Before using this information and the product it supports, be sure to read the general information in Notices.
## Contents

### Tables

- vii

### About this document

- ix
  - Who should read this document
  - ix
  - How this document is organized
  - ix
  - Notices and statements used in this document
  - ix
  - Using the documentation
  - ix

### New in this release

- xi

### Introduction

- 1
  - REXX and SA IOM
  - 1
  - Enhancing SA IOM with REXX
  - 2
  - ANSI escape sequence support
  - 3
  - REXX functions, by their use
  - 7

### Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFR_AUDIT_LOGSIZE</td>
<td>17</td>
</tr>
<tr>
<td>AFR_AUDIT_REOPEN_LOG</td>
<td>17</td>
</tr>
<tr>
<td>AFR_ANSI</td>
<td>18</td>
</tr>
<tr>
<td>AFR_ASYNC_DELETE</td>
<td>18</td>
</tr>
<tr>
<td>AFR_ASYNC_WTOR</td>
<td>19</td>
</tr>
<tr>
<td>AFR_BROADCAST</td>
<td>20</td>
</tr>
<tr>
<td>AFR_CHECK_TRAP</td>
<td>21</td>
</tr>
<tr>
<td>AFR_CHECK_WTOR</td>
<td>22</td>
</tr>
<tr>
<td>AFR_CLEAR_TRAP</td>
<td>23</td>
</tr>
<tr>
<td>AFR_CLEAR_WTOR</td>
<td>23</td>
</tr>
<tr>
<td>AFR_CLS</td>
<td>24</td>
</tr>
<tr>
<td>AFR_CNTL_TAPI_START</td>
<td>24</td>
</tr>
<tr>
<td>AFR_CNTL_TAPI_STