starts a program named HELPDESK.CMD to service the call automatically.

```
/* HDCTRL.CMD */

signal on halt Name DoExit

do forever
  /* Wait forever for an incoming call and */
  /* any channel within the HELPDESK group.*/

address vox 'answer group(helpdesk) hookstate(onhook) prefix(handle) timeout(600)'

  if RC == 0 then
  do
    /* Call answered. Start the HELPDESK.CMD */
    /* program to service the call.          */

    address vox 'startrexx program(helpdesk.cmd 'handle')'
  end
end

DoExit:

/* Exit from the comm*and shell so that CA Automation Point can "clean up." */

'@exit'
```

ANSWERPLAY Command

The ANSWERPLAY command waits for an incoming telephone call on a single channel and plays one or more prerecorded voice messages after answering the call. When answering an incoming call, the ANSWERPLAY command:

- Sets the hook state of the receiving voice channel to off-hook
- Resets the play volume
- Clears the following channel attributes:
  - The digit buffer
  - The call progress analysis (CPA) parameters
  - The history of loop drop, silence-on, and silence-off events

This command has the following formats:

Use the following syntax to answer a call and play a message from a non-indexed voice file:

```
ADDRESS VOX "ANSWERPLAY CHANNEL(channelhandle)
  FILE(filename_1[...filename_n])
  FILETYPE(NONINDEX)
  [SYSTEM(sysname)]
  [INTERRUPT(YES|NO)]
  [ANSRING(ringnumber)]
  [TIMEOUT(waittime|0|-1)]
  [HOOKSTATE(hookstate)]"
```

Use the following syntax to answer a call and play a message from a voice word library file:

```
ADDRESS VOX "ANSWERPLAY CHANNEL(channelhandle)
  FILE(filename)
  FILETYPE(WORDLIB)
  VAR(varname)
  [SYSTEM(sysname)]
  [INTERRUPT(YES|NO)]
  [ANSRING(ringnumber)]
  [TIMEOUT(waittime|0|-1)]
  [HOOKSTATE(hookstate)]"
```

CHANNEL

Identifies the physical channel (*channelhandle*) that the GETCHANNEL command returns.

Default: There is no default.

FILE

Specifies the name of a voice file or voice word library.

FILETYPE

Specifies the type of voice file to play. Valid values are:

- NONINDEX - A nonindexed voice file
- INDEX - An indexed voice file
- WORDLIB - A voice word library

Note: When specifying FILETYPE (WORDLIB), do not include a file extension on the FILE operand.

Default: NONINDEX

VAR

This operand is required only when specifying a voice word library on the FILETYPE operand.

For a voice word library, this specifies either a REXX regular variable or stem variable. When coding regular variables with the *var* keyword, only that variable is searched. With a stem variable, all numeric indexes for the stem are searched.

When coding a stem variable, *varname* should follow this format:

***varname*.0**

The number of words in your message.

varname*.1 through *varname*.*n

Each variable contains one word from the voice word library.

SYSTEM

(Optional) Specifies the name of the system running the notification server to which you want to direct the command.

The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name

INTERRUPT

(Optional) Specifies whether CA Automation Point interrupts the currently playing voice message if it receives a tone digit (that is, if the remote party presses a key on the telephone keypad). Valid values are:

YES

Allows the interruption.

NO

Prevents the interruption.

Default: YES

ANSRING

(Optional) Specifies the ring number on which to answer an incoming call.

Default: 1

TIMEOUT

(Optional) Specifies the maximum amount of time to wait for rings (in 1/10-second units). Or, you can specify 0 (zero) to return immediately if no ring exists or -1 to wait indefinitely.

Default: -1

HOOKSTATE

(Optional) Specifies the desired hook state after CA Automation Point plays a voice message. Valid values are OFFHOOK and ONHOOK.

Default: OFFHOOK

Return Information:

After the ANSWERPLAY command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5105, 5106, 5107, 5109, 5112, 5116, 5117, 5155, 5199, 5204, 5252, 5300. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Usage Notes:

Rather than issuing the ANSWER command followed by a PLAY command, CA recommends issuing the single ANSWERPLAY command whenever possible to improve performance.

Example:

The following REXX code illustrates how to use the ANSWERPLAY command to answer a call and offer the caller four menu options:

```
GREETING=1
...
MAINMNU_OPT1=5
MAINMNU_OPT2=6
MAINMNU_OPT3=7
...

/* Acquire exclusive access to a voice channel.*/

address vox 'getchannel channelnum(1)'



/* Wait forever for an incoming call and play */
/* greeting message followed by the main menu */
/* comprised of four separate voice messages  */
/* from within an indexed voice file.        */

msg.0=4
msg.1=1
msg.2=5
msg.3=6
msg.4=7
address vox 'answerplay channel('handle') file(helpdesk.vap) filetype(index) var(msg.)'
...

/* Release the voice channel. */
address vox 'releasechannel channel('handle')'
```

CALL Command

The CALL command initiates a telephone call by placing the voice channel in an off-hook state and dialing the tone string (telephone number).

This command has the following format:

```
ADDRESS VOX "CALL CHANNEL(channelhandle) TONESTRING(tonestring)
  [SYSTEM(sysname)]
  [NAME(CPAparameterset)]
  [RINGS(maxrings)]
  [RETRY(numretries)]
  [WAIT(secs)]
  [PREFIX(varname)]
  [CMDRESP(destination)]"
```

CHANNEL

Specifies the channel handle (*channelhandle*)-identifying some physical channel-that the GETCHANNEL command returns.

Default: There is no default.

TONESTRING

Specifies the telephone number or other special digits to dial.

Valid characters are: 0 1 2 3 4 5 6 7 8 9 * # a b c d & - , T P M L I X

SYSTEM

(Optional) Specifies the name of the system running the notification server to which you want to direct the command.

The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name

NAME

(Optional) Specifies the name of the call progress analysis (CPA) parameter set to use for dialing.

CPA set names can be created and configured using the Configuration Manager.

Default: EngineDefault

RINGS

(Optional) Specifies the approximate number of rings to allow before timing out the request and returning a “noanswer” call completion state. The number of rings specified is not an exact count of the actual rings on the telephone, but an estimation of how long the call should wait to get the given rings count.

Default: 4

RETRY

(Optional) Specifies the maximum number of call retries (after the initial call attempt) to establish a connection.

Default: 0

WAIT

(Optional) Specifies the amount of time to wait, in 1/10-second units, before redialing if you specify the RETRY operand.

Default: 600 (One minute)

PREFIX

(Optional) Specifies the name of the REXX stem variable (other than the default name) that contains the return information for the command.

For information about changing the default variable name, see ADDRESS VOX Return Information in this chapter.

Default: VOX.CALL

CMDRESP

(Optional) Directs return information to a specific destination (*destination*). For a list of valid destination values, see ADDRESS VOX Return Information in this chapter.

Default: REXX

Return Information:

After the CALL command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. The VOX.CALL variable contains the following:
  - **CADNCBRK**-Connection due to cadence break.
  - **LCDDROP**-Connection due to loop current.
  - **PVD**-Connection due to Positive Voice Detection.
  - **PAMD**-Connection due to Positive Answerplay Machine Detection.

  For more details about specific return information, see the ATDX_CONNTYPE() function definition in the Intel Dialogic documentation.

  The connection reason is useful when you need to adjust the call progress analysis (CPA) parameters of your voice card using the Configuration Manager to fine-tune your voice application. The Dialogic voice software determines the connection reason, but the algorithm is not always correct.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5112, 5117, 5155, 5199, 5204, 5301, 5302, 5303, 5304, 5305, 5306. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Usage Notes:

- The return information that the CALL command provides (see PREFIX) contains the reason that the command was able to establish a connection.
- See Valid Dialing Characters for information on valid DTMF dialing characters.

Example:

Suppose that you have written a REXX program to call a remote party and play a voice message. If the reason that the CALL command establishes a connection is Positive Voice Detection, a person probably answered the telephone. If the reason for the connection is Positive Answering Machine Detection, an answering machine or voice-mail system probably answered the call. In both cases, the default values for the CPA parameters are acceptable; your REXX code can simply provide for playing the prerecorded voice message at the appropriate time.

On the other hand, suppose that the office of the remote party is equipped with a PBX system and that the connection reason is Loop Drop. The voice message might already be in progress by the time the remote party hears it. In such a scenario, the PBX has sent a loop-drop signal before the remote party answered the call, causing the CALL command to complete and the subsequent lines in your REXX code to execute. To fix the problem, you may need to adjust some CPA parameters.

The following REXX code illustrates how to use the CALL command to call a remote party and report a problem:

```
phonenumber = '2000'

/* A T-1 trunk has dropped.                    */
/* Notify the telcom group immediately using the */
/* notification server located in the Los Angeles office. */

/* Acquire exclusive access to a voice channel.  */

address vox 'getchannel channelnum(1) prefix(handle) system(laengine)'
```

```
/* Call the main number of the telcom group */
/* to make sure that we did not get their   */
/* voice mail.                              */

address vox 'call channel('handle') tonestring('phonenumber') system(laengine)'
if vox.call != PAMD then
do
  /* Because some answering machines are so clear*/
  /* that they cannot be detected easily,       */
  /* make sure that we are connected to a person */
  /* by asking them to press 1.                 */

  address vox 'playgetdigits channel('handle') file(greeting.vox press1.vox) prefix(response) system(laengine)'

  if response == 1 then
  do
    /* Inform the remote party of     */
    /* the problem and offer options  */
    /* for responding to the problem. */

    address vox 'play channel('handle') file(t1down.vox options.vox) prefix(response) system(laengine)'
    ...
    /* Correcting problem according to */
    /* remote party's response         */
    ...
  end

end



/* Release the voice channel */

address vox 'releasechannel channel('handle')'
```

CALLPLAY Command

The CALLPLAY command initiates a telephone call and plays one or more prerecorded voice messages. It sets the hook state to off-hook, dials the tone string (following a successful connection), plays the voice message(s), and resets the hook state to on-hook.

This command has one of the following formats:

Use the following format to play a message from a non-indexed voice file:

```
ADDRESS VOX "CALLPLAY CHANNEL(channelhandle)
  TONESTRING(tonestring)
  FILE(filename_1[...filename_n])
  FILETYPE(NONINDEX)
  [SYSTEM(sysname)]
  [NAME(CPAparameterset)]
  [RINGS(maxrings)]
  [RETRY(numretries)]
  [WAIT(redialwait)]
  [INTERRUPT(YES|NO)]
  [HOOKSTATE(ONHOOK|OFFHOOK)]
  [PREFIX(varname)]
  [CMDRESP(destination)]"
```

Use the following format to play a message from a voice word library file:

```
ADDRESS VOX "CALLPLAY CHANNEL(channelhandle)
  TONESTRING(tonestring)
  FILE(filename)
  FILETYPE(WORDLIB)
  VAR(varname)
  [SYSTEM(sysname)]
  [RINGS(maxrings)]
  [RETRY(numretries)]
  [WAIT(redialwait)]
  [INTERRUPT(YES|NO)]
  [HOOKSTATE(ONHOOK|OFFHOOK)]
  [PREFIX(varname)]
  [CMDRESP(destination)]"
```

CHANNEL

Specifies the channel handle (*channelhandle*)-identifying some physical channel-that the GETCHANNEL command returns.

Default: There is no default.

TONESTRING

Specifies the telephone number or other special digits (*tonestring*) to dial.

Valid characters are: 0 1 2 3 4 5 6 7 8 9 * # a b c d & - , T P M L I X

FILE

Specifies the name of a voice file or voice word library.

Default: There is no default.

FILETYPE

Specifies the type of voice file to play. Valid values are:

NONINDEX

Specifies a non-indexed voice file.

WORDLIB

Specifies a voice word library.

Note: When specifying FILETYPE (WORDLIB), do not include a file extension on the FILE operand.

Default: NONINDEX

VAR

This operand is required only when specifying a voice word library on the FILETYPE operand.

For a voice word library, code either a REXX regular variable or stem variable. When coding regular variables with the *var* keyword, only that variable is searched.

With a stem variable, all numeric indexes for this stem will be searched. When coding a stem variable, *varname.* should follow this format:

***varname*.0**

The number of words in your message.

***varname*.1 through *varname*.n**

Each variable contains one word from the voice word library.

SYSTEM

(Optional) Specifies the name of the system running the notification server to which you want to direct the command.

The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name

NAME

(Optional) Specifies the name of the call progress analysis (CPA) parameter set to use for dialing. CPA set names can be created and configured using the Configuration Manager.

Default: EngineDefault

RINGS

(Optional) Specifies the approximate number of rings to allow before timing out the request and returning a “noanswer” call completion state. The number of rings specified is not an exact count of the actual rings on the telephone, but an estimation of how long the call should wait to get the given rings count.

Default: 4

RETRY

(Optional) Specifies the maximum number of call retries (after the initial call attempt) to establish a connection.

Default: 0

WAIT

(Optional) Specifies the amount of time to wait (in 1/10-second units) before redialing (if you specify the RETRY operand).

Default: 600 (One minute)

INTERRUPT

(Optional) Specifies whether CA Automation Point interrupts a currently playing voice message if it receives a tone digit (that is, if the remote party presses a key on the telephone keypad). Valid values are:

YES

Allows the interruption.

NO

Prevents the interruption.

Default: YES

HOOKSTATE

(Optional) Specifies whether CA Automation Point resets the hook state of the channel to on-hook after playing the voice message. Valid values are:

ONHOOK

Reset the channel to the on-hook state

OFFHOOK

Allow the channel to remain in the off-hook state

Default: OFFHOOK

PREFIX

(Optional) Specifies the name of the REXX stem variable (other than the default name) that contains the return information for the command.

For information about changing the default variable name, see ADDRESS VOX Return Information in this chapter.

Default: VOX.CALLPLAY

CMDRESP

(Optional) Directs return information to a specific destination (*destination*). For a list of valid *destination* values, see ADDRESS VOX Return Information in this chapter.

Default: REXX

Return Information:

After the CALLPLAY command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully,** the RC variable contains a value of zero. Additionally, the VOX.CALLPLAY variable contains the following:
  - **CAD**-Connection due to cadence break.
  - **LPC**-Connection due to loop current.
  - **PVD**-Connection due to Positive Voice Detection.
  - **PAMD**-Connection due to Positive Answerplay Machine Detection.

  For more details about specific return information, refer to the ATDX_CONNTYPE() function definition in the Dialogic manuals.

  The connection reason is useful when you need to adjust the call progress analysis (CPA) parameters of your voice card using the Configuration Manager to fine-tune your voice application.

  The Dialogic voice software determines the connection reason, but the algorithm is not always perfect.

- **If the command does not execute successfully,** the RC variable contains one of the following values: 5107, 5109, 5112, 5116, 5117, 5155, 5199, 5204, 5252, 5301, 5302, 5303, 5304, 5305, 5306. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Usage Notes:

Keep the following information in mind when using the CALLPLAY command:

- Instead of issuing the CALL command followed by a PLAY command, we recommend issuing the single CALLPLAY command whenever possible to improve performance.
- You can use the CALLPLAY command to play voice messages over a voice channel connected to an amplified speaker. The TONESTRING operand must contain the access code (tone-digit string) necessary to activate the speaker feature of the system.
- The return information that the CALLPLAY command provides (see PREFIX) indicates why the command was able to establish a connection.
- See Valid Dialing Characters for information on valid DTMF dialing characters.

Example:

The following REXX code illustrates how to use the CALLPLAY command to call a remote party and play one or more voice messages after making contact:

```
phonenumber = '412-555-2000'
/* Get exclusive access to a voice channel. */
address vox 'getchannel channelnum(1) prefix(handle)'

/* Attempt to reach the "on-call" operator at most three times,*/
/* waiting one minute between each attempt.                    */

address vox 'callplay channel('handle') tonestring('phonenumber') retry(3) wait(600) file('greeting.vox entruid.vox')'
```

```
/* Make sure that we are connected to the right person by */
/* verifying the operator's User-ID/PIN combination.      */

address vox 'getdigits channel('handle') count(8) singlestring(yes) prefix(userid)'

address vox 'playgetdigits channel('handle') file(entrpin.vox) singlestring(yes) prefix(pin)'

address vox 'verifyuser userid('userid') pin('pin')'

/* Deliver the message. */

address vox 'play channel('handle') file(message.vox)'

/* Release the voice channel. */

address vox 'releasechannel channel('handle')'
```

CLEAR Command

The CLEAR command clears the digit buffer of a voice channel or its call analysis parameters.

This command has the following format:

```
ADDRESS VOX "CLEAR CHANNEL(channelhandle)
 CLEARTYPE(cleartype)
 [SYSTEM(sysname)]"
```

CHANNEL

Specifies the channel handle (*channelhandle*)-identifying some physical channel-that the GETCHANNEL command returns.

CLEARTYPE

Specifies the type of CLEAR operation to perform. Valid values are:

DIGBUF

Clears the digit buffer (containing digits entered from the remote party's telephone keypad)

ANALYSIS

Clears the call progress analysis (CPA) parameters and resets them to the voice card manufacturer's defaults

ALL

Clears all of the above

SYSTEM

(Optional) Specifies the name of the system running the notification server to which you want to direct the command.

The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name

Return Information:

After the CLEAR command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5117, 5155, 5199, 5204. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Usage Note:

Determining when to clear a voice channel's digit buffer is application-specific. For example, an application that supports type-ahead dialing clears the digit buffer only on each call's call startup or termination. Generally, it is appropriate to clear the digit buffer immediately before initiating an I/O operation.

Example:

The following REXX code illustrates the CLEAR command. The program segment forces the remote party to listen to an entire voice message before allowing the remote party to enter a PIN number.

```
/* Get exclusive access to a voice channel. */

address vox 'getchannel channelnum(1) prefix(handle)'
...
/* Play a voice message, ensuring that the remote party listens */
/* to it in its entirety.                                       */

address vox 'play channel('handle') file(message.vox) interrupt(no)'
```

```
/* Clear the voice channel's digit buffer in case the remote   */
/* party entered digits (pressed keys on the telephone keypad) */
/* while the message was playing.                              */

address vox 'clear channel('handle') cleartype(digbuf)'
/* Get the remote party's PIN. */

address vox 'playgetdigits channel('handle') file(entrpin.vox) count(8) singlestring(yes) prefix(response)'
...

/* Release the voice channel. */

address vox 'releasechannel channel('handle')'
```

GETCHANNEL Command

The GETCHANNEL command retrieves an available voice channel and marks it as in-use and performs these additional channel management tasks:

- Prepares the channel for subsequent I/O operations by resetting the channel's call progress analysis (CPA) parameters
- Clears the channel's digit buffer
- Clears the channel's loop-drop, silence-on, and silence-off history
- Resets the channel's volume level (the level at which CA Automation Point plays voice files)
- Sets the channel's hook state to on-hook

Use the GETCHANNEL command with the RELEASECHANNEL command to serialize a voice channel's I/O activity.

This command has the following format:

```
ADDRESS VOX "GETCHANNEL {GROUP(groupname|ALL)|CHANNELNUM(channelnum)}
 [SYSTEM(sysname)]
 [TIMEOUT(waittime|0|-1)]
 [PREFIX(varname)]
 [CMDRESP(destination)]"
```

GROUP

Specifies the name of the group from which to retrieve an available voice channel and assign a channel handle. (This is the most common method for retrieving a channel.)

Specifying ALL enables the GETCHANNEL command to search all groups for an available channel.

CHANNELNUM

Specifies the number of a specific, physical channel number to use.

Valid values range from 1 through the number of lines installed.

SYSTEM

(Optional) Specifies the name of the system running the notification server to which you want to direct the command.

The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name

TIMEOUT

(Optional) Specifies the maximum amount of time to wait for a channel to become available (in 1/10-second units). Or, you can specify 0 (zero) to return immediately if a channel is not available or -1 to wait indefinitely.

If you specify a TIMEOUT value, CA Automation Point queues and services requests for an unavailable channel in the order you requested.

Default: 0

PREFIX

(Optional) Specifies the name of the REXX stem variable (other than the default name) that contains the return information for the command.

For information about changing the default variable name, see ADDRESS VOX Return Information in this chapter.

Default: VOX.GETCHANNEL

CMDRESP

(Optional) Directs return information to a specific *destination*. For a list of valid *destination* values, see ADDRESS VOX Return Information in this chapter.

Default: REXX

Return Information:

After the GETCHANNEL command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. Additionally, the VOX.GETCHANNEL variable contains the returned channel handle.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5105, 5117, 5135, 5155, 5201, 5202, 5206. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Usage Notes:

- You do not need to issue the GETCHANNEL command before issuing the ANSWER or ANSWERPLAY commands (if GROUP is specified).
- The calling REXX program has exclusive access to the channel. Exclusive access remains in effect until one of the following occurs:
  - The RELEASECHANNEL command executes.
  - You reset the channel through the Configuration Manager.
  - You close or exit from the command shell that executed the REXX program.

Examples:

- The following REXX code illustrates how to use the GETCHANNEL command to access a specific voice channel:

```
/* Get exclusive access a specific voice channel. */

address vox 'getchannel channelnum(1) prefix(handle)'
...

/* Perform voice processing using the voice channel. */
...

/* Release the voice channel when processing is completed. */

address vox 'releasechannel channel('handle')'
```

- The following REXX code illustrates how to use the GETCHANNEL command to access the first available voice channel:

```
/* Get exclusive access to the first available voice channel. */

address vox 'getchannel group(all) prefix(handle)'
...

/* Perform voice processing using the voice channel. */
...

/* Release the voice channel when processing is completed. */

address vox 'releasechannel channel('handle')'
```

GETCHANNELNUM Command

The GETCHANNELNUM command identifies the physical voice channel number associated with the specified channel handle.

This command has the following format:

ADDRESS VOX "GETCHANNELNUM CHANNEL(*channelhandle*)
[SYSTEM(*sysname*)]
[PREFIX(*varname*)]
[CMDRESP(*destination*)]"

CHANNEL

Specifies the channel handle (*channelhandle*)-identifying some physical channel-that the GETCHANNEL command returns.

SYSTEM

(Optional) Specifies the name of the system to which you want to direct the command.

The *sysname* value can contain up to eight alphanumeric characters.

Note: A *system* is a workstation on which a notification server resides. If CA Automation Point is running within a distributed system, (that is, if all CA Automation Point components are not running on a single workstation), you must specify the SYSTEM operand.

Default: The local system name.

PREFIX

(Optional) Specifies the name of the REXX stem variable (other than the default name) that contains the return information for the command.

For information about changing the default variable name, see ADDRESS VOX Return Information in this chapter.

Default: VOX.GETCHANNELNUM

CMDRESP

(Optional) Directs return information to a specific destination. For a list of valid *destination* values, see ADDRESS VOX Return Information in this chapter.

Default: REXX

Usage Note:

Typically, you do not need to issue the GETCHANNELNUM command except in the following situations:

- When you need to reset it
- When you have posted a callback number
- When you need to get a particular channel

Return Information:

After the GETCHANNELNUM command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. Additionally, the VOX.GETCHANNELNUM variable contains the returned channel handle.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5117, 5155, 5199, 5204. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Example:

The following REXX code illustrates how to use the GETCHANNELNUM command to determine the physical line number associated with the current channel handle, and then use that information in a REXX program:

```
/* Get exclusive access to the first available voice channel. */

address vox 'getchannel group(all) prefix(handle)'

/* Get the physical line number associated */
/* with the channel handle.              */

address vox 'getchannelnum channel('handle') prefix(channelnum)'

/* Get the telephone number of the line connected to the channel.*/

select
  when channelnum == 1 then phonenum='555-1000'
  when channelnum == 2 then phonenum='555-1001'
  when channelnum == 3 then phonenum='555-1002'
  when channelnum == 4 then phonenum='555-1003'
end



/* Page the "on-call" operator */

address vox 'call channel('handle') tonestring('pagernum')'

/* Indicate the telephone number that the operator should call */

address vox 'sendtones channel('handle') tonestring('phonenum')'

/* Wait up to 5 minutes for the operator to call back. */

address vox 'answerplay channel('handle') file(problem.vox options.vox) timeout(3000)'
...

/* Release the voice channel. */

address vox 'releasechannel channel('handle')
```

GETDIGITS Command

The GETDIGITS command retrieves tone digits from a voice channel's digit buffer. The digits can represent one of the following data types:

- A menu selection or access code, such as a personal identification number (PIN), that a remote party enters from the telephone keypad in response to a voice prompt.
- ANI (caller identification) digits that the telephone company sends in response to a WINK command.

This command has the following format:

ADDRESS VOX "GETDIGITS CHANNEL(*channelhandle*)
[SYSTEM(*sysname*)]
[COUNT(*numtones*)]
[IDDELAY(*maxdelay*)]
[TERMKEY(*keytone*)]
[PREFIX(*varname*)]
[CMDRESP(*destination*)]
[SINGLESTRING(YES|NO)]"

CHANNEL

Specifies the channel handle (*channelhandle*)-identifying some physical channel-that the GETCHANNEL command returns.

SYSTEM

(Optional) Specifies the name of the system running the notification server to which you want to direct the command.

The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name

COUNT

(Optional) Specifies the maximum number of tones to retrieve. Valid values are 0 (zero) through 31.

Note: If you specify COUNT(0), the GETDIGITS command returns only the number of digits in the digit buffer (but no digits).

Default: 1

IDDELAY

(Optional) Specifies the maximum time delay allowed between each digit's retrieval (in 1/10-second units).

Default: 30

TERMKEY

(Optional) A termination tone digit that the remote party enters on the telephone keypad.

If CA Automation Point detects a tone that you specify, it terminates the GETDIGITS operation. You can specify one or more termination tones.

Valid values are: 0 1 2 3 4 5 6 7 8 9 * # a b c d

Notes:

- The pound key (#) is a common tone-string terminator.
- A special keypad is necessary to send a, b, c, or d from a phone.

Default: The pound key (#)

PREFIX

(Optional) Specifies the name of the REXX stem variable (other than the default name) that contains the return information for the command.

For information about changing the default variable name, see ADDRESS VOX Return Information in this chapter.

Default: VOX.GETDIGITS

CMDRESP

(Optional) Directs return information to a specific destination. For a list of valid *destination* values, see ADDRESS VOX Return Information in this chapter.

Default: REXX

SINGLESTRING

(Optional) Specifies whether the GETDIGITS command's return string appears as a single string of digits in one variable (rather than as individual digits in separate variables).

Default: SINGLESTRING(YES)

Return Information:

After the GETDIGITS command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. Additionally, if you accept the default SINGLESTRING(YES) operand, the entire string of tone digits appears in the VOX.GETDIGITS (or VOX.GETDIGITS.1) variable.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5106, 5107, 5112, 5117, 5155, 5199, 5204, 5300. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

If you specify the optional SINGLESTRING(NO) operand, the GETDIGITS command returns this information in each of the VOX.GETDIGITS variables (VOX.GETDIGITS.1 through VOX.GETDIGITS.*n*):

Field	Description of Returned Information
1	A single digit entered by the remote party or sent by the telephone company.
2	The tone type of the digit that was retrieved: ■ DTMF-Dual Tone Multi-Frequency ■ AP-Audio Pulse ■ LP-Loop Pulse MF-Multi-Frequency

The value stored in the VOX.GETDIGITS.0 variable represents the number of tone digits that were retrieved.

Example:

The following REXX code illustrates how to use the GETDIGITS command to collect a remote user's personal identification number (PIN) entered from the telephone keypad:

```
/* Get exclusive access to a specific voice channel. */
/* If the channel is in use, wait up to two minutes  */
/* for it to become available.                       */

address vox 'getchannel channelnum(1) timeout(1200) prefix(handle)'
...

/* Prompt the remote party to enter a PIN. */

address vox 'play channel('handle') file(entrpin.vox)'

/* Collect the PIN digits entered. */

address vox 'getdigits channel('handle') count(8) singlestring(yes) prefix(response)'
...

/* Release the voice channel. */

address vox 'releasechannel channel('handle')
```

GETGROUP Command

The GETGROUP command returns the group name that you specify and a list of all physical voice channel numbers associated with the group. The return information also indicates whether the channels in the specified group can be interrupted by other REXX programs.

This command has the following format:

```
ADDRESS VOX "GETGROUP
 [GROUP(groupname)]
 [SYSTEM(sysname)]
 [PREFIX(varname)]
 [CMDRESP(destination)]"
```

The GETGROUP command requires no operands.

Specifying the GETGROUP command with no operands returns information about the ALL group only. To see all defined groups, use GETGROUP GROUP(*).

GROUP

(Optional) Specifies the name of the group from which to retrieve a list of all associated voice channels.

Default: ALL

SYSTEM

(Optional) Specifies the name of the system that is running the notification server to which you want to direct the command.

The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name

PREFIX

(Optional) Specifies the name of the REXX stem variable (other than the default name) that contains the return information for the command.

For information about changing the default variable name, see ADDRESS VOX Return Information in this chapter.

Default: VOX.GETGROUP

CMDRESP

(Optional) Directs return information to a specific destination (*destination*). For a list of valid *destination* values, see ADDRESS VOX Return Information in this chapter.

Default: REXX

Return Information:

After the GETGROUP command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. Additionally, the VOX.GETGROUP variable contains the returned channel listing.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5117, 5155, 5199, 5203. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

The GETGROUP command returns this information in each of the VOX.GETGROUP variables (VOX.GETGROUP.1 through VOX.GETGROUP.*n*):

Field	Description of Returned Information
1	The group name.
2	Whether the channels in the group are interruptible by other REXX programs (YES or NO) .
3	The number of channels associated with the group.
4	The number of the first channel in the group.
5	The number of the second channel in the group.
n	The number of the *n*th channel in the group.

The value stored in the VOX.GETGROUP.0 variable represents the number of groups about which the GETGROUP command returned information.

Example:

The following REXX code illustrates how to use the GETGROUP command to retrieve the information for the group ALL, and then use that information to create a new channel group:

```
/* Get a list of the local notification server's available channels. */

address vox 'getgroup group(all)'

if RC == 0 then
do
  parse var vox.getgroup.1 name interrupt numchannels channelsingroup

  /* Define the new group to contain all available channels. */

  newgroup.0 = 1
  newgroup.1 = "NEWGROUP" || " " || interrupt || " " || "set " || numchannels || " " || channelsingroup

  /* Create the new group on the local notification server workstation.*/

  address vox 'setgroup var(newgroup.)'
  ...

end
```

GETSTATUS Command

The GETSTATUS command returns the current status of a voice channel.

This command has the following format:

```
ADDRESS VOX "GETSTATUS CHANNELNUM(channelnum)
 [SYSTEM(sysname)]
 [PREFIX(varname)]
 [CMDRESP(destination)]"
```

CHANNELNUM

A specific, physical channel number. Valid values range from 1 through the number of lines installed.

SYSTEM

(Optional) Specifies the name of the system that is running the notification server to which you want to direct the command.

The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name

PREFIX

(Optional) Specifies the name of the REXX stem variable (other than the default name) that contains the return information for the command.

For information about changing the default variable name, see ADDRESS VOX Return Information in this chapter.

Default: VOX.GETSTATUS

CMDRESP

(Optional) Directs return information to a specific destination (*destination*). For a list of valid *destination* values, see ADDRESS VOX Return Information in this chapter.

Default: REXX

Return Information:

After the GETSTATUS command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. Additionally, The GETSTATUS command returns one of the following status information values in the VOX.GETSTATUS variable:

Channel Status	Meaning
BUSY	Busy
DIAL	Dialing a telephone number
GETDIG	Getting digits from the voice channel's digit buffer
IDLE	Idle (no I/O activity on the channel)
NOTINUSE	Channel not in use
PLAY	Playing a voice message
RECD	Recording a voice message
HOOK	Setting the hook state to either on-hook or off-hook
STOPD	The current operation is stopped, but the channel is not idle
WTEVT	Waiting for a specified event to occur
WTRNG	Waiting for rings

- **If the command does not execute successfully**, the RC variable contains one of the following values: 5155, 5199, 5201. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code, and complete error message text.

Example:

The following REXX code illustrates how to use the GETSTATUS command to determine the number of channels that are currently in use on a notification server:

```
/* Get a list of the local notification server's available channels. */

address vox 'getgroup group(all)'
parse var vox.getgroup.1 name interrupt numchannels
channelsingroup
channelsinuse = 0

/* Check every channel to determine in-use status of each. */

do i = 1 to numchannels

/* Get the next channel number in the group */

parse var channelsingroup channelnumber channelsingroup

/* Check channel. */

address vox 'getstatus channelnum('channelnumber') prefix(status)'

/* Maintain a count of the number of in-use channels. */

if status <> "NOTINUSE" then channelsinuse = channelsinuse + 1
end
```

GETSYSNAMES Command

The GETSYSNAMES command retrieves the system name of the local workstation and the system names of all connected notification server workstations, if any.

This command has the following format:

```
ADDRESS VOX "GETSYSNAMES [PREFIX(varname)]
 [CMDRESP(destination)]"
```

PREFIX

(Optional) Specifies the name of the REXX stem variable (other than the default name) that contains the return information for the command.

For information about changing the default variable name, see ADDRESS VOX Return Information in this chapter.

Default: VOX.GETSYSNAMES

CMDRESP

(Optional) Directs return information to a specific destination (*destination*). For a list of valid destination values, see ADDRESS VOX Return Information in this chapter.

Default: REXX

Return Information:

After the GETSYSNAMES command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. Additionally, the GETSYSNAMES command returns this information in each of the VOX.GETSYSNAMES variables (VOX.GETSYSNAMES.1 through VOX.GETSYSNAMES.*n*):

Field	Description of Returned Information
1	System name
2	System type, either local (LOCAL) or connected (CONNECTED)
3	Whether the engine is active (YES or NO)

  The value stored in the VOX.GETSYSNAMES.0 variable represents the total number of system names retrieved.

- **If the command does not execute successfully**, the RC variable contains the following value: 5117. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Example:

The following REXX code illustrates how to use the GETSYSNAMES command to retrieve the system names of all connected notification servers, and then use that information to determine whether each notification server is running and whether the connection to each notification server is active:

```
/* Get the system names of all connected notification servers. */

address vox 'getsysnames prefix(engines)'

/* Verify that the connected notification server is running and */
/* that the connection to it is still active.          */
/* If you are running the client only, this will fail. */
/* Adjust the REXX accordingly.                        */

do i = 1 to vox.getsysnames.0

  /* Issue a VOX command to the connected notification server */

  address vox 'getgroup system('engines.i')'

  /* Display the result. */

  select
    when RC == 0 then say 'Communication with notification server:
' engines.i 'verified.'
    ...
  end

end
```

LOAD Command

The LOAD command loads a voice file or voice word library index file into your workstation's memory, allowing faster access to your voice data.

This command has the following format:

```
ADDRESS VOX "LOAD FILE(filename)
 [FILETYPE(NONINDEX|filetype)]
 [SYSTEM(sysname)]"
```

FILE

Specifies the name (*filename*) of a voice file or voice word library.

FILETYPE

(Optional) Specifies the type of voice file to load. Valid values are:

NONINDEX

A nonindexed voice file

WORDLIB

A voice word library's index file. Specifying WORDLIB *does not* load the voice word library's digitized speech data into memory.

Note: When specifying FILETYPE (WORDLIB), do not include a file extension on the FILE operand.

Default: NONINDEX

SYSTEM

(Optional) Specifies the name of the system that is running the notification server to which you want to direct the command.

The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name

Return Information:

After the LOAD command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5108, 5109, 5117, 5155, 5180, 5199, 5252. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Usage Note:

CA Automation Point loads the voice word library's index file into memory *the first time your voice application requires its use.* For more information, see the descriptions of the ANSWERPLAY, CALLPLAY, and PLAYGETDIGITS commands.

Example:

The following REXX code illustrates how to use the LOAD command to load an indexed voice file into memory, allowing for faster access within your voice application:

```
/* Load all prompts in the indexed voice file (for use by */
/* the problem-escalation application). */

address vox 'load file(probesc) filetype(wordlib)'

/* Get exclusive access to the first available voice channel. */

address vox 'getchannel group(all) prefix(handle)'

... body of the problem-escalation application code

/* Release the voice channel. */

address vox 'releasechannel channel('handle')'
```

PLAY Command

The PLAY command plays voice data (prerecorded speech) through a specified voice channel. The voice data can exist in one of these formats:

- **A nonindexed file-**To play a nonindexed voice file, specify the file on the FILE operand. If you specify multiple file names, CA Automation Point plays each file in the order listed with no pauses or clicks inserted between the files.
- **A voice word library-**To play words from a voice word library, specify the name of the library on the FILE operand and the text of the word to play in a REXX variable (or REXX stem variables), and then code the REXX variable's name on the VAR operand.

This command has the following formats:

Use the following format to play a message from a nonindexed voice file:

```
ADDRESS VOX "PLAY CHANNEL(channelhandle)
 FILE(filename_1 [...filename_n])
 FILETYPE(NONINDEX)
 [SYSTEM(sysname)]
 [INTERRUPT(YES|NO)]"
```

Use the following format to play a message from a voice word library file:

```
ADDRESS VOX "PLAY CHANNEL(channelhandle)
 FILE(filename)
 FILETYPE(WORDLIB)
 VAR(varname)
 [SYSTEM(sysname)]
 [INTERRUPT(YES|NO)]"
```

CHANNEL

Specifies the channel handle (*channelhandle*)-identifying some physical channel-that the GETCHANNEL command returns.

FILE

Specifies the name of a voice file or voice word library.

FILETYPE

Specifies the type of voice file to play. Valid values are:

- NONINDEX - A nonindexed voice file
- WORDLIB - A voice word library

Note: When specifying FILETYPE(WORDLIB), do not include a file extension on the FILE operand.

Default: NONINDEX

VAR

This operand is required only when specifying a voice word library on the FILETYPE operand.

For a voice word library, code either a REXX regular variable or stem variable. When coding regular variables with the VAR keyword, only that variable is searched.

With a stem variable, all numeric indexes for this stem will be searched. When coding a stem variable, "*varname.*" should follow this format:

- **varname.**0-The number of words in your message.
- **varname**.1 through **varname.***n*-Each variable contains one word from the voice word library.

SYSTEM

(Optional) Specifies the name of the system that is running the notification server to which you want to direct the command. The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name

INTERRUPT

(Optional) Specifies whether CA Automation Point interrupts a currently playing voice message if it receives a tone digit (that is, if the remote party presses a key on the telephone keypad). Valid values are:

YES

Allows the interruption.

NO

Prevents the interruption.

Default: YES

Return Information:

After the PLAY command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5106, 5109, 5112, 5116, 5117, 5155, 5199, 5204, 5252, 5300. (For more information, see the *Message Reference Guide*.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Usage Notes:

You can use the PLAY command to play voice messages through a voice channel connected to an amplified speaker attached directly to the voice card's analog expansion bus (with pins GND and AUD1-AUD4).

Examples:

- The following REXX code illustrates how to use the PLAY command to play three individual voice messages. Each message is stored in a nonindexed (flat) voice file.

```
/* Get exclusive access to a specific voice channel. */

address vox 'getchannel channelnum(1) prefix(handle)'
...

/* Play messages from nonindexed voice files. */

address vox 'play channel('handle') file(greeting.vox mainmnu1.vox mainmnu2.vox)'
...

/* Release the voice channel. */

address vox 'releasechannel channel('handle')'
```

- The following REXX code illustrates how to use the PLAY command to play voice messages stored in an indexed voice file:

```
/* Get exclusive access to a specific voice channel. */

address vox 'getchannel channelnum(1) prefix(handle)'
...

/* Play messages from an indexed file. */

greeting = 1
mainmenu1 = 2
mainmenu2 = 2
...
messages.0 = 3
messages.1 = greeting
messages.2 = mainmenu1
messages.2 = mainmenu2

address vox 'play channel('handle') file(helpdesk.vap) filetype(index) var(messages.)'
...

/* Release the voice channel. */

address vox 'releasechannel channel('handle')'
```

- The following REXX code illustrates how to use the PLAY command to play voice messages using the voice word library:

```
/* Get exclusive access to a specific voice channel. */

address vox 'getchannel channelnum(1) prefix(handle) '
...

/*Play messages via the VOX Word Library. */

messages.0 = 4
messages.1 = "System"
messages.2 = "IMS"
messages.3 = "Is"
messages.4 = "Down"

address vox 'play channel('handle') file(voxm_a) filetype(wordlib) var(messages.)'
...

/* Release the voice channel. */

address vox 'releasechannel channel('handle')'
```

PLAYGETDIGITS Command

The PLAYGETDIGITS command performs the following operations:

- Plays voice data (prerecorded speech) through a specified voice channel. The voice data can exist in one of these formats:
  - A nonindexed file

    To play nonindexed files, specify the file on the FILE operand. If you specify multiple file names, CA Automation Point plays each file in the order listed. CA Automation Point does not insert pauses between multiple files.
  - A voice word library

    To play words from a voice word library, specify the name of the library on the FILE operand and the text of the word to play on the VAR operand.
- Retrieves tone digits from a voice channel's digit buffer. The digits can be menu selections or access codes that the remote party enters from the telephone keypad in response to a prompt from the voice file.

This command has the following format:

Use the following format to play a message from a non-indexed voice file:

```
ADDRESS VOX "PLAYGETDIGITS CHANNEL(channelhandle)
 FILE(filename_1 [...filename_n])
 FILETYPE(NONINDEX)
 [SYSTEM(sysname)]
 [INTERRUPT(YES|NO)]
 [COUNT(numtones)]
 [IDDELAY(maxdelay)]
 [TERMKEY(keytone)]
 [PREFIX(varname)]
 [CMDRESP(destination)]
 [SINGLESTRING(YES|NO)"
```

Use the following format to play a message from a voice word library file:

```
ADDRESS VOX "PLAYGETDIGITS CHANNEL(channelhandle)
 STRING(tonestring)
 FILE(filename)
 FILETYPE(WORDLIB)
 VAR(varname)
 [SYSTEM(sysname)]
 [INTERRUPT(YES|NO)]
 [COUNT(numtones)]
 [IDDELAY(maxdelay)]
 [TERMKEY(keytone)]
 [PREFIX(varname)]
 [CMDRESP(destination)]
 [SINGLESTRING(YES|NO)"
```

CHANNEL

Specifies the channel handle (*channelhandle*)-identifying some physical channel-that the GETCHANNEL command returns.

FILE

Specifies the name of a voice file or voice word library.

FILETYPE

Specifies the type of voice file to play. Valid values are:

NONINDEX - A nonindexed voice file

WORDLIB - A voice word library

Note: When specifying FILETYPE (WORDLIB), do not include a file extension on the FILE operand.

Default: NONINDEX

VAR

This operand is required only when specifying a voice word library on the FILETYPE operand.

For a voice word library, code either a REXX regular variable or stem variable. When coding regular variables with the VAR keyword, only that variable is searched.

With a stem variable, all numeric indexes for this stem will be searched. When coding a stem variable, *varname.* should follow this format:

***varname*.0**

Specifies the number of words in your message.

varname*.1 through *varname.n

Each variable contains one word from the voice word library.

SYSTEM

(Optional) Specifies the name of the system that is running the notification server to which you want to direct the command. The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name

INTERRUPT

(Optional) Specifies whether CA Automation Point interrupts a currently playing voice message if it receives a tone digit (that is, if the remote party presses a key on the telephone keypad). Valid values are:

YES

Allows the interruption

NO

Prevents the interruption

Default: YES

COUNT

(Optional) Specifies the maximum number of tones to retrieve.

Default: 1

IDDELAY

(Optional) Specifies the maximum time delay allowed between each digit's retrieval (in 1/10-second units).

Default: 30

TERMKEY

(Optional) A termination tone digit that the remote party enters on the telephone keypad.

If CA Automation Point detects a tone that you specify, it terminates the PLAYGETDIGITS operation. You can specify one or more termination tones.

Valid values are: 0 1 2 3 4 5 6 7 8 9 * # a b c d

Note: The pound key (#) is a common tone-string terminator.

Default: #

PREFIX

(Optional) Specifies the name of the REXX stem variable (other than the default name) that contains the returned digit string.

For information about changing the default variable name, see ADDRESS VOX Return Information in this chapter.

Default: VOX.PLAYGETDIGITS

CMDRESP

(Optional) Directs return information to a specific destination. For a list of valid *destination* values, see ADDRESS VOX Return Information in this chapter.

Default: REXX

SINGLESTRING

(Optional) Specifies whether the PLAYGETDIGITS command's return string appears as a single string of digits in one variable (rather than as individual digits in separate variables).

Default: YES

Return Information:

After the PLAYGETDIGITS command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. Additionally, if you accept the default SINGLESTRING(YES) operand, the entire string of tone digits appears in the VOX.PLAYGETDIGITS (or VOX.PLAYGETDIGITS.1) variable.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5106, 5107, 5109, 5112, 5116, 5117, 5155, 5199, 5204, 5300. (For more information, see the *Message Reference Guide*.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

If you specify the optional SINGLESTRING(NO) operand, the PLAYGETDIGITS command returns this information in each of the VOX.PLAYGETDIGITS variables (VOX.PLAYGETDIGITS.1 through VOX.PLAYGETDIGITS.*n*):

Field	Description of Returned Information
1	A single digit entered by the remote party or sent by the telephone company.
2	The tone type of the digit that was retrieved: ■ **DTMF**-Dual Tone Multi-Frequency ■ **AP**-Audio Pulse ■ **LP**-Loop Pulse ■ **MF**-Multi-Frequency The value stored in the VOX.PLAYGETDIGITS.0 variable represents the number of tone digits that were retrieved.

Example:

The following REXX code illustrates how to use the PLAYGETDIGITS command to play a voice message from a nonindexed (flat) voice file, and then collect the tone digits that the remote party enters from the telephone keypad in response:

```
/* Get exclusive access to a specific voice channel. */

address vox 'getchannel channelnum(1) prefix(handle)'
...

/* Play the Reset menu message and give the user five seconds */
/* to respond. */

address vox 'playgetdigits channel('handle') file(resetmnu.vox) iddelay(5) prefix(response)'
...

/* Release the voice channel. */
address vox 'releasechannel channel('handle')'
```

RECORDFILE Command

The RECORDFILE command allows a remote party to record a message. CA Automation Point stores the voice data (digitized speech) in a non-indexed disk file. A two-second audible tone precedes recording, alerting the remote party that the recording operation is active.

Recording terminates when one of the following events occurs:

- CA Automation Point receives a tone digit (that is, the remote user presses a key on the telephone keypad)
- The maximum period of silence has elapsed (specified on the SILENCE operand)
- The maximum recording time has expired (specified on the RECORD operand)

This command has the following format:

```
ADDRESS VOX "RECORDFILE CHANNEL(channelhandle)
 FILE(filename)
 [SYSTEM(sysname)]
 [RECORDTIME(maxrectime)]
 [SILENCE(maxsilence)]
 [OVERWRITE(YES|NO)]
 [INTERRUPT(YES|NO)]"
```

CHANNEL

Specifies the channel handle (*channelhandle*)-identifying some physical channel-that the GETCHANNEL command returns.

FILE

Specifies the name of the nonindexed voice file in which to store your recorded voice data.

SYSTEM

(Optional) Specifies the name of the system that is running the notification server to which you want to direct the command.

The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name

RECORDTIME

(Optional) Specifies the maximum recording time allowed (in 1/10-second units).

Default: RECORDTIME(6000) (five minutes)

SILENCE

(Optional) Specifies the maximum amount of time to allow for silence during the recording operation (in 1/10-second units). The recording operation terminates when the *maxsilence* time expires.

Default: 50 (five seconds)

OVERWRITE

(Optional) Specifies whether a new recorded file (specified on the FILE operand) overwrites a file of the same name, if one exists.

Default: YES

INTERRUPT

(Optional) Specifies whether CA Automation Point interrupts (terminates) the recording operation if it receives a tone digit (that is, if the remote party presses a key on the telephone keypad). Valid values are:

YES

Allows the interruption

NO

Prevents the interruption

Default: YES

Return Information:

After the RECORDFILE command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5103, 5106, 5112, 5113, 5117, 5155, 5199, 5204, 5300. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message ID, which includes the four-digit return code and complete error message text.

Usage Notes:

We recommend that you do *not* use the RECORDFILE command to record voice data for your applications. Doing so causes the following problems, which are common in an analog environment:

- Unwanted leading and trailing silence (usually three to five seconds)
- Background noise (hissing), pops, and cracks
- A leading click at the beginning of the recorded message

Instead, use a voice-editor application to record your voice data.

Example:

The following REXX code illustrates how to use the RECORDFILE command to record a caller's message. The example code then delivers the recorded message to another party.

```
DELIVER_MESSAGE = 1

/* Get exclusive access to a specific voice channel. */

address vox 'getchannel channelnum(1) prefix(handle)'
...
/* Determine whether the caller wants to deliver a message. */

address vox 'playgetdigits channel('handle') file(leavemsg.vox) prefix(selection)'

if selection == DELIVER_MESSAGE then
do
  /* Record the message to deliver. */

address vox 'recordfile channel('handle') file(message.vox)'

  /* Get the four-digit extension of the party to whom the */
  /* caller wants to deliver the message. */

  address vox 'playgetdigits channel('handle') file(leavemsg.vox) count(4) singlestring(yes) prefix(extension)'


  /* Put the caller on hold and deliver the message. */

  address vox 'sendtones channel('handle') tonestring(&)'

  /* Get exclusive access to another specific voice channel. */

  address vox 'getchannel channelnum(2) prefix(handle2)'

  /* Call the specified extension and play the message. */

  address vox 'callplay tonestring('extension') channel('handle2') file(message.vox) prefix(result)'

  /* Release the second voice channel. */

  address vox 'releasechannel channel('handle')'
```

```
  /* Take the caller off hold. Let the caller know */
  /* whether the message was delivered.          */

  address vox 'sendtones channel('handle') tonestring(&)'

  if result == 0 then message = 'MESSAGE DELIVERED'
    else message = 'UNABLE TO DELIVER MESSAGE'
  address vox 'play tonestring('extension') channel('handle2') file(voxm_a) filetype(wordlib) var(message.)'
...
end

/* Release the voice channel. */
address vox 'releasechannel channel('handle')
```

RELEASECHANNEL Command

The RELEASECHANNEL command releases the voice channel identified by the specified channel handle, changing the channel's status from in-use to available and setting the channel's hook-state to on-hook. Use the RELEASECHANNEL command with the GETCHANNEL command to serialize a channel's I/O activity.

This command has the following format:

```
ADDRESS VOX "RELEASECHANNEL CHANNEL(channelhandle)
  [SYSTEM(sysname)]"
```

CHANNEL

Specifies the channel handle (*channelhandle*)-identifying some physical channel-that the GETCHANNEL command returns.

SYSTEM

(Optional) Specifies the name of the system to which you want to direct the command.

The *sysname* value can contain up to eight alphanumeric characters.

Note: A *system* is a workstation on which a notification server resides. If CA Automation Point is running within a distributed system-that is, if all CA Automation Point components are not running on a single workstation-you must specify the SYSTEM operand.

Return Information:

After the RELEASECHANNEL command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5117, 5155, 5199, 5204. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Example:

The following REXX code illustrates how to use the RELEASECHANNEL command to release ownership of a previously acquired voice channel, freeing it for use by other REXX programs:

```
/* Get exclusive access to a specific voice channel. */

address vox 'getchannel channelnum(1) prefix(handle)'

or

/* Get exclusive access to the first available channel in the */
/* OUTBOUND channel group. */

address vox 'getchannel group(outbound) prefix(handle)'

or

/* Get exclusive access to the first available channel in the */
/* INBOUND channel group. */

address vox 'answer group(inbound) prefix(handle)'
...

/* Release the voice channel. */

address vox 'releasechannel channel('handle')
```

SENDTONES Command

The SENDTONES command sends additional tones (telephone keypad digits) to an off-hook voice channel *after* a successful call connection.

This command has the following format:

ADDRESS VOX "SENDTONES CHANNEL(*channelhandle*) TONESTRING(*tonestring*)
[ANALYSIS(YES|NO)]
[SYSTEM(*sysname*)]
[NAME(*CPAparameterset*)]"

For information on valid DTMF dialing characters, see the section Valid Dialing Characters (see page 226).

CHANNEL

Specifies the channel handle (*channelhandle*)-identifying some physical channel-that the GETCHANNEL command returns.

TONESTRING

Specifies the numbers or other special digits (*tonestring*) to dial. Valid characters are: 0 1 2 3 4 5 6 7 8 9 * # a b c d & - , T P M L I X

ANALYSIS

(Optional) Activates call progress analysis for the voice channel.

If you experience a problem when sending a tone string, this option can help you to determine the reason. For example, a loop-drop condition (disconnection) may simply be a one-time occurrence; however, if CA Automation Point reports that it has detected a FAX tone, it is likely that your REXX program has connected with the wrong telephone extension.

Valid values are:

YES

Activates call progress analysis when sending tones (this is necessary for a supervised call-transfer operation).

NO

Does not activate call progress analysis when sending tones (this is necessary for a blind call-transfer operation).

Default: NO, unless the NAME operand is specified, in which case ANALYSIS(YES) is used.

SYSTEM

(Optional) Specifies the name of the system that is running the notification server to which you want to direct the command.

The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name.

NAME

(Optional) Specifies the name of the call progress analysis (CPA) parameter set to use for dialing. CPA set names can be created and configured using the Configuration Manager.

When NAME is specified, the ANALYSIS operand uses the value YES.

Default: EngineDefault

Return Information:

After the SENDTONES command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5102, 5112, 5117, 5155, 5199, 5204, 5300, 5301, 5302, 5303, 5304, 5305, 5306. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Usage Note:

Unlike the CALL command, the SENDTONES command does not set a voice channel's hook state.

Example:

The following REXX code illustrates how to use the SENDTONES command to send the tone digits of a telephone extension to which you want to transfer:

```
mainnumber = '9,555-4000'
extension = '1492'

/* Get exclusive access to a specific voice channel. */

address vox 'getchannel channelnum(1) prefix(handle) system(laengine)'

/* Call your company's main telephone number. */

address vox 'call channel('handle') tonestring('mainnumber') system(laengine)'

/* Have the automated attendant transfer */
/* you to the desired extension.        */

address vox 'sendtones channel('handle') tonestring('extension') analysis(yes)'
...

/* Release the voice channel. */

address vox 'releasechannel channel('handle')'
exit
```

SETGROUP Command

The SETGROUP command associates a group name with one or more voice channels, enabling you to control the available channels more easily. The number of channels that you can associate with a single group is limited only by the number of lines that are installed at your site.

This command has the following format:

ADDRESS VOX "SETGROUP VAR(*varname*) [SYSTEM(*sysname*)]"

VAR

A stem variable name that you assign that contains the group definition information.

Specifies the related variables contain the following information:

***varname*.0**

The number of groups to define.

***varname*.1**

Formatted information for the first group name that you are defining. Each field contains specific information about the first group.

***varname*.2**

Formatted information for the second group name that you are defining, if applicable. Each field contains specific information about the second group.

varname.n

Formatted information for the *n*th group name that you are defining, if applicable. Each field contains specific information about the *n*th group.

SYSTEM

(Optional) Specifies the name of the system that is running the notification server to which you want to direct the command.

The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name.

Information Required to Define a Channel Group

The stem variable *varname* that you specify on the VAR operand contains your group definition information.

The value in the *varname*.0 variable contains the number of groups that you want to define. Variables *varname*.1 through *varname*.*n* each contain a line of formatted information necessary to define one channel group.

The line of information containing a group definition is divided into several fields, as shown:

Field	Description
1	The group name (of up to eight characters) that you want to assign. **Note:** The group name ALL is a reserved name that includes all channels on a given notification server. You cannot change this name.
2	If the ANSWER command is waiting for incoming calls on the group, the value in this field specifies whether another REXX program can interrupt the ANSWER operation and retrieve one of the group's channels. Valid values are: ■ **YES--**Releases a channel in the group if another REXX program requests it. **Note:** When the other REXX program releases the channel, CA Automation Point adds the channel back into the group automatically. ■ **NO--**Does not release a channel in the group if another REXX program requests it.
3	The type of SETGROUP operation that you want to perform. Valid values are: ■ **ADD** - Adds the following channels to the specified group. ■ **SET** - Creates the specified group to contain only the following channels. ■ **REMOVE** - Removes the following channels from the specified group. ■ **PURGE** - Purges (deletes) the specified group and its associated channels.
4	The number of channels that you want this group to contain. Valid values range from 1 through the number of lines installed.
5	The first physical voice channel number to associate with the group.
6	The second physical voice channel number to associate with the group.

Field	Description
n	The *n*th physical voice channel number to associate with the group (specified in field 4).

Return Information:

After the SETGROUP command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5108, 5110, 5114, 5117, 5155, 5199, 5203, 5206. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Example:

Suppose that you want to define one new channel group-for use by your technical support department-with these characteristics:

- The group name is SUPPORT.
- An ANSWER operation on the group is not interruptible by other REXX programs.
- The group contains four channels.
- Channels 03, 04, 05, and 06 are available at your site.

Assume that you have specified the variable MYVARGRP to contain the group definition information. The variable MYVARGRP.0 contains a value of 1 and the group definition in variable MYVARGRP.1 looks like this:

```
SUPPORT NO ADD 4 03 04 05 06
```

The following REXX code illustrates how to use the SETGROUP command to create a new channel group:

```
/* Get a list of the available voice channels */
/* on the New York office's notification server.      */

address vox 'getgroup group(all) system(nyengine)'

if RC == 0 then
do
  parse var vox.getgroup.1 name interrupt numchannels channelsingroup

  /* Define a new channel group to contain */
  /* all available channels.              */

  newgroup.0 = 1
  newgroup.1 = "NEWGROUP" || " " || interrupt || " " || "set " || numchannels || " " || channelsingroup

  /* Create the new group on the New York office's notification server.*/

  address vox 'setgroup var(newgroup.) system(nyengine)'
  ...
end
```

SETHOOK Command

The SETHOOK command explicitly sets a voice channel's hook switch state to either on-hook or off-hook.

In most cases, you need to issue the SETHOOK command only when your REXX program contains code for collecting ANI digits. The telephone company's switching office sends ANI digits between *the ring signals* of an incoming call, requiring the ANSWER (or ANSWERPLAY) command to keep the line in an on-hook state. After the ANI digits have been collected, the SETHOOK command sets the line to an off-hook state so that CA Automation Point can answer the call.

This command has the following format:

```
ADDRESS VOX "SETHOOK CHANNEL(channelhandle)
 HOOKSTATE(ONHOOK|OFFHOOK)
 [SYSTEM(sysname)]"
```

CHANNEL

Specifies the channel handle (*channelhandle*)-identifying some physical channel-that the GETCHANNEL command returns.

HOOKSTATE

Specifies the hook state that you want to set. Valid values are:

ONHOOK

Analogous to hanging up (replacing a telephone handset to end a call).

OFFHOOK

Analogous to picking up a telephone handset to place a call.

SYSTEM

(Optional) Specifies the name of the system that is running the notification server to which you want to direct the command.

The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name.

Return Information:

After the SETHOOK command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5112, 5117, 5155, 5199, 5204. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Example:

The following REXX code illustrates the SETHOOK command. The code segment collects ANI digits (which must occur while the line is still in an on-hook state), and then answers the call and plays a greeting.

```
/* Get exclusive access to a specific voice channel. */

address vox 'getchannel channelnum(1) prefix(handle)'

/* Wait for an incoming call. Leave the line in an on-hook */
/* state so that we can collect ANI digits. */

address vox 'answer channel('handle') hookstate(onhook)'

/* Send a wink signal to central office (CO) switch. */

address vox 'wink channel('handle')'

/* Collect the ANI (or ''caller ID'') digits. */

address vox 'getdigits channel('handle') count(7) singlestring(yes) prefix(phonenum)'
...

/* Answer the incoming call by finally taking the line off-hook. */

address vox 'sethook channel('handle') hookstate(offhook)'
...
/* Play the system greeting message. */

address vox 'play channel('handle') file(greeting.vox)'

/* Release the voice channel. */

address vox 'releasechannel channel('handle')'
```

SETVOLUME Command

The SETVOLUME command adjusts a voice channel's volume level for all subsequent voice message plays (until you change or cancel the volume setting).

This command has the following format:

ADDRESS VOX "SETVOLUME CHANNEL(*channelhandle*) VOLUME(*volumelevel*)
[SYSTEM(*sysname*)]"

CHANNEL

Specifies the specific, physical channel on which to adjust the volume. Valid values range from 1 through the number of lines installed.

VOLUME

Specifies the degree of volume adjustment (in decibels) between (-10) and (+10) at which CA Automation Point plays the voice file on the specified channel.

(The SETVOLUME command does not modify the voice file in any way.)

Specifies the volume adjustment values are absolute, not relative. Specifying VOLUME(0) resets the volume to the default level.

Default: 0

SYSTEM

(Optional) Specifies the name of the system that is running the notification server to which you want to direct the command.

The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name.

Return Information:

After the SETVOLUME command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5102, 5104, 5155, 5199, 5201. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message ID, which includes the four-digit return code and complete error message text.

Example:

The following REXX code illustrates how to use the SETVOLUME command to set a voice channel so that it plays voice messages at a higher volume level than the one at which it was recorded. After the specified voice messages play, the SETVOLUME command executes again to reset the voice channel.

```
/* Get exclusive access to a specific voice channel. */

address vox 'getchannel channelnum(1) prefix(handle)'
...

/* Set the volume adjustment on the associated channel so that */
/* voice files play 10 dB louder than they were recorded. */

address vox 'setvolume channel('handle') volume(10)'

/* Play messages from nonindexed voice files. */

address vox 'play channel('handle') file(atten.vox warning.vox)'
...

/* Reset the volume adjustment on the associated channel so that */
/* voice files play at the volume at which they were recorded.   */

address vox 'setvolume channel('handle') volume(0)'
...

/* Release the voice channel. */

address vox 'releasechannel channel('handle')'
```

STOP Command

The STOP command terminates a currently active I/O operation on a voice channel.

This command has the following format:

```
ADDRESS VOX "STOP CHANNELNUM(channelnumber)
 [SYSTEM(sysname)]"
```

CHANNELNUM

Specifies the channel number (*channelnumber*)-identifying some physical channel-that the GETCHANNELNUM command returns.

SYSTEM

(Optional) Specifies the name of the system that is running the notification server to which you want to direct the command. The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name.

Return Information:

After the STOP command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5117, 5199, 5201. (See the *Message Reference Guide* for more information.) Additionally the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Example:

The following REXX code illustrates how to use the STOP command to stop activity on all voice channels waiting for an incoming call:

```
signal on halt Shutdown
...

Shutdown:

/* Get a list of the local notification server's available channels. */

address vox 'getgroup group(all)'

parse var vox.getgroup.1 name interrupt numchannels channelsingroup

/* Check each channel to determine in-use status. */

do i = 1 to numchannels

   /* Get the next channel number in the group. */

   parse var channelsingroup channelnumber channelsingroup

   /* Determine the channel's in-use status. */

   address vox 'getstatus channelnum('channelnumber') prefix(status)'

  /* Stop all channels that are waiting for an incoming call. */

  if status <> "WTRNG" then
   do
     address vox 'stop channelnum('channelnumber')'
     ...

   end
```

VERIFYUSER Command

The VERIFYUSER command verifies that the remote party's user ID and password are valid.

This command has the following format:

ADDRESS VOX "VERIFYUSER USERID(*userid*) PIN(*pin*) [SYSTEM(*sysname*)]"

USERID

Specifies the remote party's user identification number. The *userid* value must contain between four and eight tone digits (valid values are 0000 through 99999999). Valid characters are: 0 1 2 3 4 5 6 7 8 9

PIN

Specifies the remote party's personal identification number (*pin*) password. The *pin* value must contain between four and eight tone digits (valid values are 0000 through 99999999). Valid characters are: 0 1 2 3 4 5 6 7 8 9

SYSTEM

(Optional) Specifies the name of the system that is running the notification server to which you want to direct the command.

The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name

Return Information:

After the VERIFYUSER command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5111, 5117, 5155, 5199. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Note: When assigning user IDs and PINs, we recommend that you choose tone digits that spell a name or meaningful code on the telephone keypad. For example, if a user ID is SAMJONES, the remote party with that user ID would enter these corresponding tone digits: 72656637. It is usually much easier for a remote user to remember a mnemonic string (such as a name) than it is to remember an arbitrary string of numeric digits.

Example:

The following REXX code illustrates how to use the VERIFYUSER command to verify a caller's user ID and PIN:

```
/* Get exclusive access to a specific voice channel. */

address vox 'getchannel channelnum(1) prefix(handle)'

/* Get the caller's user ID. */

address vox 'playgetdigits channel('handle') file(entruid.vox) count(8) singlestring(yes) prefix(userid)'

/* Get the caller's personal identification number (PIN). */

address vox 'playgetdigits channel('handle') file(entrpin.vox) count(8) singlestring(yes) prefix(pin)'
...

/* Verify that the remote user has entered a */
/* valid user-id/pin combination.           */

address vox 'verifyuser userid('userid') pin('pin')'

if RC == 0 then
do
  /* Processing for valid remote user. */
  ...

end

/* Release the voice channel. */

address vox 'releasechannel('handle')'
```

WINK Command

The WINK command sends a brief handshaking protocol signal through a voice channel.

A common use of the WINK command is to signal your telephone company's switching office to activate its Automatic Number Identification (ANI) service (if your telephone company offers the service in your area and you subscribe to it). The switching office returns the telephone number of the calling party; your REXX program can then collect the digits by issuing the GETDIGITS command.

This command has the following format:

ADDRESS VOX "WINK CHANNEL(*channelhandle*) [SYSTEM(*sysname*)]"

CHANNEL

Specifies the channel handle (*channelhandle*)-identifying some physical channel-that the GETCHANNEL command returns.

SYSTEM

(Optional) Specifies the name of the system that is running the notification server to which you want to direct the command.

The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name

Return Information:

After the WINK command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5112, 5117, 5155, 5199, 5204. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Usage Note:

The switching office usually sends ANI digits between the first and second rings of an incoming call. The protocol required to obtain the ANI digits may vary from that described, depending on your local telephone company's requirements.

Example:

The following REXX code illustrates how to use the WINK command to send a wink signal to the CO switch:

```
/* Get exclusive access to a specific voice channel. */

address vox 'getchannel channelnum(1) prefix(handle)'

...

/* Send a wink signal to the central office (CO) switch. */

address vox 'wink channel('handle')'

...

/* Release the voice channel. */

address vox 'releasechannel('handle')'
```

Utility Commands

The following sections describe CA Automation Point utility commands used with Notification Manager.

GETTAPIDEVICELIST Command

The GETTAPIDEVICELIST command lists all the Telephony Application Programming Interface (TAPI) devices that are installed under Windows on the Notification Server.

This command has the following format:

```
ADDRESS VOX "GETTAPIDEVICELIST
 [PREFIX(varname)]
 [CMDRESP(destination)]
 [SYSTEM(sysname)]"
```

PREFIX

(Optional) Specifies the name of the REXX stem variable (other than the default name) that contains return information for the command.

For information about changing the default variable name, see ADDRESS VOX Return Information in this chapter.

Default: VOX.GETTAPIDEVICELIST

CMDRESP

(Optional) Directs return information, if any, to a specific destination. For a list of valid destination values, see the ANSWER command.

Default: REXX

SYSTEM

(Optional) Specifies the name of the system that is running the notification server to which you want to direct the command.

The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name.

Return Information:

After the GETTAPIDEVICELIST command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully,** the RC variable contains a value of zero. Additionally, information is returned to variables VOX.GETTAPIDEVICELIST.0, VOX.GETTAPIDEVICELIST.1, and so on.

  The information returned contains the device ID (in numeric form) and the modem description installed on that device ID. Each line of information is in the format *n*:d*esc* where *n* is the device ID and *desc* is the modem description. The value stored in the VOX.GETTAPIDEVICELIST.0 variable represents the total number of TAPI devices.

- **If the command does not execute successfully,** the RC variable contains a non-zero value. Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

  For example:

  0:U.S. Robotics 56K FAX EXT
  1:Motorola V.3225

  Note: You can write a REXX program to retrieve and parse this returned information. For coding techniques on how to do this, see the sample REXX program fragment under the PAGE command description.

  The numeric value returned in the VOX.GETTAPIDEVICELIST.0 variable represents the number of lines of data retrieved. Each line of information is in the format described above, and one line is returned per modem installed on the notification server workstation.

Example:

The following example returns all the devices in a stem variable and issues a SAY statement:

```
/* Display available TAPI device on AP Notification server using GETTAPIDEVICELIST */
address VOX "GETTAPIDEVICELIST"
if rc = 0 then
do
 if datatype (VOX.GETTAPIDEVICELIST.0) = "NUM" then
 do
  do i = 1 to VOX.GETTAPIDEVICELIST.0
   say VOX.GETTAPIDEVICELIST.i

  end
 end
end
else
do
  say vox.error
end

return
```

SETENGINE Command

The SETENGINE command allows you to modify various settings that are directly associated with the notification server.

This command has the following format:

ADDRESS VOX "SETENGINE ENGINESETTING(AUTORESET) VAR(var.) [SYSTEM(*sysname*)]"

ENGINESETTING

Specifies the ENGINESETTING operand enables you to change the default autoreset period to a setting other than five minutes.

VAR

A stem variable name (*var.*) that you assign containing the autoreset value. The specified stem variable must contain the following information:

***varname*.0**

Must be 1

***varname*.1**

The value you wish to set for the ENGINESETTING value

SYSTEM

(Optional) Specifies the name of the system to which you want to direct the command.

The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name

Return Information:

After the SETENGINE command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5117, 5155, 5199. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Example:

The following REXX code illustrates how to use the SETENGINE command to set the autoreset to three minutes:

```
value.0 = 1
value.1 = 3  /* value of 3 minutes for auto-reset */
  address VOX "SETENGINE ENGINESETTING(AUTORESET)     VAR(VALUE.)"
```

SETMSGSTREAM Command

The SETMSGSTREAM command serves as a connection between the notification server and the rest of CA Automation Point. The SETMSGSTREAM command duplicates the message stream from the notification server to a queue that is accessible using PPQs, local or remote. This queue can then be read and will act like a source of messages to the other CA Automation Point tools.

This command has the following format:

```
ADDRESS VOX "SETMSGSTREAM QUEUE(AP_SERVER)
 [SEVERITY(ERROR|WARNING|INFORMATIONAL)]"
```

QUEUE

Specifies the name of the queue you want to use to communicate with another application.

SEVERITY

(Optional) This operand allows you to analyze the errors in the notification server and determine the severity of each error. Values are:

ERROR

Duplicates only error messages

WARNING

Duplicates warning and error messages

INFORMATIONAL

Duplicates all messages

Default: INFORMATIONAL

Return Information:

After the SETMSGSTREAM command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5102, 5117, 5155, 5199, 5201. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

SETTRACE Command

The SETTRACE command starts and stops trace logging. The logged trace information is useful for debugging your REXX programs or to help Technical Support solve your problem.

You can start or stop trace logging for an individual voice channel. Logged messages include:

- The CA Automation Point own error, warning, and informational system messages
- Trace-specific messages

Logged trace messages appear in the ASOTRACE.LOG file.

This command has the following format:

ADDRESS VOX "SETTRACE CHANNELNUM(*channelnum*) [STATE(ON|OFF)] [SYSTEM(*sysname*)]"

CHANNELNUM

A specific, physical channel number on which to perform problem tracing. Valid values range from 1 through the number of lines installed.

You can specify the CHANNELNUM operand only once in a single SETTRACE command statement. If you want to set problem tracing for more than one channel, issue the SETTRACE command for each channel that you want to trace.

STATE

(Optional) Activates (ON) or deactivates (OFF) trace logging.

SYSTEM

(Optional) Specifies the name of the system that is running the notification server to which you want to direct the command.

The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name

Return Information:

After the SETTRACE command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5102, 5117, 5155, 5199, 5201. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Example:

The following REXX code illustrates how to use the SETTRACE command to activate trace logging on a specified voice channel:

```
/* Get exclusive access to a specific voice channel. */

address vox 'getchannel channelnum(1) prefix(handle)'

/* Activate tracing for on the channel. */
address vox 'settrace channelnum(1) state(on)'

...Code containing VOX commands to trace on the specified channel

/* Deactivate tracing for the specific channel. */

address vox 'settrace channelnum(1) state(off)'

/* Release the voice channel. */
address vox 'releasechannel channel('handle')'
```

SLEEP Command

The SLEEP command causes the issuing REXX EXEC to enter a system sleep state for the given amount of time.

This command has the following format:

ADDRESS VOX "SLEEP SECONDS(*seconds*) MILLISECONDS(*milliseconds*)"

SECONDS

Specifies the number of seconds to sleep.

MILLISECONDS

Specifies the number of milliseconds to sleep.

Return Information:

After the SLEEP command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5117, 5155, 5199. (See the *Message Reference Guide* for further information.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Example:

The following example of the SLEEP command gets a channel, calls a number, and-when the connection is made-sleeps for two seconds:

```
address vox 'getchannel channelnum(1) prefix(handle)'
address vox 'call channel ('handle')tonestring('phonenumber')'
if vox.call.1 = 'PAMD' then do

/* Because some automated attendants have a long salutation */
/* wait for 2 seconds before sending any tones.          */

address vox 'sleep seconds(2)'

/* Have the automated attendant transfer */
/* your call to the desired extension.   */

address vox 'sendtones channel('handle') tonestring('extension')'

end
...

address vox 'releasechannel channel('handle')'
```

STARTREXX Command

The STARTREXX command executes another REXX program.

Note: You can also use this command to start **any** executable program on a local system.

This command has the following format:

ADDRESS VOX "STARTREXX PROGRAM(*progname*[*arguments*])"

PROGRAM

Specifies the name of the REXX program that you want to execute and the arguments to pass to the program, if any.

Return Information:

After the STARTREXX command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero.
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5117, 5307. (See the *Message Guide*Message Reference Guideinformation.) Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

Example:

The following REXX code illustrates how to use the STARTREXX command in a program that processes incoming calls destined for the help desk. When CA Automation Point receives a call, the HDCTRL.CMD program starts a program named HELPDESK.cmd to service the call automatically.

```
/* HDCTRL.CMD */

signal on halt Name DoExit

do forever
  /* Wait forever for an incoming call and any channel within */
  /* the HELPDESK group. */

address vox answer group(helpdesk) hookstate(onhook) prefix(handle)'

if RC == 0 then
  do
    /* Call answered. Start the HELPDESK.CMD */
    /* program to service the call.         */

    address vox 'startrexx program(helpdesk.cmd 'handle')'

  end

end

DoExit:

/*Exit from the command shell so that CA Automation Point can "cleanup."*/

'@exit'
end
```

VER Command

The VER command returns the version number and configuration information of the local VOX command environment.

This command has the following format:

ADDRESS VOX "VER [CMDRESP(*destination*)] [PREFIX(*varname*)]"

PREFIX

(Optional) Specifies the name of the REXX stem variable (other than the default name) that contains the return information for the command.

For information about changing the default variable name, see ADDRESS VOX Return Information in this chapter.

Default: VOX.VER

CMDRESP

(Optional) Directs return information to a specific destination (*destination*). For a list of valid *destination* values, see ADDRESS VOX Return Information in this chapter.

Default: REXX

Return Information:

After the VER command executes, it sets the special REXX return code variable RC.

- **If the command executes successfully**, the RC variable contains a value of zero. Additionally, the VOX.VER variable contains the return information for the command.
- **If the command does not execute successfully**, the RC variable contains the following value: 5117. Additionally, the VOX.ERROR variable contains the complete VOX error message, which includes the four-digit return code and complete error message text.

The VER command returns this information in the VOX.VER variable:

Field	Description of Returned Information
1	The product name (for example, CA Automation Point)
2	The version number (for example, 11.3.0.0 Rev=99999)
3	The build date of the current notification service in *mmm dd yyyy* format (for example, Jul 5 2010)
4	The system name for the notification service as defined during configuration (for example, PVINFORM)

The following is an example of a return information line:

```
CA Automation Point 11.3.0.0 Rev=99999 Jul 5 2010 APSVR01
```

Chapter 8: ADDRESS TNG Commands

This section describes the commands that you can issue through the CA Automation Point ADDRESS TNG environment. For details about this environment, see the chapter on interacting with external event systems in the *Administrator Guide.*

ADDRESS TNG Command Summary

The following sections list the CA Automation Point ADDRESS TNG commands by category.

ADDRESS TNG Environment Commands

Use the following commands for accessing the CA NSM Common Object Repository.

CREATE

Creates a new object.

DELETE

Deletes a specified object.

GET

Reads one or more PROPERTY/VALUE pairs from a specified object.

LIST

Lists all objects from a specified class.

SET

Writes one or more PROPERTY/VALUE pairs or the contents of a stem variable construct to a specified object.

ADDRESS TNG Event Management Commands

Use the following commands for CA NSM Event Management.

SNMPTRAP

Sends an SNMP trap to the specified host.

UNICMD

Tells the CA NSM Event Manager component, which resides on the specified host, to execute the supplied command.

UNIWTO

Sends the supplied message to the CA NSM Event Manager component on the specified host.

UNIWTOR

Sends the supplied message to the CA NSM Event Manager component on the specified host and receives a reply.

ADDRESS TNG Utility Command

The following is the utility command for the ADDRESS TNG environment.

VER

Returns information about the version of the ADDRESS TNG environment that is running.

ADDRESS TNG Command Syntax

Issue a command from within a REXX program by specifying an ADDRESS TNG statement, as shown:

```
ADDRESS TNG 'tngcommand operand(s)'
```

Follow these guidelines when issuing a command:

- When specifying a required or optional operand with a command, use parentheses to pass values. For example:

  PROPERTY(*propertyname*)

- Do not name your variables with names reserved for ADDRESS TNG commands when creating REXX programs that issue TNG commands.
- Do not include the following characters when creating REXX programs that issue TNG commands,: ()'"<>|. REXX interprets them as REXX delimiters.
- Uppercase or lowercase characters are valid. For example, the following are both valid:

  HOST(*hostname*) and HOST(*HOSTNAME*)

- Leading and trailing blanks are ignored. For example, the following are both valid:

  HOST(*hostname*) and HOST (*hostname*)

- Single and double quotes are valid. For example, the following are all valid:

  COMMAND(*commandstring*) and COMMAND('*commandstring*') and COMMAND("*commandstring*")

- Operands shown in brackets ([ ]) are optional.

Command Requirements When Using the WorldView Component

The following requirements apply when issuing commands to access the WorldView component.

Required Properties

When creating objects, you must specify all required fields in either the PROPERTY/VALUE parameter pair or in the stem variable passed through the VAR parameter. The list of required fields for a given object type can be found by using the CA NSM Class Browser to display the definition of the class of object you are trying to create.

The only exceptions to the requirement to specify required fields concern the UUID and LABEL fields found on all object classes. The ADDRESS TNG environment automatically obtains a UUID for you as part of the object created. If you do not specify a value for the LABEL property, the ADDRESS TNG environment uses the same value as the object NAME.

Dot Notation for Objects

WorldView commands require the OBJECT parameter. The argument to the OBJECT parameter consists of the following two parts:

- The name of the class to which the target object belongs
- The name of the object

These two parts *must* be separated by a period.

ADDRESS TNG Return Information

This section discusses the data returned by ADDRESS TNG commands.

The RC Variable

RC is the REXX variable that contains the return codes from the ADDRESS TNG environment. RC is set by every command and should be programmatically checked for acceptable results (usually a zero value) after each command executes.

- **If the command executes successfully**, the RC variable contains a value of 0 (zero).
- **If the command does not execute successfully**, the RC variable contains one of the following values: 5905, 5908, 5910, 5913, 5920, 5925, 5929, 5930, 5931, 5932, 5935, 5940, 5943, 5961, 5962, 5965, 5966, 5980, 5981, 5982, 5983, 5984, 5985, 5986, 5987, 5988, 5989, 5990, 5991, 5992, 5993, 5994, 5999. (See the *Message Reference Guide* for further information.)

The TNG.ERROR Variable

If an ADDRESS TNG command does not execute successfully (that is, if it returns a non-zero RC value), it generates an error message and stores the message in the special REXX variable called TNG.ERROR. The error message ID begins with the prefix TNG, followed by a four-digit number corresponding to the RC return code value, and a letter indicating the message type. See the *Message Reference Guide* for a more detailed description of the error message.

For example, if RC=5928, then the error message contained in the TNG.ERROR variable is

```
TNG05928E Repository login failed
```

Additional Return Information

The following ADDRESS TNG commands, if executed successfully, return additional information beyond an RC value:

- LIST
- GET
- UNIWTOR
- VER

Return information is stored in the TNG.*tngcommand* variable. (The *tngcommand* portion of the stem variable represents the name of any of the ADDRESS TNG commands above.)

The TNG.*tngcommand.*0 variable contains the number of lines of information returned (that is, the number of elements in the TNG.*tngcommand* variable). The variables TNG.*tngcommand.*1 through TNG.*tngcommand.n* (where *n* is the value of TNG.*tngcommand.0*) each contain a line of information.

Note: The TNG.*tngcommand* variable contains the same value stored in the TNG.*tngcommand.*1 variable. It is the first line of return information.

Change the Default Variable with PREFIX

You can direct return information to a variable other than the default TNG.*tngcommand* by specifying the PREFIX operand as follows:

PREFIX(*newvarname*)

newvarname

Specifies the name of the variable to replace the default.

ADDRESS TNG Environment Commands

The following sections describe the ADDRESS TNG commands used for managing the CA NSM environment.

CREATE Command

The CREATE command creates a new object.

This command has the following format:

ADDRESS TNG 'CREATE OBJECT(*class.object*) PROPERTY(*propertyname*)
VALUE(*propertyvalue*)|VAR(*varlist.*)'

OBJECT

WorldView commands require the OBJECT parameter. The argument to the OBJECT parameter consists of the following two parts:

class

The 1- to 31- character name of the CA NSM class to which the created object belongs.

object

The 1- to 31-character name of the object. The object name can contain alphanumeric characters and any of these special characters: ! @ # $ _

Periods (.) should *not* be used in object names.

After you create a new object, you also usually create an Inclusion object to include your new object under another object on the WorldView 2D map. The OBJECT parameter that you specify for a new inclusion object is

Inclusion. <*Class of the child object*> . <*name of the child object*>

To specify the UUIDs for the new object and its parent object as attributes to the new inclusion object, use the VAR parameter .

PROPERTY

Specifies the name of an object property as listed in the CA NSM class browser.

VALUE

Specifies the value of an object property as listed in the CA NSM class browser. The ADDRESS TNG environment automatically performs the conversion from the REXX variable format to the internal format of CA NSM.

VAR

Specifies the name of a REXX stem variable (*varlist*) that contains PROPERTY/VALUE pairs of data. For example, if the command contains the following:

... VAR(pairlist.)

The ADDRESS TNG environment examines the value of the variable pairlist.0 to determine the number of entries to process. Each of the variables pairlist.1 to pairlist.*n* (where *n* is the value of pairlist.0) should contain a quoted string consisting of a property followed by at least one blank space followed by the value to be set for the property. For example:

```
pairlist.1 = "name  FRED"
pairlist.2 = "label South Dakota"
```

DELETE Command

The DELETE command deletes the specified object.

This command has the following format:

ADDRESS TNG 'DELETE OBJECT(*class.object*) [VAR(*varlist.*)]'

OBJECT

WorldView commands require the OBJECT parameter. The argument to the OBJECT parameter consists of the following two parts:

class

The 1- to 31- character name of the CA NSM class to which the created object belongs.

object

The 1- to 31-character name of the object. The object name can contain alphanumeric characters and any of these special characters: ! @ # $ _

Periods (.) should not be used in object names.

After you delete a new object, you also usually create an Inclusion object to include your new object under another object on the WorldView 2D map. The OBJECT parameter that you specify for a new inclusion object is

Inclusion. *<Class of the child object>* . *<name of the child object>*

Because an pbject may be included under multiple parents, you also typically use the VAR parameter to specify the UUIDs for the child object and its parent object as attributes of the inclusion object to be deleted.

VAR

(Optional) Specifies the name of a REXX stem variable (*varlist*) that contains PROPERTY/VALUE pairs of data to limit the matching objects. For example, suppose the command contains the following:

... VAR(pairlist.)

The ADDRESS TNG environment examines the value of the variable pairlist.0 to determine the number of entries to process. Each of the variables pairlist.1 to pairlist.*value of* pairlist.0 should contain a quoted string consisting of a property followed by at least one blank space followed by the value to be set for the property. For example:

pairlist.1 = "name FRED"

pairlist.2 = "label South Dakota"

GET Command

The GET command reads one or more PROPERTY names from a specified object. The PROPERTY parameter retrieves the value of the requested property. The VAR parameter retrieves the value of the properties contained in the stem variable passed as the argument to the VAR parameter.

This command has the following format:

ADDRESS TNG 'GET OBJECT(*class.object*) PROPERTY(*propertyname*)|VAR(*varlist*)
[PREFIX(*newvarname*)]'

OBJECT

WorldView commands require the OBJECT parameter. The argument to the OBJECT parameter consists of the following two parts:

class

The 1- to 31- character name of the CA NSM class to which the created object belongs.

object

The 1- to 31-character name of the object. The object name can contain alphanumeric characters and any of these special characters: ! @ # $ _

Periods (.) should not be used in object names.

PROPERTY

Specifies the name of an object property (*propertyname*) as listed in the CA NSM class browser.

VAR

Specifies the name of a REXX stem variable (*varlist*) that contains the PROPERTY names of an object. For example, suppose the command contains the following:

... VAR(propname.)

The ADDRESS TNG environment will examine the value of the variable propname.0 to determine the number of entries to process. Each of the variables propname.*1* to *propname.n,* where *n* is the value of propname.0, should contain a quoted string consisting of a property name. For example:

propname.1 = "name"
propname.2 = "label"

PREFIX

(Optional) Directs return information to a stem variable name (*newvarname*) other than the default. For more information, see ADDRESS TNG Return Information in this chapter.

Note: The PREFIX operand is only valid if REXX is the destination of the return information.

Default: TNG.GET

LIST Command

The LIST command reads one or more objects from a specified class. Using the LIST command with only a class in the OBJECT argument returns a list of all objects in the class in stem variable form. The returned data includes the name and UUID properties for each object found. These take the form TNG.LIST.x = "*uuid name*" or *prefix.x* = "*uuid name*".

This command has the following format:

ADDRESS TNG 'LIST OBJECT(*classname*.) [PREFIX(*newvarname*)]'

OBJECT

WorldView commands require the OBJECT parameter. The argument to the OBJECT parameter for the LIST command is the one- to 31- character name of the *CA NSM class* (a period after the class name is required).

PREFIX

(Optional) Directs return information to a stem variable name (*newvarname)* other than the default. For more information, see ADDRESS TNG Return Information in this chapter.

Note: The PREFIX operand is only valid if REXX is the destination of the return information.

Default: TNG.LIST

SET Command

The SET command writes one or more PROPERTY/VALUE pairs or the contents of a stem variable construct to a specified object.

This command has the following format:

```
ADDRESS TNG 'SET OBJECT(class.object)
 PROPERTY(propertyname)
 VALUE(propertyvalue)|VAR(varlist.)'
```

OBJECT

WorldView commands require the OBJECT parameter. The argument to the OBJECT parameter consists of the following two parts:

class

Specifies the 1- to 31- character name of the CA NSM class to which the created object belongs.

object

Specifies the 1- to 31-character name of the object. The object name can contain alphanumeric characters and any of these special characters: ! @ # $ _

Periods (.) should not be used in object names.

PROPERTY

Specifies the name of an object property (*propertyname*) as listed in the CA NSM class browser.

VALUE

Specifies the value of an object property (*propertyvalue*) as listed in the CA NSM class browser. The ADDRESS TNG environment automatically performs the conversion from REXX variable form to the internal format of CA NSM.

VAR

Specifies the name of a REXX stem variable (*varlist*) that contains PROPERTY/VALUE pairs of data. For example, if the command contains the following:

```
... VAR(pairlist.)
```

The ADDRESS TNG environment examines the value of the variable pairlist.0 to determine the number of entries to process. Each of the variables pairlist.1 to pairlist.*n* (where *n* is the value of pairlist.0) should contain a quoted string consisting of a property followed by at least one blank space followed by the value to be set for the property, for example:

```
pairlist.1 = "name  FRED"
pairlist.2 = "label South Dakota"
```

ADDRESS TNG Event Management Commands

The following sections describe the ADDRESS TNG commands used for CA NSM event management.

SNMPTRAP Command

The SNMPTRAP command sends an SNMP trap to the specified host.

This command has the following format:

ADDRESS TNG "SNMPTRAP HOST({*HostName*|*}) [COMMUNITY(*CommunityName*)]
GTRAP(*GenericTrapNum*)
STRAP(*SpecificTrapNum*)
DATA(*RexxVar*)"

HOST

Specifies the TCP/IP host name (*HostName*) to which the event is to be sent *or* * for all host names specified for message forwarding in the Configuration Manager GUI.

COMMUNITY

(Optional) Specifies the community name (*CommunityName*) under which the trap is to be issued.

Default: public

GTRAP

Specifies the SNMP generic trap type (*GenericTrapNum*), which can be a value of 0 through 6.

STRAP

Specifies the specific trap number (*SpecificTrapNum*) to be used when the enterprise-specific *GenericTrapNum* has a value of 6.

DATA

Specifies the data content of the SNMP trap, which is stored in a REXX variable (*RexxVar*). The REXX variable must contain the following:

rexxvar.0 = *The total number of values supplied* (this number must always be a multiple of 3; six are shown below)
rexxvar.1 = *OID 1*
rexxvar.2 = *type 1*
rexxvar.3 = *value 1*
rexxvar.4 = *OID 2*
rexxvar.5 = *type 2*
rexxvar.6 = *value 2*

OID

Specifies he SNMP object identifier for the trap.

type

Specifies the data type of the value that follows. *type* can be one of the following:

counter	null	opaque
gauge	objectidentifier	opaqueascii
integer	octetstring	timeticks
ipaddress	octetstringascii	

value

Specifies the value to be sent by the trap.

UNICMD Command

The UNICMD command tells the CA NSM Event Manager component, which resides on the specified host, to execute the supplied command.

This command has the following format:

ADDRESS TNG "UNICMD HOST(*HostName*) COMMAND(*CommandString*)"

HOST

Specifies the TCP/IP host name (*HostName*) to which the event is to be sent

COMMAND

Specifies the command (*CommandString*) to be executed by CA NSM

UNIWTO Command

The UNIWTO command sends the supplied message to the CA NSM Event Manager component on the specified host.

This command has the following format:

ADDRESS TNG "UNIWTO HOST(*HostName*|*) MESSAGE(*MessageString*)
[ORIGHOST(*OriginatingHostName*)]
[SEVERITY(E|F|I||S|W)]"

HOST

Specifies the TCP/IP host name (*HostName*) to which the event is to be sent *or* * for all host names specified for message forwarding in the Configuration Manager GUI.

MESSAGE

Specifies the message (*MessageString*) to be sent to CA NSM.

ORIGHOST

(Optional) Specifies the name of the host (*OriginatingHostName*) on which the event originated.

Default: The host name of the CA Automation Point machine.

SEVERITY

(Optional) Specifies the CA NSM severity value. Values are:

E - Error

F - Fatal

I - Information

S - Success

W - Warning

Default: I

UNIWTOR Command

The UNIWTOR command sends the supplied message to the CA NSM Event Manager component on the specified host and receives a reply.

If the RC for the command is 0, the variable TNG.UNIWTOR.1 contains the reply. In addition, the variable TNG.UNIWTOR is set to the same value as TNG.UNIWTOR.1.

This command has the following format:

ADDRESS TNG "UNIWTOR HOST(*HostName*) MESSAGE(*MessageString*)
[ORIGHOST(*OriginatingHostName*)]
[PREFIX (*newvarname*)
[SEVERITY(E|F|I|S|W)]"

HOST

Specifies the TCP/IP host name (*Hostname*) to which the event is to be sent.

MESSAGE

Specifies the message (*MessageString*) to be sent to CA NSM.

ORIGHOST

(Optional) Specifies the name of the host (*OriginatingHostName*) on which the event originated.

Default: The host name of the CA Automation Point machine.

PREFIX

(Optional) Directs return information to a stem variable name (*newvarname*) other than the default. For more information, see ADDRESS TNG Return Information in this chapter.

Note: The PREFIX operand is only valid if REXX is the destination of the return information.

Default: TNG.VER

SEVERITY

(Optional) Specifies the CA NSM severity value. Values are:

E - Error

F - Fatal

I - Information

S - Success

W - Warning

Default: I

ADDRESS TNG Utility Command

The following section describes the ADDRESS TNG utility command.

VER Command

The VER command provides the version number of the ADDRESS TNG environment on your workstation.

The VER command returns one line of information containing three fields. The fields contain the following information:

Field	Description
1	The name of the API (for example, CA NSM REXX API)
2	The version number (for example, 11.3.0.0 Rev=99999)
3	The build date of the current ADDRESS TNG environment in *mmm dd yyyy* format (for example, Jul 5 2010)

The following is an example of a return information line:

```
CA NSM Rexx API 11.3.0.0 Rev=99999 Jul 5 2010
```

This command has the following format:

```
ADDRESS TNG "VER [PREFIX (newvarname)]"
```

PREFIX

(Optional) Directs return information to a stem variable name (*newvarname*) other than the default. For more information, see ADDRESS TNG Return Information in this chapter.

Note: The PREFIX operand is only valid if REXX is the destination of the return information.

Default: TNG.VER

Chapter 9: ADDRESS OPS Commands

This section describes the commands that you can issue through the CA Automation Point ADDRESS OPS environment. For details about this environment, see the *Administrator Guide.* For more information on OPS commands, see the CA OPS/MVS *Command and Function Reference*.

ADDRESS OPS Command Summary

The following list summarizes CA Automation Point ADDRESS OPS commands.

ACTIVATE

Sends a request to the CA Automation Point interface to request activation of the CCI connection to the MSF node specified on the SYSTEM keyword.

DEACTIVATE

Sends a request to the CA Automation Point interface to request deactivation of the CCI connection to the MSF node specified on the SYSTEM keyword.

LIST

Returns a list of MSF nodes that are currently configured on the issuing CA Automation Point workstation.

OPER

Sends a z/OS system command execution request to the CA OPS/MVS host.

OSFTSO

Sends a TSO command execution request to the CA OPS/MVS Operator Server Facility (OSF) TSO component.

VER

Returns information about the version of the ADDRESS OPS command environment that is running.

WTO

Sends a message to the CA OPS/MVS Multi-System Facility (MSF) component.

ADDRESS OPS Command Syntax

Issue an OPS command from within a REXX program by specifying an ADDRESS OPS statement, as shown:

ADDRESS OPS "*opscommand operands(s)*"

Operands shown in brackets ([ ]) are optional.

Follow these guidelines when issuing an OPS command:

- Use parentheses to pass values when specifying a required or optional operand with an OPS command. For example**:**

  SYSTEM(MSF *system name*)

- Do not name your variables with names reserved for OPS commands when creating REXX programs that issue OPS commands.
- Uppercase or lowercase characters are valid. For example, the following are both valid:

  TEXT(*text*) and TEXT(*TEXT*)

- Leading and trailing blanks are ignored. For example, the following are both valid:

  TEXT(*text*) and TEXT (*text*)

- Single and double quotes are valid. For example, the following are all valid:

  COMMAND(*commandstring*) and COMMAND('*commandstring*') and COMMAND("*commandstring*")

- Use two adjacent double quotes ("") to indicate a literal double quote. Use two adjacent single quotes ('') to indicate a literal single quote.
- The usual delimiter for the COMMAND operand in the OSFTSO and OPER command processors is the single quote ('). If the command string you are specifying contains single quotes, use a double quote (") as the operand delimiter, change the REXX delimiter from a double quote (") to a single quote ('), and use two adjacent single quotes ('') to indicate a literal single quote in the text of the command string.

  For example,

  ADDRESS OPS 'OSFTSO SYSTEM(OPS123) COMMAND("SEND "djones has been notified" USER(jsmith)")'

ADDRESS OPS Return Information

This section discusses the data returned by ADDRESS OPS commands.

The RC Variable

RC is the REXX variable that contains the return codes from the ADDRESS OPS environment. RC is set by every command and should be programmatically checked for acceptable results (usually a zero value) after each command executes.

The OPS.ERROR Variable

If an OPS command does not execute successfully (if it returns a non-zero RC value), it generates an error message and stores the message in the special REXX variable called OPS.ERROR. The OPS.ERROR message ID consists of the prefix AXC, followed by a four-digit number (corresponding to the RC return code value), and a letter indicating the message type. See the *Message Reference Guide* for a more detailed description of the error message.

For example, if RC=1122, then the error message contained in the OPS.ERROR variable is:

```
AXC1122E OPS interface; no response
```

Additional Return Information

If executed successfully, the following OPS commands return additional information beyond an RC value:

- LIST
- OPER
- VER

Return information is stored in the special variable OPS.*opscommand.* (The *opscommand* portion of the stem variable represents the name of the OPS command.)

The OPS.*opscommand.*0 variable contains the number of lines of information returned (that is, the number of elements in the OPS.*opscommand* variable). The variables OPS.*opscommand.*1 through OPS.*opscommand.n* each contain a line of information. The *n* value represents the last line of return information.

Note: The OPS.*opscommand* variable contains the same value stored in the OPS.*opscommand.*1 variable. It is the first line of return information.

Change the Default Variable With PREFIX

You can direct return information to a variable other than the default OPS.*opscommand* by specifying the PREFIX operand as follows:

PREFIX(*newvarname*)

where *newvarname* is the name of the variable to replace the default.

Note: The PREFIX operand is valid only if the destination of the return information is REXX.

Example:

The following statement returns the status of the CA OPS/MVS system with an MSF ID of OPS44T, and directs the information to a variable called CURSTATUS:

```
ADDRESS OPS "LIST CMDRESP(REXX) PREFIX(CURSTATUS) SYSTEM(ALL)"
```

Change the Default Return Destination with CMDRESP

The CMDRESP operand shown directs the return information from an OPS command to a specific destination:

CMDRESP(*destination*)

The following are valid values for destination:

REXX

Directs return information to a REXX variable. For more information, see the following section.

Note: The optional PREFIX operand is valid only if the destination of the return information is REXX.

XDQ

Directs return information to the external data queue.

TERMINAL

Directs return information to the terminal. This form uses the same output mechanism used by the REXX SAY command.

NOWHERE

Directs return information to the “bit bucket.” This value discards the return information.

Default: REXX

ADDRESS OPS Command Descriptions

The following sections describe the ADDRESS OPS commands.

ACTIVATE Command

The ACTIVATE command sends a request to the CA Automation Point interface to request activation of the CCI connection to the MSF node specified on the SYSTEM keyword.

This command has the following format:

ADDRESS OPS "ACTIVATE SYSTEM(*MSF system name*)"

SYSTEM

Specifies the CA OPS/MVS node name that is defined in the Selected MSF nodes section of the CA OPS/MVS Event Traffic Configuration dialog. This name corresponds to the name used by the CA OPS/MVS Multi-System Facility (MSF) to identify itself to CAICCI. You can specify only one CA OPS/MVS system per ACTIVATE command.

Usage Notes:

This command is asynchronous in nature. It does not wait for the request to be carried out by the underlying logic of the CA OPS/MVS interface. Thus, a return code (rc) of 0 means only that the command syntax was correct; it does not imply that activation of the specified MSF host was successful. Therefore, this command should only be used for interactive troubleshooting and not be relied upon in automation. Issue an ADDRESS OPS LIST command to display the status of the CCI connection and thereby determine whether the command was successful.

Example:

This example command attempts to activate remote host OPS01P:

```
address OPS "ACTIVATE SYSTEM(OPS01P)"
```

DEACTIVATE Command

The DEACTIVATE command sends a request to the CA Automation Point interface to request deactivation of the CCI connection to the MSF node specified on the SYSTEM keyword.

This command has the following format:

ADDRESS OPS "DEACTIVATE SYSTEM(*MSF system name*)"

SYSTEM

Specifies the CA OPS/MVS node name that is defined in the Selected MSF nodes section of the CA OPS/MVS Event Traffic Configuration dialog. This name corresponds to the name used by the CA OPS/MVS Multi-System Facility (MSF) to identify itself to CAICCI. You can specify only one CA OPS/MVS system per DEACTIVATE command.

Usage Note:

This command is asynchronous in nature. It does not wait for the request to be carried out by the underlying logic of the CA OPS/MVS interface. Thus, a return code (rc) of 0 means only that the command syntax was correct; it does not imply that deactivation of the specified MSF host was successful. Therefore, this command should only be used for interactive troubleshooting and not be relied upon in automation. Issue an ADDRESS OPS LIST command to display the status of the CCI connection and thereby determine whether the command was successful.

Example:

This example command attempts to deactivate remote host OPS01P:

```
address OPS "DEACTIVATE SYSTEM(OPS01P)"
```

LIST Command

The LIST command returns a list of nodes that are currently configured on the issuing CA Automation Point workstation.

Note: This command should be used for diagnostic purposes only.

This command has the following format:

```
ADDRESS OPS "LIST [CMDRESP(destination)] [PREFIX(newvarname)]
  [SYSTEM(MSF system name|ALL)]"
```

CMDRESP

(Optional) Directs return information to a specific destination (*destination*). For a description of valid *destination* values, see ADDRESS OPS Return Information in this chapter.

Default: REXX

PREFIX

(Optional) Directs return information to a stem variable name (*newvarname*) other than the default. For more information, see ADDRESS OPS Return Information in this chapter.

Note: The PREFIX operand is valid only if the destination of the return information is REXX.

Default: OPS.LIST

SYSTEM

(Optional) Specifies the CA OPS/MVS node name that is defined in the Selected MSF nodes section of the CA-OPS/MVS Event Traffic Configuration dialog. This name corresponds to the name used by the CA OPS/MVS Multi-System Facility (MSF) to identify itself to CAICCI. You can specify only one CA OPS/MVS system per LIST command.

Note: If this operand is specified, it acts as a filter and returns data only for the specific CA OPS/MVS host.

Default: ALL

Return Information:

The LIST command returns a line of information for each MSF system specified by the SYSTEM operand. Each line contains three fields of information, as described in the following table.

Field	Description
1	Specifies the name of the remote CA OPS/MVS node name that is defined by the MSF facility.
2	Specifies the status of the CA Automation Point to CA OPS/MVS connection. ■ **INACTIVE**- CA Automation Point is not connected to this CA OPS/MVS node. ■ **ACTIVE**- CA Automation Point is successfully connected to this CA OPS/MVS node. ■ **FAILED**- CA Automation Point failed to make a successful connection to this CA OPS/MVS node. Check the ASOTRACE log for more information about the CAICCI connection errors. **Note:** If you specify a specific host on the SYSTEM operand, only a single line of output is generated for that host.
3	Specifies the ENF/CCI system name used by CAICCI to identify the z/OS host.

For more information, see the section ADDRESS OPS Return Information (see page 324).

OPER Command

The OPER command sends a z/OS system command execution request to the CA OPS/MVS host. This command enables you to execute authorized z/OS system commands on the remote CA OPS/MVS system and receive the responses. For more information, see the CA OPS/MVS for z/OS *Command and Function Reference*.

This command has the following format:

ADDRESS OPS "OPER SYSTEM|SYSID(*MSF system name*) COMMAND('*command text*')
[BMPCMDOUT(OPSLOG|WTO|NONE)]
[CAPTURE(*msgtext list*)]
[CMDECHO(YES|NO)]
[CMDLOG(YES|NO)]
[CMDRESP(*destination*)]
[CMDWAIT(*seconds*)]
[CONNAME|NAME]
[CONTYPE(ANY|EXTCONS|MIGCONS|SSCONS)]
[IMSID(*imsid*)]
[IMSREPLY]
[INTERVAL(*centiseconds*)]
[LOG(YES|NO|OFF)]
[MAXCMDOUT(*number*)]
[MFORM(J|M)]
[NAME|CONNAME(*consolename*)]
[OUTPUT|NOOUTPUT]
[PREFIX(*newvarname*)]
[STOPEND(YES|NO)]
[STOPMSG(*msgtextlist*)]
[STOPRESP(*msgtextlist*)]
[SYSWAIT(*seconds*)]
[WAIT(*seconds*)]"

SYSTEM|SYSID

Sends the command to the specified OPS system by name as it is defined to the Multi-System Facility (MSF) on a CA OPS/MVS host. You can only specify one CA OPS/MVS system per OPER command.

Note: SYSID and SYSTEM are mutually exclusive synonyms that can be used interchangeably; however, you can only specify one on a single OPER command.

COMMAND

Specifies the z/OS command that is to be executed on a CA OPS/MVS OSF TSO server on the designated MSF system. The command can be any valid z/OS command. The command text can be up to 126 bytes long.

If the command contains keywords, you may need to enclose the COMMAND argument in quotation marks (see the example).

BMPCMDOUT

(Optional) Controls the echoing of the current IMS command output.

CAPTURE

(Optional) Specifies one to ten message text segments that can be trapped as responses.

CMDECHO

(Optional) Captures or omits the command echo.

Default: YES

CMDLOG

(Optional) Specifies whether or not to log echo line to SYSLOG.

Default: YES

CMDRESP

(Optional) Directs return information to a specific destination (*destination*). For a description of valid destination values, see ADDRESS OPS Return Information in this chapter.

Default: REXX

CMDWAIT

(Optional) Conditionally waits up to n seconds for all responses on the CA OPS/MVS system where this command runs. The default for CMDWAIT is specified on the Advanced Settings dialog under CA OPS/MVS Interface in Configuration Manager. The initial value in the dialog for CMDWAIT (when the product is installed) is 10 seconds. For more information, see Waittime Before Timeout in this chapter.

CONNAME|NAME

(Optional) Specifies console name to use.

Note: CONNAME and NAME are mutually exclusive synonyms that can be used interchangeably; however, you can only specify one on a single OPER command.

CONTYPE

(Optional) Type of console to use on z/OS. Valid values are: ANY, EXTCONS, MIGCONS or SSCONS.

Note: The CONTYPE keyword is mutually exclusive with the CONNAME|NAME keyword.

IMSID

(Optional) Specifies the IMS ID to send the command to.

IMSREPLY

(Optional) Causes the current IMS command to be issued using the IMS WTOR instead of the BMP**.**

INTERVAL

(Optional) Amount of time CA OPS/MVS waits to see if a further response is forthcoming. Once two output lines have been received, CA OPS/MVS concludes that the response is complete if this interval expires.

LOG

(Optional) Specifies whether or not to log responses to SYSLOG.

Note: Specifying LOG(OFF) is the same as specifying LOG(NO) or CMDLOG(NO).

Default: YES

MAXCMDOUT

(Optional) Maximum of *n* command response lines. The default for MAXCMDOUT is 2000; however, you can change the defaults for all OPER commands by specifying a new default on the Advanced Settings dialog under CA OPS/MVS Interface in Configuration Manager.

MFORM(J|M)

(Optional) Specifies format for response output. For more information, see OPSCMD Command Processor in the CA OPS/MVS for z/OS *Command and Function Reference*.

NOOUTPUT

(Optional) No response lines will be returned.

Note: NOOUTPUT and OUTPUT are mutually exclusive.

OUTPUT

(Optional) Specifies that response lines will be returned. (This is the default.)

PREFIX

(Optional) Directs return information to a stem variable name (*newvarname*) other than the default. For more information, see ADDRESS OPS Return Information in this chapter.

Note: The PREFIX operand is only valid if the destination of the return information is REXX.

Default: OPS.OPER

STOPEND

(Optional) Determines whether the end line of a multiline WTO message stops CA OPS/MVS from collecting further responses.

Default: YES

STOPMSG

(Optional) Lists one to ten message text segments that terminate the collection of command response lines. When CA OPS/MVS detects any of these text segments, it stops collecting responses. You do not have to direct the message segment(s) that you specify to the console receiving the command response.

STOPRESP

(Optional) Lists one to ten message text segments that terminate the collection of command response lines. The message segments that you specify must be directed to the console receiving the command response.

SYSWAIT

(Optional) CA Automation Point adds *n* seconds to its overall wait time to account for network delay times incurred communicating with the target CA OPS/MVS host. The default for SYSWAIT is specified on the Advanced Settings dialog under CA OPS/MVS Interface in Configuration Manager. The initial value in the dialog for SYSWAIT (when the product is installed) is 20 seconds. For more information, see the section Waittime Before Timeout (see page 334).

WAIT

(Optional) Unconditionally waits up to *n* seconds for all responses on the targeted CA OPS/MVS system. For more information, see the section Waittime Before Timeout (see page 334).

Waittime Before Timeout

To determine the total length of time CA Automation Point waits before timing out, the following algorithms are used:

- **If WAIT is specified-**The values of WAIT and SYSWAIT are added together for the total wait time.
- **If CMDWAIT is specified-**The values of CMDWAIT and SYSWAIT are added together for the total wait time.
- **If neither WAIT nor CMDWAIT is specified-**The default values for CMDWAIT and SYSWAIT are added together for the total wait time.

Note: The SYSWAIT default is initially set to 20 seconds through a Configuration Manager option. This is a longer amount of time than the corresponding default value on CA OPS/MVS systems because the CA Automation Point OPER command needs to wait longer than, or at least as long as, it waits on the CA OPS/MVS system; otherwise, you will not receive all the responses from CA OPS/MVS.

REXX Variables Set at Termination

At the termination of an OPS OPER command the following REXX variables will be set:

OPS.OPER.RETCODE

The return code from the z/OS command executed (or attempted). If the command could not be executed, this return code indicates the specific error.

OPS.OPER.TIMEOUT

If CMDWAIT plus SYSWAIT or WAIT plus SYSWAIT were exceeded, the value is set to YES; Otherwise, the value is set to NO.

OPS.OPER.MAXCMDOUT

If MAXCMDOUT number of lines was exceeded, the value is set to YES; otherwise, the value is set to NO.

Note: If the responses all come back before the specified timeout period and below the number specified by MAXCMDOUT, the OPS.OPER.TIMEOUT and OPS.OPER.MAXCMDOUT variables are set to NO.

Return Information:

The OPER command returns each line of output returned by the Z/OS command specified by COMMAND. For more information, see ADDRESS OPS Return Information in this chapter.

For a list of all possible return codes that CA Automation Point returns to REXX callers of address OPS OPER in the REXX variable OPS.OPER.RETCODE, see Return Codes From OPSCMD in the CA OPS/MVS for z/OS *Command and Function Reference*.

Example:

This sample REXX fragment uses OPER to issue the z/OS command D T and display the responses. The D T command is directed to the CA OPS/MVS system with an MSF ID of OPS123:

```
ADDRESS OPS "OPER COMMAND ('D T') SYSTEM(OPS123)"
If rc<> 0 then do
  say "OPS OPER failed; rc=" rc
  return
  end
 do i = 1 to ops.oper.0
  say ops.oper.i
                  end
 say "OPS return code=" ops.oper.retcode
```

Note: In the example above, ops.oper.0 contains the number of lines of response.

OSFTSO Command

The OSFTSO command sends a TSO command execution request to the CA OPS/MVS Operator Server Facility (OSF) TSO component. This command enables you to run any TSO command processor, OPS/REXX program, or TSO/E REXX program under the OSF on the remote CA OPS/MVS system.

This command has the following format:

ADDRESS OPS "OSFTSO SYSTEM(*MSF system name*) COMMAND('*command text*')"

SYSTEM

Specifies the system name as it is defined to the Multi-System Facility (MSF) on a CA OPS/MVS host. You can only specify one CA OPS/MVS system per OSFTSO command.

COMMAND

Specifies the TSO command that is to be executed on a CA OPS/MVS OSF TSO server on the designated MSF system. Though the *command text* can be up to 32,000 bytes in length, it is still subject to length limitations imposed by CA OPS/MVS.

Usage Note:

OPS/REXX programs are specified by indicating the name of the OPS/REXX command processor (OI, OX, OPSIMEX, or OPSEXEC) followed by the REXX program arguments. (See the CA OPS/MVS documentation for details.) If the command contains keywords, you may need to enclose the COMMAND argument in quotes (see the example).

Examples:

- The following statement executes the OPS/REXX program named REXXPGM2, on the CA OPS/MVS system with an MSF ID of OPS123, with an argument of TEST DATA, and a workspace size of 2,000,000 bytes, using the OI command processor:

  ADDRESS OPS "OSFTSO COMMAND ('OI PROGRAM(REXXPGM2) ARG(TEST DATA) WS(2000000)') SYSTEM(OPS123)"

- The following statement executes the OPS/REXX program named REXXPGM3, on the CA OPS/MVS system with an MSF ID of OPS123, using the OX command processor:

  ADDRESS OPS "OSFTSO COMMAND ('OX PROGRAM(""'DSN.NOTIN.SYSEXEC(REXXPGM3)'"")') SYSTEM(OPS123)"

VER Command

The VER command provides the version number of the CA Automation Point interface to CA OPS/MVS on your workstation.

This command has the following format:

ADDRESS OPS 'VER [CMDRESP(*destination*)] [PREFIX(*newvarname*)]'

CMDRESP

(Optional) Directs return information to a specific destination. For a description of valid *destination* values, see ADDRESS OPS Return Information in this chapter.

Default: REXX

PREFIX

(Optional) Directs return information to a stem variable name (*newvarname*) other than the default. For more information, see the section ADDRESS OPS Return Information.

Note: The PREFIX operand is valid only if the destination of the return information is REXX.

Default: OPS.VER

Return Information:

The VER command returns one line of information containing four fields. The fields contain the following information:

Field	Description
1	The name of the API (for example, OPS Rexx C API)
2	The version number (for example, 11.3.0.0 Rev=99999)
3	The Build date of the current CA OPS/MVS service in *mmm dd yyyy* format (for example, Jul 05 2010)

The following is an example of a return information line:

OPS Rexx C API 11.3.0.0 Rev=99999 Jul 05 2010

For more information, see ADDRESS OPS Return Information in this chapter.

WTO Command

The WTO command sends a message to the CA OPS/MVS Multi-System Facility (MSF) component. The message ID and message text are logged to the OPSLOG on the target CA OPS/MVS system. The first word of the message text is used as the message ID. If this word is greater than ten characters, the default message ID AXC1134I will prefix the message text.

This command has the following format:

ADDRESS OPS "WTO SYSTEM(*MSF system name*)[TEXT(*text*)|TEXTVAR(*rexx variable*)]"

SYSTEM

Specifies the remote CA OPS/MVS system to which the WTO message text is to be sent. You can only specify one CA OPS/MVS system per WTO command.

TEXT

Specifies the text of the message, up to 125 bytes.

TEXT and TEXTVAR are mutually exclusive.

TEXTVAR

Specifies the text of the REXX variable, up to 125 bytes.

TEXTVAR and TEXT are mutually exclusive.

Example:

This statement sends a WTO (write-to-operator message) to the CA OPS/MVS system with an MSF ID of OPS123:

```
ADDRESS OPS "WTO TEXT('AXC1134I TEST MESSAGE FROM AP') SYSTEM(OPS123)"
```

Chapter 10: Notification Manager Commands

This section describes the commands that you can issue through the CA Automation Point Notification Manager. For details about this component, see the chapter on using Notification Manager in the *Administrator Guide.*

Notification Manager Command Descriptions

The following sections describe how to use the CA Automation Point Notification Manager commands.

NMANSWER Command

The NMANSWER script is the REXX program that handles the Notification Manager call-in interface. Usually, you should not invoke it unless instructed to do so by Technical Support. The notification server automatically starts NMANSWER when it starts.

This command has the following format:

NMANSWER [DEBUG(YES|NO)] SYSTEM(*sysname*) [MINIMIZE(MIN|")]

DEBUG

(Optional) Indicates whether debug mode should be turned on.

Default: NO

SYSTEM

Specifies the name of the system (*sysname*) running the notification server to which you want to direct the command. The *sysname* value can contain up to eight alphanumeric characters.

MINIMIZE

(Optional) When a copy of NMANSWER is started, a command prompt window is opened. Valid values are:

MIN

The window opens minimized

(blank)

The window opens to the default command prompt window size for the operating system.

Default: blank

NMFIND Command

The NMFIND program is the external interface for invoking Notification Manager to contact people. You tell it:

- Who to contact
- The amount of time to wait between attempts to contact someone
- What to tell the person
- What, if anything, to ask the person
- What action to take when the person answers your question

Optionally, you can tell Notification Manager to use a date or time that is different from the current one when searching the database to determine which methods to use when contacting the specified persons.

This command has the following format:

NMFIND {GROUP(*group*)|KEY(*key*)|NAME(*name*)|PERSON(*person*)} TELL(*tell-text*)
[ASK(*question*
[,*answer1*[::*action1*]
[,*answer2*[::*action2*]
[,...*answer9*[::*action9*]]]])]
[ACKNOWLEDGE(AP|UNI|APUNI[=*host*],*string*)]
[ACKNOWLEDGEAP(*string*)]
[ACKNOWLEDGEOPS(*host*,*string*)]
[ATTACHMENT(*filename*)]
[DATE(*date*)]
[DEBUG(YES|NO)]
[ESCALATIONWAIT(*esc-wait-time*)]
[EMERGENCYINTVL(*e-interval*)]
[EMERGENCYWAIT(*e-wait-time*)]
[FAILUREREXX(*failure-action*)]
[MTUP (A|*profile*)]
[TIME(*time*)]
[USERPARMS(*parameter1*(*value1*)[*parameter2*(*value2*)]]

GROUP

Specifies the name of the group to contact.

KEY

Specifies the key (numeric ID) of the group or person to contact.

The key is shown on the GUI and is returned by the LISTENTITY VOX command.

NAME

Specifies the name of the group or person to contact.

PERSON

Specifies the name of the person to contact.

TELL

Specifies the string or voice file that you want to tell the contacted persons.

ASK

(Optional) Specifies the *question* and *answer1*, *answer2*, and so on arguments can each be either a string or a voice file.

The *question* is played to the person contacted first, and it is followed by each of the answer strings (there may be up to nine). The *question* cannot be omitted. NMFIND prefixes the words "Push 'n' for" to each answer. If you do not specify any answer strings, *answer1* defaults to TRUE and *answer2* defaults to FALSE.

The *action1*, *action2*, and so on arguments are optional. If they are supplied, each one is the name of the REXX program to be run if the associated answer is selected. These REXX program names must contain the associated file extension (.REX or .CMD). The full path to these REXX files may be omitted if the REXX program resides in either the Distrib directory or the Site\MyFiles\REXX directory.

Each REXX *action* program must set a return code on exit. The return code value can be either 0 or any number greater than 9.

ACKNOWLEDGE

(Optional) If specified, this parameter forces an acknowledgement message to be written to the CA NSM Event Console (UNI) or the CA Automation Point Message window (AP), or both (APUNI). If a host name is appended to this first parameter value by an equal sign (=), this CA NSM host receives the acknowledgement string. Otherwise, all CA NSM hosts configured by the Event Traffic Controller receive a copy of this acknowledgement message.

Note: The *host* value is only valid if the UNI or APUNI destination is specified. If an additional string is passed in through this parameter (*string*), this string is added to the acknowledgment text.

ACKNOWLEDGEAP

(Optional) If specified, this operand causes an acknowledge message to be written to the AP MSG window. This operand is the same as ACKNOWLEDGE(AP,*string*).

ACKNOWLEDGEOPS

(Optional) If specified, this operand causes an acknowledge message to be written to the CA-OPS/MVS host. For the acknowledgement message to be sent, the specified host must be currently configured and active.

ATTACHMENT

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

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

DATE

(Optional) Specifies the date, in the form *mm*/*dd* (no year), you want to use when determining how to get in touch with the specified person or group.

Default: The current date

DEBUG

(Optional) Determines whether debugging messages are to be generated. Values are:

YES

Generate debugging messages.

NO

Do not generate debugging messages.

Default: NO

EMERGENCYINTVL

(Optional) If you specify this operand, NMFIND works in emergency mode, submitting all actions to the operating system to be run asynchronously (as separate processes) from NMFIND. This allows you to have NMFIND attempt to contact several people simultaneously about the problem for which it was invoked. The number you specify is the number of seconds NMFIND should wait in between submitting actions.

Note: Use this operand sparingly because running NMFIND in emergency mode consumes a large amount of operating system resources. Use this operand only when you have a situation that requires the quickest response possible. For details, see the section Special Notes About Emergency Mode Processing in this chapter.

If you do *not* specify this operand, Notification Manager waits for each action (for example, voice notification) to succeed or fail before attempting the next action. (The actions are performed synchronously.)

EMERGENCYWAIT

(Optional) Specifies the number of hours to wait for an answer before executing the FAILUREREXX program.

Note: This operand is valid only when NMFIND is in emergency mode (see the description of the EMERGENCYINTVL operand).

Default: 1000 hours

ESCALATIONWAIT

(Optional) Tells NMFIND to wait until all notification methods at given level are exhausted before starting notification escalation. This operand specifies the number of seconds (esc-wait-time) that NMFIND is to wait before proceeding with an escalation. This allows you to give contacts additional time to respond. If there is a response during the wait period, the notification is considered successful and escalation will not take place.

Notes:

- This operand is valid only when the ASK operand is specified.
- In emergency mode, the ESCALATIONWAIT parameter has no effect. For more information, see the description of the EMERGENCYINTVL parameter.

FAILUREREXX

(Optional) Specifies the name of the REXX program to run when every action in the call tree fails or when the value specified for the EMERGENCYWAIT operand expires.

Note: The name of the REXX program passed in this parameter must include the file extension, if present.

MTUP

(Optional) Use the Methods to Use Profile (MTUP) operand to specify which methods are attempted for a particular instance of an NMFIND notification request. The method type code for each method is defined in the Notification Manager database under the TYP parameter for the method. Before attempting to notify a contact using a scheduled method, Notification Manager compares the value its method type code with the profile specified on the MTUP operand. If the method type code is not part of the MTUP profile, notification is not attempted, and the next scheduled method is compared against the MTUP profile. If the method type code is part of the MTUP profile, then the notification is attempted using that method.

The values for this operand are:

profile

Any combination of method type codes B through W.

A

All method types specified for all active schedules. No comparisons are made, and all methods are attempted.

Default: A

Consider the following scenario. Employee Terry Jones is entered into the Notification Manager database, and has two 24 x 7 time blocks defined. One time block uses EMail 1 as a method and the other uses Numeric Pager 1. An NMFIND without MTUP specified is issued for Terry Jones. The value of MTUP defaults to A, and no comparisons are made. By default, both the EMail 1 and Numeric Pager 1 methods are attempted. However, since certain messages are of lesser importance than others, the EMail 1 method notification alone may sometimes suffice. The TYP parameter of the EMail 1 method is E; and TYP parameter of the Numeric Pager 1 method is P. An NMFIND for Terry Jones can be issued with MTUP set to E. The EMail 1 method will be attempted (E is part of E), and Numeric Pager 1 method will not (P is not part of E). These results will occur conversely if you set MTUP to P.

MTUP can contain multiple letters. For example, in the scenario above, MTUP could be set to EP, allowing both methods to be attempted (E is part of EP, P is part of EP). Setting MTUP to EP in this case is the same as using the default setting.

Consider the next scenario. Another time block is added for Terry Jones. This time block is set to use Voice 1 method, and its TYP parameter has the value of V. If a call to Terry desires to use EMail 1 and Numeric Pager 1 methods, but not the Voice 1 method, MTUP could be set to EP to make this occur (E is part of EP, P is part of EP, V is not part of EP.).

To preview which methods will be attempted when MTUP is specified for a particular NMFIND request without issuing a notification, run the Notification Manager utility program, listMTUP.rex. You can find this utility in the subdirectory SAMPLE\NM in your CA Automation Point installation directory.

TIME

(Optional) Specifies the time you want to use when determining how to get in touch with the specified person or group. The format is military time format (00:00 to 23:59).

Default: The current time

USERPARMS

(Optional) Specifies a list of method parameters whose associated values override any like-named parameters during the execution of NMFIND.REX. For example, assume this operand is defined as follows:

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

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

Examples:

Suppose you determined that when a particular JES is having difficulties you need to notify the lead JES systems programmer, Jim Smith. The CA Automation Point rule that trapped the error message from JES contains this clause:

```
REXX(NMFIND PERSON(JIM SMITH) TELL('JES is down'))
```

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

- If the contact method was the CA Automation Point voice technology, Jim receives a phone call at the phone number pointed to by his notification schedule.
- If the contact method was the numeric pager, Notification Manager pages Jim with the numeric message consisting of the phone number that he needs to call and an ID number that authorizes him to receive the voice message.
- If the contact method was the alphanumeric pager, the message appears on Jim's pager with a phone number and ID that he can use to obtain any information that was not sent using his pager. (For instance, the message to be sent may be longer than the length supported by his paging service.)

Extending the previous example, suppose you have written a REXX program that obtains control whenever JES is down and you have determined that you do not know how to handle the situation. Thus, you want to allow Jim to specify what should be done about JES being down. You can code a call to Notification Manager within your REXX program as follows:

```
CALL NMFIND.REX "PERSON(JIM SMITH) TELL('JES is down')",
  "ASK('What should I do', 'WARM START', 'COLD START')"
```

Your program receives a return code of 1 from NMFIND if Jim wants it to warm start JES (because WARM START is the first answer) and a return code of 2 if Jim wants it to cold start JES (because COLD START is the second answer).

Alternately, you may want to code REXX programs that handle various situations, and then code your automation so that Notification Manager invokes those programs based on the response it received from the person it calls. For example, you could code the following invocation of Notification Manager in your automation:

```
REXX(NMFIND PERSON(JIM SMITH) TELL('JES IS DOWN')
 ASK('What should I do',
    'WARM START'::JESSTART.REX WARM,
    'COLD START'::JESSTART.REX COLD,
    'IPL THE SYSTEM'::IPLSYS.REX)
```

Note: The example shown above is split across lines for presentation on the page. However, when you enter this code, it must be on a single line.

If Jim asks for a warm start, Notification Manager runs the JESSTART REXX program with a parameter of WARM. If Jim asks for a cold start, Notification Manager runs the JESSTART REXX program with a parameter of COLD. If Jim asks for a system IPL, Notification Manager runs the IPLSYS REXX program.

Notification Manager also supports group definition and notification. For example, if you define a group called CICS_SYS_PROGS, you could replace the PERSON(JIM SMITH) in the previous examples with GROUP(CICS_SYS_PROGS). Instead of attempting to contact just Jim Smith, Notification Manager systematically attempts to contact each member of the CICS_SYS_PROGS group until someone is successfully contacted. If you want, you can also tell Notification Manager to notify all members of the group by defining the group as a broadcast group.

Length of Message Sent

The length of the message that is being sent is limited according to where you issue the NMFIND.REX. The TELL operand holds the message to be sent.

Messages can be sent in three ways:

- From Rules
- From the command line
- From another REXX program

Sending Messages from Rules

You can issue an NMFIND.REX from Rules using the REXX keyword. The length of the call is limited by the 512-character rule limitation. The template to issue NMFIND.REX from rules is MSGID() REXX(NMFIND.REX NAME() TELL ()), which is 38 characters long. This means the value defined for the TELL operand can be at most 474 characters. The TELL string maximum is reduced by the lengths of the values specified for MSGID and NAME. In addition, the length of any optional NMFIND operands (for example, ASK or FAILUREREXX) that are specified, as well as the lengths of their values, further reduce the maximum length of the TELL operand.

Sending Messages from the Command Line

You can issue an NMFIND from a command line. The length of the NMFIND call is limited to 2048 characters. This is a limitation imposed by the operating system. The template for launching an NMFIND.REX is ASOREXX NMFIND NAME() TELL(). The maximum length of the TELL operand is reduced the same way as described in the section above for Rules.

Sending Messages from Another REXX Program

Larger messages can be sent if the NMFIND is launched from within another REXX program. These NMFIND calls can have a length of up to 30K characters. To send such messages from within CA Automation Point, write the message to a data store that is accessible to the REXX environment. Next, launch a REXX program from Rules. This program should access the stored data, create the NMFIND call, and launch NMFIND.REX.

The basic template to issue NMFIND from another REXX program is CALL NMFIND.REX NAME() TELL(). The maximum length of the TELL operand is reduced the same way as described in Rules above with the specification of additional operands.

Note: The length of the message actually sent by a specific Notification Manager method is limited by the value specified for the MaxMsgLen parameter for the method. The MaxMsgLen parameter is set based on the limitations of the medium that is used to send the notification. This means an NMFIND with a message of 30K can be run, but contacts set up to receive pages will only receive the length of the message defined by the MaxMsgLen parameter. For paging methods, the setting has traditionally been 240 characters.

Special Notes About Emergency Mode Processing

NMFIND provides you, the user of the NMFIND script, with emergency mode processing for a simple reason. As the coder of NMFIND in your automation, you have no control over the methods that are used to contact the person you need to reach, and many methods can take a long time to complete.

For instance, the LongVoice method, which invokes NMVOICE and tells it to try to call the person 20 times at intervals of five minutes and let the phone ring 10 times on each try, can take over 100 minutes to complete before returning control to NMFIND. Thus, even if there were other methods to be tried after the LongVoice method, they are not attempted for over 100 minutes. This is unacceptable when you need a response in a short amount of time.

Emergency mode processing circumvents this problem by providing you with a way to specify the amount of time each method can hold up the process of getting notification to someone who can handle the problem for which you invoked NMFIND. Emergency mode processing (using the EMERGENCYWAIT and FAILUREREXX operands) also gives you a way to specify the maximum amount of time you want to wait before attempting some other means (besides NMFIND) of obtaining a solution to a problem.

When you specify EMERGENCYINTVL(*n*), NMFIND always starts new actions approximately *n* seconds apart. Even if all prior actions complete so that none are running, NMFIND waits for the time interval to expire before starting a new action. NMFIND starts actions asynchronously on Windows by issuing the START command. Thus, each action runs as a separate process, using far more system resources than running all the actions synchronously inside a single process. Therefore, use this feature sparingly.

Before starting a new action, NMFIND checks all prior actions to see whether the request has been satisfied. If the entity had the perform all active methods flag set to OFF, the success of a single action satisfies the request. If the entity had the perform all active methods flag set to ON, all the actions that belong to the entity have to succeed before the request is satisfied. Once the request is satisfied, NMFIND does not submit any new actions.

Because NMFIND submits new actions without regard to the completion of prior actions, NMFIND can end up with several actions running simultaneously. Therefore, the NMFIND normal behavior of quitting after the first successful action completes is somewhat modified. After an action completes successfully, new actions are not submitted, but actions that are already running are allowed to complete normally.

If NMFIND is only telling asynchronously (no ASK operand was specified), the first time NMFIND succeeds in telling the message, it considers the NMFIND request to be complete. Once NMFIND considers the request complete, it stops starting new actions, but actions that it has already started continue to run and may eventually succeed in telling the message to the person that they were supposed to contact. Thus, more than one person can receive the message. (In the synchronous case, only one person receives the message.)

If you are asking a question asynchronously, the first answer received from someone is considered to be the answer to the question (even if that person was not the first person that NMFIND attempted to contact). Once the question is answered, NMFIND does not start any new actions, but permits existing actions to continue to run. If another action subsequently succeeds in reaching someone, then that person hears the TELL message, the ASK question and answers, and the answer that was chosen, but NMFIND does not allow the person to answer the question.

Methods Called by NMFIND

The following are Notification Manager methods that are called by NMFIND.

NMMAIL

Provides a means for NMFIND to contact a person or group using the local e-mail system.

NMMAILPG

Provides a front-end to the NMMAIL program. NMFIND uses this command to contact a person or group by initiating an alphanumeric page through the e-mail system.

NMNETSND

Provides a means for NMFIND to contact people through the local area network (LAN) by issuing a NET SEND command.

NMPAGE

Provides a means for NMFIND to contact someone through a TAP alphanumeric or numeric pager.

NMPAGE2WAY

Provides a means for NMFIND to contact someone through a 2-way messaging device.

NMSPEAK

The NMSPEAK program is used by NMFIND to contact a person or group through speech using the TCP/IP network.

NMTAP

Provides a means for NMFIND to contact someone through a TAP alphanumeric or numeric pager. It differs from NMPAGE in that the REXX code talks directly to the modem instead of using the ADDRESS VOX PAGE command to perform the page.

NMTAP supports the batching of pages (that is, sending multiple pages to a paging service with a single phone call). NMTAP is designed so that you can easily modify it to support a different protocol than TAP. For more information, see the NMTAP.TXT file in the Distrib directory.

NMVOICE

Provides a means for NMFIND to contact someone through a voice card and, optionally, ask them to make a selection from a set of options.

NMMAIL Method

The NMMAIL program is used by NMFIND to contact a person or group using the local e-mail system.

Specify the following items when defining a method that uses NMMAIL:

- The invocation string is NMMAIL
- NMMAIL supports *only* the TELL parameter of NMFIND. For this reason, you must choose to make any method that invokes this program support only the TELL parameter.

The following is a list of method parameters that are to be stored in the Notification Manager database for each NMMAIL method definition:

Parameter	Required
AnswerWait	No
CCList	No
Debug	No
MailRetry	No
MailWait	No
MaxListLen	No
MaxMsgLen	No
MaxSigLen	No
MaxSubjLen	No
NMfindWait	No
SignatureText	No
SubjectText	No
System	No
TellExact	No
ToList	Yes

NMMAIL Parameter Descriptions

AnswerWait

Specifies the number of seconds that the e-mail request is to delay the processing of subsequent actions in the call tree.

Default: 60

CCList

Specifies the e-mail recipient names that are to receive a carbon copy of this e-mail.

Default: There is no default.

Debug

Determines whether Notification Manager is to generate debugging messages at the command prompt or the AXCREXX window and in the ASOTRACE log.

Default: NO

MailRetry

Specifies the number of times the e-mail request is to be retried if it does not go through.

Default: 4

MailWait

Specifies the number of seconds that Notification Manager is to wait before trying to send an e-mail request again.

Default: 60

MaxListLen

Specifies the maximum length of an e-mail recipient list (either the TO or CC list), including the separators.

Default: 1000

MaxMsgLen

Specifies the maximum length of a message that the e-mail system supports.

Default: 30000

MaxSigLen

Specifies the maximum length of a Signature text string (Mail ID).

Default: 40

MaxSubjLen

Specifies the maximum length of a Subject text string.

Default: 240

NMFindWait

Specifies the number of seconds that Notification Manager allows the person who received the e-mail to call-in and handle the item before the FAILURE command is executed.

Database Default: 300

Internal Default: 900

SignatureText

Specifies the optional user-defined text that identifies or tracks the e-mail message.

Default: Notification Manager Automated Response

SubjectText

Specifies the Subject line of the generated e-mail message. The Notification Manager adds a prefix of "NM: " to the generated e-mail subject line and an additional prefix of "Callback Item XXXX", if the ASK operand is specified.

Internal Default: Notification Manager TELL setting

System

Specifies the name of the system that is to handle all notification server requests.

Internal Default: The local system name

TellExact

Determines whether the TELL text from the NMFIND command is to be written to the e-mail verbatim. Values are:

YES

The TELL text is written to the e-mail verbatim.

NO

Allows Notification Manager to add a call-in number and item number to the TELL text.

Default: There is no default.

ToList

Specifies the e-mail recipient names that are to receive this message. If more than one name is specified, these names must be separated by a semicolon (;).

Default: There is no default.

NMMAILPG Method

The NMMAILPG program provides a front-end to the NMMAIL program. NMFIND uses NMMAILPG to contact a person or group by initiating an alphanumeric page using the e-mail system.

Specify the following items when defining a method that uses NMMAILPG:

- The invocation string is NMMAILPG
- NMMAILPG supports *only* the TELL parameter of NMFIND. For this reason, you must choose to make any method that invokes this program support only the TELL parameter.

The following is a list of method parameters that are to be stored in the Notification Manager database for each NMMAILPG method definition:

Parameter	Required
AnswerWait	No
CCList	No
Debug	No
MailRetry	No
MailWait	No
MaxListLen	No
MaxMsgLen	No
MaxSigLen	No
MaxSubjLen	No
NMfindWait	No
Numeric	No
PagerPW	No
Phone	Yes
PhoneNumLength	Yes
PIN	Yes
SignatureText	No
SubjectText	No
System	No
TellExact	No

Parameter	Required
ToList	Yes
ToListFormula	Yes

NMMAILPG Parameter Descriptions

AnswerWait

Specifies the number of seconds that the e-mail request is to delay the processing of subsequent actions in the call tree.

Default: 60

CCList

Specifies the e-mail recipient names that are to receive a carbon copy of this e-mail.

Debug

Determines whether Notification Manager is to generate debugging messages at the command prompt or the AXCREXX window and in the ASOTRACE log.

Default: NO

MailRetry

Specifies the number of times the e-mail request is to be retried if it does not go through.

Default: 4

MailWait

Specifies the number of seconds that Notification Manager is to wait before trying to send an e-mail request again.

Default: 60

MaxListLen

Specifies the maximum length of an e-mail recipient list (either the TO or CC list), including the separators.

Default: 1000

MaxMsgLen

Specifies the maximum length of a message that the e-mail system supports.

Default: 30000

MaxSigLen

Specifies the maximum length of a Signature text string (Mail ID).

Default: 40

MaxSubjLen

Specifies the maximum length of a Subject text string.

Default: 240

NMFindWait

Specifies the number of seconds that Notification Manager allows the person who received the e-mail to call in and handle the item before it executes the FAILURE command.

Database Default: 300

Internal Default: 900

Numeric

Determines whether the pager to be contacted is numeric (YES) or not (NO).

Default: NO

PagerPW

Specifies the password for the pager service. Specify the password only if required by the paging service.

Default: No password

Phone

Specifies the phone number that is to issue a page to the pager. Everything except the numeric digits are omitted from this phone number.

Note: Some pager companies require the phone number to be in a special format (for example, not entering an area code or dialing 1 before an 800 number).

PhoneNumLength

Specifies the number of numeric digits that a properly specified phone number should have.

Notification Manager uses this parameter to generate warning messages during execution. If you do not want to see warning messages or if the number of digits that the pager service requires is variable, specify 0 for this parameter.

PIN

Specifies the PIN of the pager.

SignatureText

Specifies the optional user-defined text that identifies or tracks the e-mail message.

Default: Notification Manager Automated Response

SubjectText

Specifies the Subject line of the generated e-mail message.

Internal Default: Notification Manager item number *xxxx*

System

Specifies the name of the system that is to handle all notification server requests.

Internal Default: The local system name

TellExact

Determines whether the TELL text from the NMFIND command is to be written to the e-mail verbatim. Values are:

YES

The TELL text is written to the e-mail verbatim.

NO

Allows Notification Manager to add a call-in number and item number to the TELL text.

ToList

Specifies the e-mail recipient names that are to receive this message. If more than one name is specified, these names must be separated by a semicolon (;).

ToListFormula

Specifies the formula, or set of rules, the pager service is to follow to combine the Phone, PagerPW, and PIN parameters, generating an e-mail ID.

You can specify any valid REXX expression in this parameter. To use a parameter value in the expression, specify the name of the parameter, not its value. Use an exclamation point (!) for the pipe symbol (|), and use a single quote (') for any quotes.

For example, if the Skytel service required you to take the PIN and then add the string "@skytel.com", the formula would be:

```
PIN!!'@skytel.com'
```

NMNETSND Method

The NMNETSND program provides a means for NMFIND to contact people via the local area network (LAN) by issuing a NET SEND command. A message is sent to a specified computer name or a domain.

Specify the following items when defining a method that uses NMNETSND:

- The invocation string is NMNETSND.
- NMNETSND supports *only* the TELL parameter of NMFIND. For this reason, you must choose to make any method that invokes this program support only the TELL parameter.

Define the following parameters for any method that uses NMNETSND as its invocation string. If a parameter is marked as required, you must define it as a parameter for the method. If a parameter is not required, you do not need to define it in the database. If you do not define the parameter, the default value is used whenever the method is invoked.

Note: If you do not define the parameter at the method level, you cannot set a value for a parameter at the entity or time-block level (that is, override the default value).

Parameter	Required
AllUsers	No
Answerwait	No
Debug	No
Domain	No
Greeting	No
MaxMsgLen	No
Name	Yes
NMFindWait	No
Retry	No
RetryWait	No
TellExact	No

NMNETSND Parameter Descriptions

AllUsers

Set to YES to send the message to all Windows users connected to the Windows server. For a NET SEND to work, you must either set AllUsers to YES, specify a value for Domain, or specify a value for Name.

Default: NO

Answerwait

Specifies how many seconds the NET SEND request should hold up processing of subsequent actions in the call tree. Set this number to a high value if it is more important to get an answer from the particular person being notified than it is to get a quick answer.

Default: 60

Debug

Determines whether Notification Manager is to generate debugging messages at the command prompt or the AXCREXX window and in the ASOTRACE log.

Default: NO

Domain

Specifies to whom the message should be sent (Windows user name, Windows computer name, or Windows messaging name). For a NET SEND to work, you must either set AllUsers to YES, specify a value for Domain, or specify a value for Name.

Greeting

Specifies the string that Notification Manager will place before the TELL message in the Messenger dialog box

MaxMsgLen

Specifies the maximum message length the NET SEND command supports

Default: 128

Name

Specifies to whom the message should be sent (user name, computer name, or message name). For a NET SEND to work, you must either set AllUsers to YES, specify a value for Domain, or specify a value for Name.

NMFindWait

Forces NMFIND to guarantee the person that was contacted at least has a certain amount of addition time to call-in. The value is in seconds.

You should set this value higher it is difficult for the person contacted to reach a phone. The actual effect of this parameter is to cause the NMFIND to postpone running the FailureRexx (if necessary) in order to guarantee that the person paged has adequate time to reach a phone

Default: 300

Retry

Specifies the number of times the NET SEND is retired when it does not succeed. The total number of send attempts will be 1 more than the Retry value.

Default: 4

RetryWait

Specifies the number of seconds to wait before each time the NET SEND is retried.

Default: 60

TellExact

Determines whether Notification Manager sets the TELL message with a call-in number and item number. Values are:

YES

Issue the message without the call-in number and item number added.

NO

Issue the message with the call-in number and item number.

Default: NO

NMPAGE Method

The NMPAGE program provides a means for NMFIND to contact someone with a TAP alphanumeric or numeric pager. NMPAGE only communicates with TAP alphanumeric pager services. Therefore, you can only use NMPAGE to page a numeric pager if that pager is reachable with an alphanumeric pager service. You cannot call NMPAGE directly; rather, you must install it as the invocation string for a method in your Notification Manager database and use NMFIND to invoke it.

Specify the following items when defining a method that uses NMPAGE:

- The invocation string is NMPAGE.
- NMPAGE supports *only* the TELL parameter of NMFIND. For this reason, you must choose to make any method that invokes this program support only the TELL parameter.

As mentioned previously, NMPAGE supports the TELL parameter of NMFIND, but it does not support the ASK parameter of NMFIND (it can be used only as a TELL method). The text sent to a numeric pager is simply the phone number on which to call in and the item number. The text sent to an alphanumeric pager is the phone number on which to call in, the item number, and some or all the text of the TELL message.

Define the following parameters for any method that uses NMPAGE as its invocation string. If a parameter is marked as “required,” you must define it as a parameter for the method. If a parameter is not required, you do not need to define it in the database. If you do not define the parameter, the default value is to be used whenever the method is invoked. Remember, you cannot set a value for a parameter at the entity or time-block level (that is, override the default value) if you do not define the parameter at the method level.

The following table lists the method parameters that are stored in the Notification Manager database for each NMPAGE method definition:

Parameter	Required
AnswerWait	No
BaudRate	No
Comport	No
DataBits	No
Debug	No
FinalConfirm	No
Greeting	No
MaxMsgLen	No
ModemInitString	No
NMfindWait	No
Numeric	No
PageRetry	No
PagerID	Yes
PagerPW	No
PageWait	No
Parity	No
Phone	Yes
RetryRCList	No
StopBits	No
System	No
TapiDeviceID	No
TellExact	No

NMPAGE Parameter Descriptions

AnswerWait

Specifies the number of seconds that NMPAGE should wait for an answer (through a call-in) before allowing NMFIND to try the next method of contacting someone.

Setting this parameter to a higher value allows the person who was paged more time to handle the problem before Notification Manager attempts to contact someone else. Therefore, you should set the value high when it is important for a particular person to answer the problem. Setting the value lower means that you are more likely to get a quick response to a problem.

Default: 300

BaudRate

Specifies the baud rate at which to set the modem. Valid values are 300, 1200, 2400, 4800, and 9600.

Default: 9600

Comport

Specifies the communications port to use when dialing the pager service.

Default: Selects the first available communications port enabled for use by notification services.

DataBits

Specifies the number of databits per character used by the modem. The TAP protocol requires this value to be set to 7. Valid values are 4, 5, 6, 7, and 8.

Default: 7

Debug

Determines whether Notification Manager is to generate debugging messages at the command prompt or the AXCREXX window and in the ASOTRACE log. Possible values are YES and NO.

Default: NO

FinalConfirm

Some pager services do not strictly conform to TAP protocol for final confirmation during the disconnection sequence. Use the FinalConfirm parameter to specify an acceptable compliance to the TAP disconnection sequence. Values are:

COMPLETE

The disconnection sequence must fully comply with the TAP protocol.

PARTIAL

A partially TAP-compliant disconnection sequence is acceptable.

NONE

A TAP-compliant disconnection sequence is not required.

Default: COMPLETE

Greeting

Specifies the message to present at the top of the pager's display (before the TELL and ASK messages are presented). The argument can be either a string or voice file. If the argument is a voice file, the name of the voice file must not contain any blanks and the last four characters must be .VOX.

Default: NM CALLING - CALL *nnn-nnnn* ABOUT ITEM *nnnn*

MaxMsgLen

Specifies the maximum length of the message to be sent to the pager.

Default: 100 for an alphanumeric pager (Numeric=NO), and 20 for a numeric pager (Numeric=YES)

ModemInitString

Specifies the string to pass to the modem to initialize it before it makes the call.

The semicolons in the string are used to delimit the separate strings that are passed to the modem (with a one second wait between each string).

Default: ATZ;AT&C1&D2;ATV1Q0X4;ATS0=0S2=128S7=55

NMfindWait

Note: Make sure that you are familiar with the AnswerWait parameter before reading this description. The AnswerWait parameter controls the number of seconds that Notification Manager waits for a paged person to answer the page, before it attempts to contact anyone else. Once the wait time has expired, Notification Manager attempts to contact other people.

The NMfindWait parameter forces NMFIND to guarantee that the person who was paged has at least a certain additional amount of time to call-in to answer the page. The value is in seconds.

You should set this value higher if it is difficult for the person paged to reach a phone and lower if it is easy. The actual effect of this parameter is to cause NMFIND to postpone running the FailureRexx command (if necessary) in order to guarantee that the person paged has adequate time to reach a phone and answer the page through a call-in.

Default: 900

Numeric

Specifies whether the pager is numeric or alphanumeric. Valid values are:

YES

The page device is a numeric pager.

Numeric page devices are limited to messages that contain numbers 0-9, hyphens, parentheses, and commas.

When the call-in feature is not enabled, the numeric TELL message and the notification item number are sent to the page device. If the TELL message includes invalid numeric characters, NMPAGE filters them out and substitutes them with a hyphen to indicate that a string of non-numeric characters was included in the message, but could not be displayed from the numeric page device. For example, if the message for item 1234 was "Workstation 7 recycled at 12 noon," the message sent to the pager would be "7-12 1234." If no valid numeric characters exist in the TELL message, only the item number is sent to the pager. For example, if the message for item 1235 was "Workstation number seven recycled at noon," the message sent to the pager would be "1235."

When the call-in feature is enabled, the numeric message sent to the page device includes the call-in phone number and the notification item number. The page recipient must use the call-in feature to retrieve the actual TELL message.

Note: Set TellExact to YES to send TELL-only valid numeric messages to the numeric page device without the item number.

When Numeric is set to YES, the TELL message is not prefixed with a greeting.

NO

The page device is an alphanumeric pager

Default: NO

PageRetry

If a page request fails, PageRetry specifies the number of times a page is retried, provided the page fails with a return code specified by the RetryRCList parameter. The total number of attempts to resubmit a page will be one more than the value of PageRetry.

Default: 3

PagerID

Specifies the pager ID that NMPAGE is to tell the pager service needs to be paged.

PagerPW

Specifies the password for the pager service.

PageWait

Specifies the number of seconds to wait between retries of the page.

Default: 60

Parity

Specifies the method used by the modem for error checking. The TAP protocol requires this value to be set to E. Valid values are: N (None), E (Even), O (Odd), M (Mark), S (Space).

Default: E

Phone

Specifies the phone number of the pager service.

RetryRCList

Specifies the list of return codes used by NMPAGE to determine if retries will be attempted. If the return code is on the list, NMPAGE will retry the number times defined by PageRetry. If the return code is not in this list, NMPAGE will not reattempt it. The list of return codes must be separated by commas for this parameter to work properly.

StopBits

Specifies the time between transmitted characters used by the modem, represented by a number. The TAP protocol requires this value to be set to 1. Valid values are: 0, 1, and 2.

Default: 1

System

Specifies the name of the system that is running the notification server you want to perform the call. The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name

TapiDeviceID

(Optional) Specifies the numeric ID of the Telephony Application Programming Interface (TAPI) device or modem, (installed on the notification server) you want to use.

To enhance performance, if there is already an established connection to the specified pager service, notification server overrides the TAPIDEVICEID setting and issues the page over the existing connection.

Note: To view all of the Telephony Application Programming Interface (TAPI) devices that are installed under Windows on the Notification Server, issue the GETTAPIDEVICELIST command.

Default: Selects the first available TAPI device

Notes:

- The modem must be properly installed within Windows. See the *Administrator Guide* for details.
- The TAPIDEVICEID and COMPORT operands are mutually exclusive.
- When you specify TAPIDEVICEID, the PAGE command overrides default settings configured in the Alphanumeric Paging Options dialog. It then initiates the page using the specified TAPI device ID, and uses TAPI to initialize the modem.
- When neither COMPORT nor TAPIDEVICEID is specified, the next available communications port is used to initiate a page, according to settings in the Alphanumeric Paging Options dialog.

TellExact

Determines whether Notification Manager sets the TELL message with a call-in number and item number. Values are:

YES

Issue the page message without the call-in number and item number added.

NO

Issue the page message with a call-in number and item number.

Default: NO

NMPAGE2WAY Method

The NMPAGE2WAY program provides a means for NMFIND to contact someone with a 2-way messaging device. NMPAGE2WAY only communicates with paging services that support either the SNPP protocol (Simple Network Paging Protocol) or the WCTP protocol (Wireless Communication Transfer Protocol). You cannot call NMPAGE2WAY directly; rather, you must install it as the invocation string for a method in your Notification Manager database and use NMFIND to invoke it.

Specify the following items when defining a method that uses NMPAGE2WAY:

- The invocation string is NMPAGE2WAY.
- NMPAGE2WAY supports both the TELL and the ASK parameters of NMFIND, as long as the same method is used to process both of these parameters for the current invocation of NMFIND. This means that you can make a method that uses this program support TELL or BOTH (Tell and Ask), but it cannot be used to support ASK parameters of a different type of method.

When you specify the ASK parameter, the contacted person can respond to the ASK message by either replying to the page request directly from the 2-way messaging device or using the call-in feature of Notification Manager.

The text message that is sent to a numeric pager is simply the telephone line on which to call-in and the item number. The text message sent to an alphanumeric pager is the item number, the telephone line on which to call-in (optional), some or all of the text contained in the TELL message, the question contained in the ASK parameter (if specified), and the list of answers contained in the ASK parameter (if specified).

In addition to sending numeric or alphanumeric messages to the 2-way messaging device, NMPAGE2WAY also provides confirmation of message delivery by allowing Notification Manager to wait until a specified message delivery status has been reached. This message delivery status is defined using the *CompleteStatus* method parameter, which is discussed in the method parameter list that follows.

Define the following parameters for any method that uses NMPAGE2WAY as its invocation string. If a parameter is marked as “required”, you must define it as a parameter for the method. If a parameter is not required, you do not need to define it in the database. If you do not define the parameter, the default value will be used whenever the method is invoked.

Note: You cannot set a value for a parameter at the entity or time-block level (that is, override the default value) if you do not define the parameter at the method level.

The following table lists the method parameters that are stored in the Notification Manager database for each NMPAGE2WAY method definition:

Parameter	Required
AnswerWait	No
CompleteStatus	No
Debug	No
Greeting	No
MaxMsgLen	No
NMFindWait	No
Numeric	No
PagerID	Yes
PagerPW	No
Provider	Yes
ReplyPrefix	No
SendAck	No
System	No
TellExact	No
UseMCR	No

NMPAGE2WAY Parameter Descriptions

AnswerWait

Specifies the number of seconds that NMPAGE2WAY should wait for a reply (either through a reply to the page request or through a call-in) before allowing NMFIND to try the next method of contacting someone. Once this time interval elapses, the page recipient must place a call to NMANSWER to resolve the issue (no further page replies will be accepted).

Default: 300

CompleteStatus

Specifies the 2-way page status value that NMPAGE2WAY will use to determine both when the notification attempt has completed and whether the notification was successful. NMPAGE2WAY waits for the page request to reach the status specified by this parameter before either processing ASK parameters or returning control to NMFIND. If the time interval specified by the AnswerWait parameter elapses before the page request reaches the required status, an error is returned to NMFIND and processing continues. The possible values for this parameter are:

QUEUED

The page request was successfully submitted to the pager service

DELIVERED

The page request was sent to the pager and a positive receipt acknowledgement was received from the pager by the pager service

READ

The pager acknowledged receipt of the page request and the page request was viewed by the recipient

REPLIED

The page recipient received the page request and sent a reply to this page request back to the pager service

Note: The list of supported status indicators may vary by pager service provider. Contact your pager service provider for details.

Default: QUEUED

Debug

Determines whether Notification Manager is to generate debugging messages at the command prompt or the AXCREXX window and in the ASOTRACE log.

Default: NO

Greeting

Specifies the message to present at the top of the pager's display (before the TELL and ASK messages are presented).

Default: NM Calling

MaxMsgLen

Specifies the maximum length of the message to be sent to the pager. If the message is longer than the maximum length allowed, it is truncated (but the page is still sent). If the message is truncated and the call-in feature (NMANSWER) is configured, the message will be prefixed with call-in information so that the full message may be accessed using the call-in feature.

Note: Although the first *MaxMsgLen* number characters of a message are sent to the pager service, the message is still subject to message length limitations imposed by the pager service. Contact your pager service provider for message length limitation and message handling when the length limitation is exceeded.

Default: 500

NMFindWait

This parameter forces NMFIND to guarantee that the person who was paged has at least a certain additional amount of time (in seconds) after the method completes to call-in to answer the page.

Default: 300

Numeric

Specifies whether the page device is numeric or alphanumeric.

Numeric page devices are limited to messages that contain numbers 0-9, hyphens '-', parentheses '()', and commas ','.

Valid values for this parameter are:

YES

The page device is a numeric pager.

Note: When Numeric=YES, set TellExact=YES to send a valid numeric TELL message to the numeric page device.

Set TellExact=NO to send a numeric message containing a call-in phone number and the notification item number. The receipt of the page will have to use the call-in feature to retrieve the actual TELL message.

Note: When Numeric=YES, the TELL message is not prefixed with a Greeting.

NO

The page device is an alphanumeric pager.

Default: NO

PagerID

Specifies the pager ID of the page device that NMPAGE2WAY requests the pager service to page.

PagerPW

Specifies the password associated with the specified pager ID that NMPAGE2WAY uses to issue the page request. The use of pager passwords varies by pager service provider. Contact you pager service provider for details.

Default: No password

Provider

Specifies the name of the pager service that is to receive the page request (as defined in the 2-Way Paging Setup dialog).

ReplyPrefix

When a reply to a page request is sent from the 2-way messaging device, the device or the paging service itself may add additional information to the beginning of the reply (contact your paging provider for details). If this additional information is consistent for each page reply, the NMPAGE2WAY method can filter out this additional information before comparing the reply to the list of specified ASK parameters. In addition to the prefix added by the pager service, the Notification Server adds the additional prefix REPLIED to all page replies received. Since the word REPLIED always appears at the beginning of the page reply, this parameter must start with the word REPLIED, followed by any constant prefix added by the pager service.

All messages received by Notification Server from a 2-way messaging device follow this format:

REPLIED prefixed_text page_reply_message

REPLIED

The REPLIED keyword, which is added by the Notification Server, is always the first word of the reply message.

prefixed_text

A prefix text containing additional information may or may not be automatically added by the pager service or the page device. Contact your pager service provider for details.

page_reply_message

The reply message sent by the page recipient.

To filter out the beginning text of a reply message so that Notification Manager can identify the reply to the page, set ReplyPrefix to 'REPLIED *prefixed_text*'

Default: REPLIED

SendAck

Determines whether positive acknowledgement messages are sent to the page recipient during processing of the notification attempt. The possible values for this parameter are:

YES

Acknowledgement messages are sent to the page recipient during processing of the notification attempt.

NO

No additional messages are sent to inform the page recipient of events concerning the current notification attempt.

Default: NO

System

Specifies the name of the system that is running the Notification Server to which NMPAGE2WAY requests associated with the method is to be sent.

Default: The name of the Notification Server specified in the Notification Startup Options dialog.

TellExact

Determines whether Notification Manager prefixes the TELL message with a call-in number and an item number. The possible values are:

YES

Issue the page message without the header text, call-in number, and item number added.

NO

Issue the page message with a message header text, call-in number, and an item number.

Default: NO

UseMCR

Determines whether the NMPAGE2WAY method uses the MCR (Multiple Choice Response) capability of the 2-way paging device to send the specified ASK parameters to the page recipient. If the specified pager or pager service does not support MCR messages (sometimes called “programmed replies”), set this parameter to NO. The possible values for this parameter are:

YES

The ASK parameters are sent to the pager using the MCR capability of the underlying 2-way paging protocols.

NO

The ASK parameters are appended to the existing TELL message.

Note: Support for MCR messages vary by paging service provider. Contact your paging service provider for details

Default: NO

NMSPEAK Method

The NMSPEAK program is used by NMFIND to contact a person or group through speech using the TCP/IP network. For this method to work, the speech client must be installed on the computer of the person or group.

Specify the following items when defining a method that uses NMSPEAK:

- The invocation string is NMSPEAK
- NMSPEAK supports *only* the TELL parameter of NMFIND. For this reason, you must choose to make any method that invokes this program support only the TELL parameter.

The following is a list of method parameters that are to be stored in the Notification Manager database for each NMSPEAK method definition:

Parameter	Required
AnswerWait	No
Debug	No
Greeting	No
MaxMsgLength	No
Name	Yes
NMFindWait	No
Retry	No

Parameter	Required
RetryWait	No
TellExact	No

NMSPEAK Parameter Descriptions

AnswerWait

Specifies the number of seconds that the speech notification request is to delay the processing of subsequent actions in the call tree.

Default: 60

Debug

Determines whether Notification Manager is to generate debugging messages at the command prompt or the AXCREXX window and in the ASOTRACE log.

Default: NO

Greetings

Specifies the string that Notification Manager places before the TELL message.

MaxMsgLen

Specifies the maximum length of a message that the speech notification system supports. We recommend that you not increase this limit over the default.

Default: 256

Name

Specifies the name of the computer to which to send the message. The computer must have TCP/IP connectivity with the computer where the message is sent from. If the name contains blank characters, enclose it in quotation marks {` `}.

Default: The local host

NMFindWait

Notification Manager guarantees that the target person has at least this many seconds to call-in and handle the item before the FailureCommand is executed. Set this parameter to a high value if it is more important to get an answer than to resolve the item quickly.

Default: 300

Retry

Specifies how many times to retry the method action when it does not go through. The total number of send attempts will be 1 more than the Retry value.

Default: 4

RetryWait

Specifies how many seconds to wait before each time the method action is retried.

Default: 60

TellExact

TellExact{YES} causes the TELL text from the NMFIND command to be sent `as-is`. TellExact{NO} allows Notification Manager to add a call-in number and an item number to your TELL text. If you attempt an illegal operation, NM can force this to NO.

Default: NO

NMTAP Method

The NMTAP program provides a means for NMFIND to contact someone with a TAP alphanumeric or numeric pager. It differs from NMPAGE in that the REXX code talks directly to the modem instead of using the ADDRESS VOX PAGE command to perform the page. NMTAP supports the batching of pages (that is, sending multiple pages to a paging service with a single phone call). NMTAP is designed so you can easily modify it to support a different protocol than TAP. For more information, see the nmtap.txt file in the Distrib directory.

NMTAP only communicates with TAP alphanumeric pager services. Therefore, you can only use NMTAP to page a numeric pager if that pager is reachable with an alphanumeric pager service. You cannot call NMTAP directly; rather, you must install it as the invocation string for a method in your Notification Manager database and use NMFIND to invoke it.

Specify the following items when defining a method that uses NMTAP:

- The invocation string is NMTAP.
- NMTAP supports *only* the TELL parameter of NMFIND. For this reason, you must choose to make any method that invokes this program support only the TELL parameter.

As mentioned previously, NMTAP supports the TELL parameter of NMFIND, but it does not support the ASK parameter of NMFIND (it can be used only as a TELL method). The text sent to a numeric pager is the line on which to call in and the item number. The text sent to an alphanumeric pager is the line on which to call in, the item number, and some or all the text of the TELL message.

Define the following parameters below for any method that uses NMTAP as its invocation string. If a parameter is marked as required, you must define it as a parameter for the method. If a parameter is not required, you do not need to define it in the database. If you do not define the parameter, the default value is used whenever the method is invoked. Remember, you cannot set a value for a parameter at the entity or time-block level (that is, override the default value) if you do not define the parameter at the method level.

The following is a list of method parameters that are to be stored in the Notification Manager database for each NMTAP method definition:

Parameter	Required
AnswerWait	No
BatchPage	No
BatchWait	No
BaudRate	No
Comport	No
DataBits	No
Debug	No
Greeting	No
MaxMsgLen	No
ModeCmd	No
ModemInitString	No
ModeString	No
NMfindWait	No
Numeric	No
PageRetry	No
PagerID	Yes
PagerPW	No
PageWait	No
Parity	No
Phone	Yes
StopBits	No
TellExact	No

NMTAP Parameter Descriptions

AnswerWait

Specifies the number of seconds that NMPTAP should wait for an answer (using a call-in) before allowing NMFIND to try the next method of contacting someone.

Setting this parameter to a higher value allows the person who was paged more time to handle the problem before Notification Manager attempts to contact someone else. Therefore, you should set the value high when it is important for a particular person to answer the problem. Setting a lower value means that you are more likely to get a quick response to a problem.

Default: 300

BatchPage

Determines whether the page should be batched (that is, sent on the same phone call to the paging service) with any other pages that are currently being sent to the same paging service. Values are:

YES

Allow the page to be batched

NO

Do not allow the page to be batched

Default: NO

BatchWait

Specifies the number of seconds that batch processing should hold on to the paging service while waiting for another page to come in for that service. For most instances, this value should not be greater than 30 seconds.

Note: This parameter is valid only when BatchPage(YES) is specified.

Default: 30

BaudRate

Specifies the baud rate at which to set the modem. Valid values are: 300, 1200, 2400, 4800, and 9600.

Default: 9600

Comport

Specifies the communications port to use when dialing the pager service.

Default: Selects the first available communications port.

DataBits

Specifies the data bits to which to set the modem. The TAP protocol requires this value to be set to 7. Valid values are: 4, 5, 6, 7, and 8.

Default: 7

Debug

Determines whether Notification Manager is to generate debugging messages at the command prompt or the AXCREXX window and in the ASOTRACE log. Possible values are YES and NO.

Default: NO

Greeting

Specifies the message to present at the top of the pagers display (before the TELL and ASK messages are presented). The argument can be either a string or voice file. If the argument is a voice file, the name of the voice file must not contain any blanks and the last four characters must be .VOX.

Default: NM CALLING - CALL *nnn-nnnn* ABOUT ITEM *nnnn*

MaxMsgLen

Specifies the maximum length of the message to be sent to the pager.

If the message is longer than the maximum length allowed, it is truncated (but the page is still sent). If the message is truncated and the call-in feature (NMANSWER) is configured, the message will be prefixed with call-in information so that the full message may be accessed using the call-in feature.

Default: 100 for an alphanumeric pager, Numeric=NO, and 20 for a numeric pager, Numeric=YES.

ModeCmd

Specifies the name of the mode command that should be used to set the mode for the serial COM port.

You should specify this parameter if you are using specialized piece of hardware that requires a proprietary MODE command.

Default: MODE

ModemInitString

Specifies the string to pass to the modem to initialize it before it makes the call.

The semicolons in the string are used to delimit the separate strings that will be passed to the modem (with a one second wait between each string).

Default: ATZ;AT&C1&D2;ATV1Q0X4;ATS0=0S2=128S7=55

ModeString

The default processing of the ModeCmd parameter generates a string that uses the syntax of the Windows MODE command. If you are using a proprietary MODE command that has a different syntax, you should specify the **entire** MODE command, including the command name at the front.

Note: If this parameter is specified, the ModeCmd parameter is ignored.

NMfindWait

Note: Make sure that you understand the AnswerWait parameter (described previously) before reading this description. The AnswerWait parameter controls the number of seconds that Notification Manager waits for a paged person to answer the page before it attempts to contact anyone else. Once the wait time has expired, Notification Manager attempts to contact other people.

The NMfindWait parameter forces NMFIND to guarantee that the person who was paged has at least a certain additional amount of time to call-in to answer the page. The value is in seconds.

You should set this value higher if it is difficult for the person paged to reach a phone and lower if it is easy. The actual effect of this parameter is to cause NMFIND to postpone running the FailureRexx command (if necessary) to guarantee that the person paged has adequate time to reach a phone and answer the page using a call-in.

Default: 900

Numeric

Specifies whether the pager is numeric or alphanumeric. Numeric pages are limited to messages containing the numbers 0 through 9, hyphens, parentheses, and commas.

Valid values are:

YES

The page device is a numeric pager.

NO

The page device is an alphanumeric pager

Note: When Numeric is set to YES, set TellExact to YES to send a valid numeric message to the numeric page device. Set TellExact to NO to send a numeric message containing a call-in phone number and notification item number. The receipt of the page must use the call-in feature to retrieve the actual TELL message.

When Numeric is set to YES, the TELL message is not prefixed with a greeting.

Default: NO

PageRetry

If a page request fails, PageRetry specifies the number of times a page is retried, provided the page fails with a return code specified by the RetryRCList parameter. The total number of attempts to resubmit a page will be one more than the value of PageRetry.

Default: 3

PagerID

Specifies the pager ID of the paging device that NMTAP requests the pager service to page.

PagerPW

(Optional) Specifies the password required by the paging service. (Most paging services do not require a password.)

PageWait

Specifies the number of seconds to wait between retries of the page.

Default: 60

Parity

Specifies the method used by the modem for error checking. The TAP protocol requires this value to be set to E. Valid values are: N (None), E (Even), O (Odd), M (Mark), S (Space).

Default: E

Phone

Specifies the phone number of the pager service.

StopBits

Specifies the time between transmitted characters used by the modem. The TAP protocol requires this value to be set to 1. Valid values are: 0 , 1, and 2.

Default: 1

TellExact

Determines whether the TELL text from the NMFIND command is to be written verbatim. Values are:

YES

The TELL text is written verbatim

NO

Additional information is added to the TELL text

Default: NO

NMVOICE Method

NMVOICE provides a means for NMFIND to contact someone using a voice card and, optionally, ask them to make a selection from a set of options. You cannot call NMVOICE directly; rather, you must install it as the invocation string for a method in your Notification Manager database and use NMFIND to invoke it.

Note: NMVOICE also allows the person it contacts to record a message that is heard by all the people that NMFIND subsequently calls. To invoke this feature, the person contacted must press the 0 key and follow the prompts.

Specify the following items when defining a method that uses NMVOICE:

- The invocation string is drive:\path\NMVOICE
- NMVOICE supports both the TELL and ASK parameters of NMFIND. You can make a method that uses this program support TELL, ASK, or BOTH.

When you specify the ASK parameter, the contacted person can respond to the ASK message by pressing the appropriate key on the touch-tone keypad. The person can press 0 to choose from the following options: replay the message, hang up, or record a message for forwarding according to Notification Manager policy.

NMVOICE Method Parameters

Define the following parameters for any method you create that invokes NMVOICE. If a parameter is marked as required, you must define it as a parameter for the method. If a parameter is not required, you do not need to define it in the database. If you do not define the parameter, the default value is to be used whenever the method is invoked.

Note: You cannot set a value for a parameter at the entity or time-block level (that is, override the default value) if you do not define the parameter at the method level.

Parameter	Required
CallRetry	No
CallRings	No
CallWait	No
ChannelWait	No
Debug	No
Greeting	No
LeaveMessage	No
PassOnItemNumber	No
Phone	Yes
System	No
UsePIN	No
Wordlib	No

NMVOICE Parameter Descriptions

CallRetry

Specifies the number of times to retry to call a particular individual. The total number of send attempts will be one more than the CallRetry value.

Default: 4

CallRings

Specifies the number of times to let the phone ring on each call attempt.

Default: 8

CallWait

Specifies the number of seconds to wait between attempts to call a particular individual.

Default: 30

ChannelWait

Specifies the number of seconds to wait for a voice channel when attempting to call a particular individual.

Default: 60

Debug

Determines whether Notification Manager is to generate debugging messages at the command prompt or the AXCREXX window and in the ASOTRACE log. Possible values are YES and NO.

Default: NO

Greeting

Specifies the message to play when the phone is initially answered (before the TELL and ASK messages are played). The argument can be either a string or voice file. If the argument is a voice file, the name of the voice file must not contain any blanks and the last four characters must be .VOX.

Default: Notification Manager is calling about item number *nnnn*

LeaveMessage

Specifies if Notification Manager is to attempt to leave messages on answering machines. Valid values are:

YES

Notification Manager attempts to leave a message on an answering machine as many times as possible in the allotted 90 seconds.

NO

Notification Manager attempts to leave a message only once.

Default: YES

PassOnItemNumber

Specifies whether Notification Manager is to include the item number as part of the notification message. Valid values are:

YES

Include the item number as part of the message.

NO

Exclude the item number being played as part of the message.

For notification requests that have both TELL and ASK defined, or notification requests that reach an answering machine, the value of PassOnItemNumber is overridden and set to YES.

Default: NO

Phone

Specifies the phone number to call.

Note: This parameter must be specified at the method level. Simply specify a dummy phone number (for example, 555-1212) as the value at this level.

System

Specifies the name of the system that is running the notification server you want to perform the call. The *sysname* value can contain up to eight alphanumeric characters.

Default: The local system name

UsePIN

Specifies whether Notification Manager should restrict playing the notification message to just the intended contact. Valid values are:

YES

Play a message only after the user has entered a valid entity ID and password.

NO

Play a message without prompting the user to enter their entityID and password.

For notification requests with both the TELL and ASK parameters defined, the UsePIN value is always overridden and set to YES to insure system security, because ASK requires the user to take some type of action.

Default: YES

Wordlib

Specifies the Word Library to use when playing words from your TELL and ASK strings.

Default: VOXM_A

Chapter 11: Web Service API

This section describes the operations that you can perform through the CA Automation Point web service application programming interface. For an introduction to the capabilities of the web service API, see the section on "How You Interface with Third-party Software Applications" in the *CA Automation Point Product Guide*.

This section contains the following topics:

Fundamental Considerations

The AP web service API utilizes a REST (Representational State Transfer) architecture. This means that your client programs perform various operations by issuing the HTTP methods named GET, PUT, POST, and OPTIONS against a well-defined hierarchy of URIs (Universal Resource Identifiers) that represent CA Automation Point objects. These URIs are like the URIs that you type into your browser (similar to http://www.ca.com). AP web service URIs are specific to a given CA Automation Point server. These URIs are accessible only to that portion of your corporate network that can access your CA Automation Point server.

We have defined a set of URIs that represent meaningful CA Automation Point objects which you can manipulate. CA has also defined which HTTP methods can be issued against a given URI, and what operation each method actually performs against the related object.

The RESTful HTTP operations include an attached XML payload containing parameters that are associated with the desired operation. Similarly, the replies that you receive and your error results are delivered in XML documents. The XML reply document can be parsed to process detailed results from your request. Different XML documents apply to each specific HTTP method, issued against a specific CA Automation Point URI.

These concepts must be understood, to use the web service API effectively.

- Which URI represents the CA Automation Point object of interest to you?
- What operation does a given HTTP method perform?
- What XML document must you supply with your request?
- What XML document is received as a reply?

The remainder of this chapter is dedicated to answering exactly those questions.

Security Considerations

Delivering functionality through web services provides great flexibility to request an action from any computer in your corporate network. This functionality also presents a serious security concern. We require every request to contain a user ID and password. This requirement ensures that CA Automation Point only performs an action for an authorized program. For more information on specifying user ID requirements, see the documentation for each API request.

The user ID must be a valid user account for the type of action being requested. These user IDs are defined in CA Automation Point Remote Manager or defined as a login in Notification Manager. The context in which a user ID is confirmed is called an authentication realm. The HTTP standards do not require that an authentication realm be supplied by a client program when calling a web service. However, the client-side API presented by some programming languages require the caller to supply the authentication realm. One example is the Library for WWW in Perl – LWP. For all operations against notification objects the authentication realm is named *NotificationManager*. For all operations against sessions and messages, the authentication realm is named *RemoteManage*r.

A user ID and password are included in each request. Therefore, use TLS (Transport Layer Security, also known as SSL – Secure Sockets Layer) when running the web services in a production environment. This procedure ensures that the user ID and password are not accessible to an attacking program. Since you are communicating through HTTP, this means that you must use the HTTPS network scheme to secure your communications. For guidance on how to establish a TLS environment for CA Automation Point web services, see the web services section in the *CA Automation Point Administrator Guide*.

Error Replies

Typically each web service request has a separate XML reply document to reflect the unique response for that particular operation. However, when an error occurs, the same XML error document is returned for every web service request.

The HTTP Status code is a standard numeric indicator of the result of an HTTP method.

The status code is always available in an HTTP header of the reply to any request. The client application can check the HTTP Status code to determine the success or failure of the operation. When the request is successful, the applicable reply document is returned. When the request fails, an error document is returned.

The CA Automation Point web services follow the standard definitions for HTTP success and error codes. For information on special error code interpretation applicable to a given web service request, see the documentation for each API request.

If the HTTP status indicates a Method Not Allowed (405) error, the document that is returned by the CA Automation Point web service request is an XML document named AllowedMethods. That XML document is defined by the AllowedMethods.xsd schema.

If the HTTP status indicates an error other than Method Not Allowed, the document returned by the CA Automation Point web service request is an XML document named WSResult. That XML document is defined by the WSResult.xsd schema. The WSResult document contains detailed error codes and descriptive text about the cause of the error.

An exception to this behavior is the HTTP HEAD method. The HTTP specification states that the HEAD method only returns HTTP headers and must not return a body. Thus, HEAD will not return a WSResult document after an error. Use the HTTP status value in the header to detect an error. A HEAD operation can be issued against any CA Automation Point URI that accepts a GET operation. Because it returns no body, HEAD has limited value and thus is not documented in this guide. One possible use of HEAD would be to implement a heartbeat check of an CA Automation Point server. A heartbeat check only requires an indication of success or failure.

URIs, HTTP Methods, and XML Documents

A URI represents a CA Automation Point resource upon which operations are performed. The meaning of an HTTP method is determined by the specified URI. The API documentation is organized alphabetically by each resource URI and HTTP method. These methods accomplish each CA Automation Point operation, and identify the XML documents that are required or delivered by that operation.

These methods are documented in:

- Notification (see page 402)
- Internal Sessions (see page 393)
- Customer Defined Sessions (see page 410)

The base name of a URI targeted to a CA Automation Point web service running under Tomcat on the local machine using TLS would be:

https://localhost:8443/apwebsvc

This example base name is used for all URIs listed in the upcoming sections. In practice, a specific host name is used by a client application running on another computer. Further, a site can choose to configure Tomcat to use a non-default port number. Finally, during initial testing, a site may choose to avoid TLS for simplicity. That site would then use the http scheme and the non-TLS default Tomcat port of 8080. Utilize your site-specific values for any URI in upcoming examples. The remainder of the URI reflects the CA Automation Point resource of interest.

URIs can contain characters that are not normally allowed in a URI. Such special characters must be percent-encoded as per the Internet Engineering Task Force RFC 3986. Some CA Automation Point XML reply documents contain URIs. These URIs are not percent-encoded. If such a URI is used as the target URI for another web service request, it must be percent-encoded before making that second request. Direct programmed calls to the web service API, must be percent-encode such that a URI before using it as the target URI of another request. The UTF-8 character set must be used to percent-encode the URIs sent to CA Automation Point web services.

The CA RequestService client-side API and command-line program automatically percent-encode any supplied URI. Thus, when using those CA client components, customer programs must not percent-encode the URIs supplied to those CA client components.

When issuing a web service request from the command line using the CA RequestService client application, be aware of special characters that your operating system recognizes. These special characters are likely introduced when specifying a query, which is expressed as part of the URI. When the URI contains such special characters, enclose this URI in double quotes. The command interpreter for your operating system does not interpret special characters when enclosed in double quotes. For example, when requesting a list of all sessions that are automated, you issue a GET request with a query:

https://localhost:8443/apwebsvc/sessions?Automated=yes

On a Windows operating system, the equals sign is interpreted by the Windows Command Interpreter. The syntax is not interpreted as a single URI parameter, and the command fails. To prevent Windows from misinterpreting this parameter, enclose the parameter in double quotes:

"https://localhost:8443/apwebsvc/sessions?Automated=yes"

As each XML document is identified in one of the described methods, the XML schema that defines that document is specified. The XML schemas for all CA Automation Point web service XML documents are contained in:

%ap_home%\distrib\websvc*.xsd

Documentation for all AP web service XML schemas is contained in:

%ap_home%\Doc\help\websvc\xmlSchemas*.html

These XML documents explicitly identify the character set used to encode international characters within the XML document. The 'encoding' attribute of the XML 'document type declaration' is used to specify the desired character set. Data within the XML document must be binary-encoded in the character set specified by the encoding attribute, or incorrect data is transmitted to the CA Automation Point server.

All CA Automation Point sample XML documents for Windows and Linux specify an encoding value of UTF-8. The samples for z/OS specify an encoding value of IBM1047. When CA Automation Point web service components move an XML document into a Java String or transmit the document over a TCP/IP connection, they re-encode the data for the appropriate character set. When these components re-encode the XML data, they also alter the XML encoding attribute of the document to reflect the character set being used.

For example, when the RequestService command-line client outputs an XML reply document to standard output, it uses the default character set for the operating system. On the Windows operating system, the default character set is code page 1252. Thus, when RequestService displays an XML document on standard output on a Windows computer, the data is encoded with code page 1252 and the XML encoding attribute contains the value 'encoding=windows-1252'.

One final consideration is that the CA Automation Point Desktop server application processes its data in the Windows OEM code page. XML input documents can be encoded with UTF-8 or any other character set supported by the Java virtual machine. The actual character must exist in the OEM code page of the Windows server computer that is running CA Automation Point. For example, not all characters in the UTF-8 character set can be mapped to an identical representation in the commonly used OEM code page 850. When such a character cannot be represented in the OEM code page, the character is typically displayed as an unexpected graphic character.

In XML documents, CA Automation Point web services utilize namespaces to avoid naming conflicts. Set the CA Automation Point web service namespace as the default namespace in an attribute of the root element. Setting this attribute allows all CA Automation Point elements to not require qualifiers in this document. Each CA Automation Point XML document is validated against the XML schema, which defines that document. Specify the name of the schema as an attribute in the root element.

Example General Format:

```
<?xml version="1.0" encoding="utf-8"?>
<YourXMLDocumentRootElement

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.ca.com/distauto/ap/websvc/msgData  XmlSchemaName.xsd"
    xmlns="http://www.ca.com/distauto/ap/websvc/msgData">

<ContentOfYourXMLDocument>
</YourXMLDocumentRootElement>
```

Note: Only the XML schema name (bolded) changes from one document to another.

Example Specific Case:

This example sends a simple notification.

```
<?xml version="1.0" encoding="utf-8"?>
<NotificationRequest

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.ca.com/distauto/ap/websvc/msgData  NotificationRequest.xsd"
    xmlns="http://www.ca.com/distauto/ap/websvc/msgData">

<Name>PersonThatYouWantToNotifiy</Name>
<Tell>Hi. This is a notification test.</Tell>
</NotificationRequest>
```

Query String in URIs

URI can contain a query string. RFC 3986 titled “Uniform Resource Identifier (URI): Generic Syntax” describes a query string as follows:

> *"The query component contains non-hierarchical data that, along with data in the path component (Section 3.3), serves to identify a resource within the scope of the URI's scheme and naming authority (if any). The query component is indicated by the first question mark ("?") character and terminated by a number sign ("#") character or by the end of the URI … query components are often used to carry identifying information in the form of "key=value" pairs."*

The CA Automation Point URIs for which our web services accept a query string are documented in this chapter. CA follows the guidance of RFC 3986. You can often apply multiple query options when specifying a URI that accepts a query string. When specifying multiple query options, you separate each option from the previous option with an ampersand (&) character. This format is shown in the following example:

```
URI-path?option1=value1&option2=value2
```

When an unknown query option is specified, the operation returns an HTTP status error of 400 (Bad Request) and the operation returns a WSResult document. If an otherwise valid query option is specified, which is not applicable due to a conflict with another query option, the non-applicable option is ignored and the operation is processed. These conflicts are documented in this chapter.

Within some query strings, a pattern-matching expression can be used as the value of a query option. The pattern-matching expression format that CA Automation Point recognizes is:

Character	Meaning
.	Matches any one character
*	Matches any zero or more characters

The ‘\’ character serves as an escape character. The following escape sequences are recognized:

Character sequence	Meaning
\.	Matches the ‘.’ character.
*	Matches the ‘*’ character.
\\	Matches the ‘\’ character.

Any other escape sequence is treated as an error. For example, the string “AX\C” is an invalid pattern.

Internal Sessions

These URIs and operations relate to AP internal sessions, which are not defined by customers. These sessions are automatically created by CA Automation Point to surface important messages regarding the operation of the product.

URI https://localhost:8443/apwebsvc/intsessions

This URI represents the set of AP internal sessions.

GET intsessions

Get a list of defined AP internal sessions.

Query Options

The following HTTP query strings can be added to the URI to control the volume of returned URIs. Most query options represent properties of the object that are returned in the reply document. To see the definitions of those properties, you can read the schema documentation for the reply document.

SessionName=

The internal session name. A pattern-matching expression can be used to match session names.

Count=

Maximum number of items the request returns. The highest value that can be set for this option is 100000. If this option is not specified a default value of 10000 is used.

FromCount=

The numeric count within your selected query from which sessions are included in your reply. The first position is numbered 1. Each successive position is incremented by one. The FromCount option can be used to retrieve a few sessions at a time, repeatedly.

Example:

This example shows retrieving sessions in groups of 10, using a sequence.

get .../intsessions?Count=10
get .../intsessions?Count=10&FromCount=11
get .../intsessions?Count=10&FromCount=21
and so on.

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer *VIEW* privilege on the internal sessions to be returned. Any internal session that the user does not have at least *VIEW* privilege, is not returned in the list.

HTTP Response Headers

Status

Successful result = 200 = Ok

Successful Reply Document

InternalSessionList in schema InternalSessionList.xsd

The reply contains a list of internal session names for defined/enabled internal sessions. This reply includes the URIs used to access their details, for example:

https://localhost:8443/apwebsvc/intsessions/AXC, https://localhost:8443/apwebsvc/intsessions/VOX, and so on.

OPTIONS intsessions

Obtains a list of HTTP methods that can be issued against this URI.

Query Options

None

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID can be any valid user account on the AP server that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 200 = Ok

Allow

List of allowed methods. This header contains OPTIONS, GET, HEAD.

Successful Reply Document

AllowedMethods in schema AllowedMethods.xsd

This reply contains the same methods as listed in the HTTP *Allow* header.

URI https://localhost:8443/apwebsvc/intsessions/<intsessName>

This URI represents the specific AP internal session that is identified by its name.

GET intsessions/<intsessName>

Get the properties of a specific internal session.

Query Options

None

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer *VIEW* privilege on the internal session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 200 = Ok

Successful Reply Document

InternalSession in schema InternalSession.xsd

The reply returns the properties of the specified internal session.

OPTIONS intsessions/<intsessName>

Obtains a list of HTTP methods that can be issued against this URI.

Query Options

None

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer *VIEW* privilege on the internal session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 200 = Ok

Allow

List of allowed methods. This header contains OPTIONS, GET, HEAD.

Successful Reply Document

AllowedMethods in schema AllowedMethods.xsd

This reply contains the same methods as listed in the HTTP *Allow* header.

URI https://localhost:8443/apwebsvc/intsessions/<intsessName>/messages

This URI represents the messages from this AP internal session that are processed by the AP rules engine.

GET intsessions/<intsessName>/messages

Get a list of current messages processed by CA Automation Point. Message IDs are returned in order from the oldest message to the latest message, unless the request is altered by one or more query options.

Query Options

These HTTP query strings can be added to the URI to control the volume of returned URIs. Most query options represent properties of the object that are returned in the reply document. To see the definitions of those properties, you can read the schema documentation for the reply document.

Text=

The text of the message. A pattern-matching expression can be used to match message texts.

JobId=

Job ID associated with a message. A pattern-matching expression can be used to match Job ID.

AppName=

The application name that is associated with a message. A pattern-matching expression can be used to match application names.

Hostname=

The name of the host where the message originated from. A pattern-matching expression can be used to match host names.

Category=

The category that is associated with a message. A pattern-matching expression can be used to match categories.

Severity=

The severity that is associated with a message. A pattern-matching expression can be used to match severities.

ActionMessage=

Specifies whether normal or action messages are retrieved. Value is yes or no.

FromProcessTime=

A timestamp specifying the time in seconds elapsed since midnight Coordinated Universal Time (UTC), January 1, 1970. Only messages that are processed by CA Automation Point at the specified time or later are returned. An error occurs when specifying a value that is larger than the value of the ToProcessTime query option, when the ToProcess Time option is specified.

ToProcessTime=

A timestamp specifying the time in seconds elapsed since midnight Coordinated Universal Time (UTC), January 1, 1970. Only messages that are processed by CA Automation Point at the specified time or sooner are returned. An error occurs when specifying a value that is smaller than the value of the FromProcessTime query option, when the FromProcessTime is specified..

AfterId=

Specifies a message ID after which you want to retrieve messages. Only messages that occurred after AfterId are returned. The last message ID returned in a previous GET request can be used as the AfterId in your next GET request.

If the special value 0 is specified for AfterId, the GET method returns the last (most recent) message ID. Since only one message ID can be returned in this case, a *Count* option specified in the same request is ignored. Similarly a FromCount option specified in the same request is also ignored. When using a non-zero value for AfterId, both Count and FromCount are applied to the request.

AfterId can specify a message ID from a different session than the session which you attempt to query. Thus, you could retrieve messages that arrive in one session after a message which arrived in another session.

Count=

Maximum number of message IDs to be returned in this request. The highest value that can be set for this option is 100000. When this option is not specified, a default value of 10000 is used. When fewer messages than Count are available, the count value returned in the reply document specifies the actual number of message IDs returned.

FromCount=

The numeric count within your selected query from which sessions are included in your reply. The first position is numbered 1. Each successive position is incremented by one. The FromCount option can be used to retrieve a few sessions at a time, repeatedly.

Example:

This example shows retrieving messages in groups of 10, using a sequence.

```
get .../messages?Count=10
get .../messages?Count=10&FromCount=11
get .../messages?Count=10&FromCount=21
and so on.
```

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer *VIEW* privilege on the internal session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 200 = Ok

Successful Reply Document

MessageList in schema MessageList.xsd

The reply contains a list of identifiers for existing messages. This reply includes the URIs used to access those messages, for example.

https://localhost:8443/apwebsvc/intsessions/<intsessName>/messages/messageId1,
https://localhost:8443/apwebsvc/intsessions/<intsessName>/messages/messageId2,
and so on.

OPTIONS intsessions/<intsessName>/messages

Obtains a list of HTTP methods that can be issued against this URI.

Query Options

None

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer *VIEW* privilege on the internal session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 200 = Ok

Allow

List of allowed methods. This header contains OPTIONS, GET, HEAD, and POST when the user has the *FULL* privilege on the internal session that is the target of this HTTP method.

Successful Reply Document

AllowedMethods in schema AllowedMethods.xsd

This reply contains the same methods as listed in the HTTP *Allow* header.

POST intsessions/<intsessName>/messages

Submit a message into the AP rules engine treating the message as if it came from the associated AP internal session.

Query Options

None

Request Document

Message in schema Message.xsd

The request contains properties of the message. For example, message text, application name, host name, category, severity, and so on. The ProcessTime property is automatically populated by CA Automation Point and can be omitted in the request document.

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer *FULL* privilege on the internal session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 201 = Created

Location

Contains the URI to the newly created message.

Successful Reply Document

MessageReply in schema Message.xsd

The reply contains the URI which identifies the created message. That URI is also returned in the HTTP *Location* header.

URI https://localhost:8443/apwebsvc/intsessions/<intsessName>/messages/<messageId>

This URI represents a specific message that is identified by the message ID.

GET intsessions/<intsessName>/messages/<messageID>

Get the properties of a specific message.

Query Options

None

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer *VIEW* privilege on the internal session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 200 = Ok

Successful Reply Document

Message in schema Message.xsd

The reply contains the properties of the specified message.

OPTIONS intsessions/<intsessName>/messages/<messageID>

Obtains a list of HTTP methods that can be issued against this URI.

Query Options

None

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password, in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer *VIEW* privilege on the internal session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 200 = Ok

Allow

List of allowed methods. This header contains OPTIONS, GET, HEAD and PUT when the user has the *FULL* privilege on the internal session that is the target of this HTTP method.

Successful Reply Document

AllowedMethods in schema AllowedMethods.xsd

This reply contains the same methods as listed in the HTTP *Allow* header.

PUT intsessions/<intsessName>/messages/<messageID>

Change properties of a specific message

Query Options

None

Request Document

Message in schema Message.xsd

Currently the PUT method only allows changing the ActionMessage element value from yes to no, which removes the message from the AP action message window. Any other changes that are contained in the input document are ignored.

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer *FULL* privilege on the internal session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 200 = Ok

Successful Reply Document

None

Notification URIs

These URIs and operations relate to notifications that are processed by the CA Automation Point Notification Manager.

URI https://localhost:8443/apwebsvc/notifications

This URI represents the notifications that are being (or have been) processed by Notification Manager.

GET notifications

Get a list of AP notifications. Notification IDs are returned in order from the oldest notification to the latest notification, unless the request is altered by one or more query options.

Query Options

These HTTP query strings can be added to the URI to control the volume and order of returned URIs. Most query options represent properties of the object that are returned in the reply document. To see the definitions of those properties, you can read the schema documentation for the reply document.

AnswerableBy=

An NM Contact name that is allowed to answer the notification.

IssuedBy=

Name of the NM login that issued the notification.

Status=

Can specify one of these statuses for notification: Initializing, Sending, Sent, AwaitingResponse, NoResponse, Responded, LateResponse, or Failed.

FromTime=

A timestamp specifying the time in seconds elapsed since midnight Coordinated Universal Time (UTC), January 1, 1970. Only notifications that occurred at the specified time or later are returned. An error occurs when specifying a value that is larger than the value of the ToTime query option, when FromTime and ToTime are specified.

ToTime=

A timestamp specifying the time in seconds elapsed since midnight Coordinated Universal Time (UTC), January 1, 1970. Only notifications that occurred at the specified time or sooner are returned. An error occurs when specifying a value that is smaller than the value of the FromTime query option, when FromTime and ToTime are specified.

AfterId=

Specifies a notification ID after which you want to retrieve notifications. Only notifications that occurred after AfterId are returned. The last notification ID returned in a previous GET request can be used as the AfterId in your next GET request.

If the special value 0 is specified for AfterId, the GET method returns the last (most recent) notification ID. Since only one notification ID can be returned in this case, a *Count* option specified in the same request is ignored.

Count=

Maximum number of notification IDs to be returned in this request. The highest value that can be set for this option is 100000. When this option is not specified, a default value of 10000 is used. When fewer notifications than Count are available, the Count value returned in the reply document specifies the actual number of notifications returned.

Example1:

This pseudo code continuously retrieves notifications that arrive after your first GET operation. This code also limits the number of notifications that are retrieved to 10 at a time.

```
LastIdOfOurPreviousGet = 0
while ( YouWantToContinue )
{
   GET .../notifications?AfterId=LastIdOfOurPreviousGet&Count=10
   If  ( our reply contains notification IDs )
   {
      Do something with the retrieved notification IDs
      LastIdOfOurPreviousGet = last ID from our reply
   }
   Sleep a little while
}
```

Example2:

This pseudo code gets all notifications that occurred starting at 9:00 AM this morning and stop when the last notification is reached.

```
GET .../notifications?FromTime=TimeFor9:00AMToday&Count=10
While ( the number retrieved > 0 )
{
   Do something with the retrieved notification IDs
   LastIdOfOurPreviousGet = last ID from our reply
   If ( number retrieved < 10 )
      break
   GET .../notifications?AfterId=LastIdOfOurPreviousGet&Count=10
}
```

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be an NM Login.

HTTP Response Headers

Status

Successful result = 200 = Ok

Successful Reply Document

NotificationList in schema NotificationList.xsd

The reply contains a list of identifiers for existing notification requests. This reply includes the URIs used to access those notifications, for example:

http://localhost:8080/apwebsvc/notifications/notificationId1,
http://localhost:8080/apwebsvc/notifications/notificationId2,
and so on.

When the user has the View All Notifications privilege, the list contains every notification. Otherwise the user retrieves only those notifications for which the user was directly involved with the notification. Specifically, the user sent the notification, the user was the intended recipient, or the user was notified during the escalation process.

OPTIONS notifications

OPTIONS notifications

Obtains a list of HTTP methods that can be issued against this URI.

Query Options

None

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be an NM Login.

HTTP Response Headers

Status

Successful result = 200 = Ok

Allow

List of allowed methods. Contains OPTIONS, GET, HEAD, and POST when the user has the Notify All Contacts privilege.

Successful Reply Document

AllowedMethods in schema AllowedMethods.xsd

This reply contains the same methods as listed in the HTTP *Allow* header.

POST notifications

Send a notification.

Query Options

None

Request Document

NotificationRequest in schema NotificationRequest.xsd

The request contains characteristics of the notification, such as who is notified and what is NM to tell that person.

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be an NM Login with the *Notify All contacts* privilege on the AP server that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 202 = Accepted

Location

Contains the URI to the newly created notification.

Successful Reply Document

NotificationRequestReply in schema NotificationRequest.xsd

The reply contains the URI which identifies the created notification. That URI is also returned in the HTTP *Locat*ion header.

URI https://localhost:8443/apwebsvc/notifications/<notificationId>

This URI represents a specific notification that is identified by its ID number.

GET notifications/<notificationId>

Get the properties of a specific notification.

Query Options

None

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be an NM Login.

HTTP Response Headers

Status

Successful result = 200 = Ok

Successful Reply Document

Notification in schema Notification.xsd

The reply contains properties of the notification, such as Tell text, IntendedRecipient, SenderLogin, and Status.

OPTIONS notifications/<notificationId>

Obtains a list of HTTP methods that can be issued against this URI.

Query Options

None

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be an NM Login.

HTTP Response Headers

Status

Successful result = 200 = Ok

Allow

List of allowed methods. This header contains OPTIONS, GET, HEAD.

Successful Reply Document

AllowedMethods in schema AllowedMethods.xsd

This reply contains the same methods as listed in the HTTP Allow header.

URI https://localhost:8443/apwebsvc/notifications/<notificationId>/answer

This URI represents the answer for the notification that is identified by its ID number.

When the notification did not ask a question, the associated *answer* URI does not exist.

GET notifications/<notificationId>/answer

Get the properties of the answer to a notification.

Query Options

None

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be an NM Login. The user also must have the *Answer All Notifications* privilege or an associated contact for that user must have been notified as part of the notification.

HTTP Response Headers

Status

Successful result = 200 = Ok

A result of 404 (Not Found) is returned when the associated notification did not ask a question and thus cannot have an answer.

Successful Reply Document

NotificationAnswer in schema NotificationAnswer.xsd

The reply contains the numeric choice that was made to answer the notification. The valid numeric answers range from 1 to 9. When the notification has not yet been answered, the numeric answer 0 is returned. The AnswerText associated with the numeric choice is also returned.

OPTIONS notifications/<notificationId>/answer

Obtains a list of HTTP methods that can be issued against this URI.

Query Options

None

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be an NM Login. The user also must have either the *Answer All Notifications* privilege or an associated contact for that user must have been notified as part of the notification.

HTTP Response Headers

Status

Successful result = 200 = Ok

A result of 404 (Not Found) is returned when the associated notification did not ask a question and thus cannot have an answer.

Allow

List of allowed methods. This header contains OPTIONS, GET, HEAD, and PUT when the user is authorized to answer the notification and it was not answered yet.

Successful Reply Document

AllowedMethods in schema AllowedMethods.xsd

This reply contains the same methods as listed in the HTTP *Allow* header.

PUT notifications/<notificationId>/answer

Answers a specific notification.

Query Options

None

Request Document

NotificationAnswer in schema NotificationAnswer.xsd

The document contains the numeric choice that identifies the answer for the specified notification in the AnswerNumber attribute. The allowed values range from 1 to 9. The value of the Answer element is ignored and can be empty.

HTTP Request Headers

Authorization

Supply a user ID and password, in the HTTP Basic Authentication format. The user ID must be an NM Login. The user also must have either the *Answer All Notifications* privilege or an associated contact for that user must have been notified as part of the notification.

HTTP Response Headers

Status

Successful result = 200 = Ok

A result of 404 (Not Found) is returned when the associated notification did not ask a question and thus cannot have an answer.

A result of 405 (Method Not Allowed) is returned when someone else already answered the notification.

A result of 409 (Conflict) is returned when someone else answered the notification during the processing of this PUT request.

Allow

When a PUT is performed on a notification and the notification has already been answered, a 405 error is returned. This header contains OPTIONS, GET, HEAD.

Successful Reply Document

None

Customer Defined Sessions

These URIs and operations relate to AP sessions that are defined by customers and typically monitor a remote host.

URI https://localhost:8443/apwebsvc/sessions

This URI represents the set of AP sessions that are defined to monitor hosts.

GET sessions

Get a list of defined AP session names.

Query Options

These HTTP query strings can be added to the URI to control the volume of returned URIs. Most query options represent properties of the object that are returned in the reply document. To see the definitions of those properties, you can read the schema documentation for the reply document.

SessionName=

The AP session name. A pattern-matching expression can be used to match session names.

SystemName=

The AP system name field. A pattern-matching expression can be used to match system names.

ConsoleType=

The type of console that is used by the session. The valid values are:

Valid Values	
ASYNCH	SYSPLEX
Default	TANDEM
DTX	TANDEMALL
INIT	TPF3270
iSeries	TPFASYNCH
JES3	VAX
JES3MCS	VAXALL
MCS	VM
RCS	VSE

Terminal=

The terminal used by the session. The valid values are:

Valid Values		
3278	3279_3	AXC
3278_2	3279_4	VIO
3278_3	3279_5	VT52

Valid Values		
3278_4	3477	VT100
3278_5	5292	VT320
3279	6530	VT420
3279_2	ASYNCH	

Automated=

Indicates the automation status. Valid values are:

yes

The session is currently being automated.

paused

Automation on the session has been explicitly paused.

no

The session is not automated by configuration and cannot be dynamically automated.

Connected=

Indicates the state of the connection to the monitored host. Valid values are: yes and no.

Count=

Maximum number of session IDs to be returned in this request. The highest value that can be set for this option is 100000. If this option is not specified a default value of 10000 is used.

FromCount=

The numeric count within your selected query from which sessions are included in your reply. The first position is numbered 1. Each successive position is incremented by one. The FromCount option can be used to retrieve a few sessions at a time, repeatedly.

Example:

This example shows retrieving sessions in groups of 10, using a sequence.

```
get .../sessions?Count=10
get .../sessions?Count=10&FromCount=11
get .../sessions?Count=10&FromCount=21
and so on.
```

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer *VIEW* privilege on the sessions to be returned. Any session for which the user does not have at least *VIEW* privilege, is not returned in the list.

HTTP Response Headers

Status

Successful result = 200 = Ok

Successful Reply Document

SessionList in schema SessionList.xsd

This reply contains a list of customer-assigned session names for existing sessions. This reply includes the URIs used to access those sessions, for example:

https://localhost:8443/apwebsvc/sessions/sessionName1,
https://localhost:8443/apwebsvc/sessions/sessionName2, and so on.

OPTIONS sessions

Obtains a list of HTTP methods that can be issued against this URI.

Query Options

None

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID can be any valid user account on the AP server that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 200 = Ok

Allow

List of allowed methods. This header contains OPTIONS, GET, HEAD.

Successful Reply Document

AllowedMethods in schema AllowedMethods.xsd

This reply contains the same methods as listed in the HTTP Allow header.

URI https://localhost:8443/apwebsvc/sessions/<sessionName>

This URI represents the specific AP session that is identified by session name.

GET sessions/<sessionName>

Get the properties of a specific session.

Query Options

None

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer *VIEW* privilege on the session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 200 = Ok

Successful Reply Document

Session in schema Session.xsd

The reply returns the properties of the specified session, for example, SessionName, ConsoleType, and Automated.

A DeviceDetails property contains a URI to another resource. This resource contains details that are specific to the type of device that is used to make the connection to the monitored host. The URI points to an additional subnode of the AP URI hierarchy underneath the current sessionName node. The new subnode has one of the following three values:

…/apwebsvc/sessions/<sessionName>/TN3270
…/apwebsvc/sessions/<sessionName>/TN5250
…/apwebsvc/sessions/<sessionName>/Asynchronous

A Type attribute supplies information about the type of connection. A GET operation against such a subnode URI returns different properties which are characteristic of the type of connection. Details about the properties of each of these URIs are described later in this document.

OPTIONS sessions/<sessionName>

Obtains a list of HTTP methods that can be issued against this URI.

Query Options

None

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer *VIEW* privilege on the session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 200 = Ok

Allow

List of allowed methods. This header contains OPTIONS, GET, HEAD. It also contains PUT and POST when the user has the *FULL* privilege on the session that is the target of this HTTP method.

Successful Reply Document

AllowedMethods in schema AllowedMethods.xsd

This replay contains the same methods as listed in the HTTP *Allow* header.

POST sessions/<sessionName>

Execute a command in a specific session.

This operation is only valid for sessions that are connected to *live*, interactive sessions. When this command is issued into an artificial AP session, for example, Event Traffic Control sessions, there is no meaning. A POST to such a session returns the HTTP status 405 (Method not allowed).

Query Options

None

Request Document

SessionCommand in schema SessionCommand.xsd

The request contains the command that is run in the session that is the target of this HTTP method. The command text can contain the set of AP @ key abbreviations. For more information, see the key abbreviation table in the appendix on customizing special CA Automation Point files in the *CA Automation Point Administrator Guide*. No '@E' key is required at the end of the command.

By default, the command is synchronous. This means that the API waits until any X-state on a session is cleared and the command has been transmitted to the monitored host.

Within the Command document, optionally the command can be processed asynchronously.

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer *FULL* privilege on the session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result for a synchronous command = 200 = Ok

The success of a synchronous command means that the command was transmitted to the monitored host.

Success for an asynchronous command = 202 = Accepted.

The success of an asynchronous command means that the command was accepted by the AP server for later transmission to the monitored host.

Successful Reply Document

None

A command is simply executed. CA Automation Point cannot access any properties of a command after it has been executed.

PUT sessions/<sessionName>

Change the properties of a specific session.

Query Options

These HTTP query strings can be added to the URI to control the order in which the desired changes are made.

Order=

The property names *Automated* and *Connected* are listed in the order in which their values will be changed. The property names are separated by commas. Both property names must be specified, or the PUT method returns an error. When an Order option is not specified, the order the properties are changed follows this specified query string:

PUT …/<sessionName>?Order=Automated,Connected

Request Document

Session in schema Session.xsd

The request contains the requested state for this specified session. Currently only the Connected and Automated session properties can be modified. Any other differences between the current state and the data in the request document are ignored. When the current Automated state is *no* then it cannot be changed because the session is not configured to be automated. In addition, the Automated state cannot be changed from *yes* or *paused* to *no*.

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer *FULL* privilege on the session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 200 = Ok

A successful status is only returned when **all** changes are successful. Otherwise, an error is returned and error details are provided in a WSResult document. A subsequent GET operation can also be performed on the session to identify properties that were not changed to the desired values.

Successful Reply Document

None

URI https://localhost:8443/apwebsvc/sessions/<sessionName>/Asynchronous

This URI represents the asynchronous communication properties of the specified session.

Do not construct this URI in an application. Only use this URI to retrieve the value of the DeviceDetails property that is returned by the GET method on a:

…/apwebsvc/sessions/<sessionName> URI

Only one of the following URI subnodes are valid for any given session.

../<sessionName>/Asynchronous,
../<sessionName>/TN3270, or
../<sessionName>/TN5250

GET sessions/<sessionName>/Asynchronous

Get the asynchronous properties of the named session.

Query Options

None

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer *VIEW* privilege on the session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 200 = Ok

Successful Reply Document

SessionAsynchronous in schema SessionAsynchronous.xsd

The reply contains properties such as Hostname, Port, and CommunicationDevice which identifies a COM port, telnet protocol, SSH protocol, or in-memory AP session.

OPTIONS sessions/<sessionName>/Asynchronous

Obtains a list of HTTP methods that can be issued against this URI.

Query Options

None

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer *VIEW* privilege on the session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 200 = Ok

A result of 404 (Not Found) is returned when the associated session is not an asynchronous session.

Allow

List of allowed methods. This header contains OPTIONS, GET, HEAD.

Successful Reply Document

AllowedMethods in schema AllowedMethods.xsd

This reply contains the same methods as listed in the HTTP *Allow* header.

URI https://localhost:8443/apwebsvc/sessions/<sessionName>/messages

This URI represents the messages from this AP session that are processed by the AP rules engine.

GET sessions/<sessionName>/messages

Get a list of current messages processed by CA Automation Point. Message IDs are returned in order from the oldest message to the latest message, unless the request is altered by one or more query options.

Query Options

These HTTP query strings can be added to the URI to control the volume of returned URIs. Most query options represent properties of the object that are returned in the reply document. To see the definitions of those properties, you can read the schema documentation for the reply document.

Text=

The text of the message. A pattern-matching expression can be used to match message texts.

JobId=

Job ID associated with a message. A pattern-matching expression can be used to match Job ID.

AppName=

The application name that is associated with a message. A pattern-matching expression can be used to match application names.

Hostname=

The name of the host where the message originated from. A pattern-matching expression can be used to match host names.

Category=

The category that is associated with a message. A pattern-matching expression can be used to match categories.

Severity=

The severity that is associated with a message. A pattern-matching expression can be used to match severities.

ActionMessage=

Specifies whether normal or action messages are retrieved.Value is yes or no.

FromProcessTime=

A timestamp specifying the time in seconds elapsed since midnight Coordinated Universal Time (UTC), January 1, 1970. Only messages processed by Automation Point at the specified time or later are returned.An error occurs when specifying a value that is larger than the value of the ToProcessTime query option, when FromProcessTime and ToProcessTime are specified.

ToProcessTime=

A timestamp specifying the time in seconds elapsed since midnight Coordinated Universal Time (UTC), January 1, 1970. Only messages processed by Automation Point at the specified time or sooner are returned. An error occurs when specifying a value that is smaller than the value of the FromProcessTime query option, when FromProcessTime and ToProcessTime are specified.

AfterId=

Specifies a message ID after which you want to retrieve messages. Only messages that occurred after AfterId are returned. The last message ID returned in a previous GET request can be used as the AfterId in your next GET request.

When the special value 0 is specified for AfterId, the GET method returns the last (most recent) message ID. Since only one message ID can be returned in this case, a *Count* option specified in the same request is ignored. Similarly a FromCount option specified in the same request is also ignored. When using a non-zero value for AfterId, both Count and FromCount are applied to the request.

AfterId can specify a message ID from a different session than the session which you attempt to query. Thus, you could retrieve messages that arrive in one session after a message which arrived in another session.

Count=

Maximum number of message IDs to be returned in this request. The highest value that can be set for this option is 100000. If this option is not specified a default value of 10000 is used. When fewer messages than Count are available, the count value returned in the reply document specifies the actual number of message IDs returned.

FromCount=

The numeric count within your selected query from which sessions are included in your reply. The first position is numbered 1. Each successive position is incremented by one. The FromCount option can be used to retrieve a few sessions at a time, repeatedly.

Example:

This example retrieves messages in groups of 10, using a sequence.

```
get .../messages?Count=10
get .../messages?Count=10&FromCount=11
get .../messages?Count=10&FromCount=21
and so on.
```

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer VIEW privilege on the session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 200 = Ok

Successful Reply Document

MessageList in schema MessageList.xsd

The reply contains a list of identifiers for existing messages. This reply includes the URIs used to access those messages, for example:

https://localhost:8443/apwebsvc/sessions/<sessionName>/messages/messageId1,
https://localhost:8443/apwebsvc/sessions/<sessionName>/messages/messageId2,
and so on.

OPTIONS sessions/<sessionName>/messages

Obtains a list of HTTP methods that can be issued against this URI.

Query Options

None

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer *VIEW* privilege on the session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 200 = Ok

Allow

List of allowed methods. This header contains OPTIONS, GET, HEAD, and POST when the user has the *FULL* privilege on the session that is the target of this HTTP method.

Successful Reply Document

AllowedMethods in schema AllowedMethods.xsd

This reply contains the same methods as listed in the HTTP *Allow* header.

POST sessions/<sessionName>/messages

Submit a message into the AP rules engine treating the message as if it came from the associated AP session.

Query Options

None

Request Document

Message in schema Message.xsd

The request contains properties of the message, for example, message text, application name, host name, category, and severity. The ProcessTime property is automatically populated by CA Automation Point and can be omitted in the request document.

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer *FULL* privilege on the session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 201 = Created

Location

Contains the URI to the newly created message

Successful Reply Document

MessageReply in schema Message.xsd

The reply contains the URI which identifies the created message. That URI is also returned in the H*TTP Loca*tion header.

URI https://localhost:8443/apwebsvc/sessions/<sessionName>/messages/<messageId>

This URI represents a specific message that is identified by the message ID.

GET sessions/<sessionName>/messages/<messageID>

Get the properties of a specific message.

Query Options

None

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer VIEW privilege on the session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 200 = Ok

Successful Reply Document

Message in schema Message.xsd

The reply contains the properties of the specified message.

OPTIONS sessions/<sessionName>/messages/<messageID>

Obtains a list of HTTP methods that can be issued against this URI.

Query Options

None

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password, in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer *VIEW* privilege on the session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 200 = Ok

Allow

List of allowed methods. This header contains OPTIONS, GET, HEAD, and PUT when the user has the *FULL* privilege on the session that is the target of this HTTP method.

Successful Reply Document

AllowedMethods in schema AllowedMethods.xsd

This reply contains the same methods as listed in the HTTP *Allow* header.

PUT sessions/<sessionName>/messages/<messageID>

Change properties of a specific message

Query Options

None

Request Document

Message in schema Message.xsd

Currently the PUT method only allows changing the ActionMessage element value from *yes* to *no*, which removes the message from the AP action message window. Any other changes that are contained in the input document are ignored.

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer *FULL* privilege on the session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 200 = Ok

Successful Reply Document

None

URI https://localhost:8443/apwebsvc/sessions/<sessionName>/TN3270

This URI represents the 3270 communication properties of the specified session.

Do not construct this URI in an application. Only use this URI to retrieve the value of the DeviceDetails property that is returned by the GET method on a:

…/apwebsvc/sessions/<sessionName> URI

Only one of the following URI subnodes

../<sessionName>/Asynchronous,
../<sessionName>/TN3270, or
../<sessionName>/TN5250

are valided for any given session.

GET sessions/<sessionName>/TN3270

Get the 3270 properties of the named session.

Query Options

None

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer *VIEW* privilege on the session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 200 = Ok

Successful Reply Document

SessionTN3270 in schema SessionTN3270.xsd

The reply contains properties such as Hostname, Port, and DeviceName.

OPTIONS sessions/<sessionName>/TN3270

Obtains a list of HTTP methods that can be issued against this URI.

Query Options

None

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer *VIEW* privilege on the session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 200 = Ok

A result of 404 (Not Found) is returned when the associated session is not a 3270 session.

Allow

List of allowed methods. This header contains OPTIONS, HEAD, GET.

Successful Reply Document

AllowedMethods in schema AllowedMethods.xsd

This reply contains the same methods as listed in the HTTP *Allow* header.

URI https://localhost:8443/apwebsvc/sessions/<sessionName>/TN5250

This URI represents the 5250 communication properties of the specified session.

Do not construct this URI in an application. Only use this URI to retrieve the value of the DeviceDetails property that is returned by the GET method on a:

…/apwebsvc/sessions/<sessionName> URI

Only one of the following URI subnodes

../<sessionName>/Asynchronous,
../<sessionName>/TN3270, or
../<sessionName>/TN5250

are valided for any given session.

GET sessions/<sessionName>/TN5250

Get the 5250 properties of the named session.

Query Options

None

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer VIEW privilege on the session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 200 = Ok

Successful Reply Document

SessionTN5250 in schema SessionTN5250.xsd

The reply contains properties such as Hostname, Port, and DeviceName.

OPTIONS sessions/<sessionName>/TN5250

Obtains a list of HTTP methods that can be issued against this URI.

Query Options

None

Request Document

None

HTTP Request Headers

Authorization

Supply a user ID and password in the HTTP Basic Authentication format. The user ID must be a user account with the AP Remote Viewer *VIEW* privilege on the session that is the target of this HTTP method.

HTTP Response Headers

Status

Successful result = 200 = Ok

A result of 404 (Not Found) is returned when the associated session is not a 5250 session.

Allow

List of allowed methods. This header contains OPTIONS, HEAD, GET.

Successful Reply Document

AllowedMethods in schema AllowedMethods.xsd

This reply contains the same methods as listed in the HTTP Allow header.

Index

3

A

C

D

E

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X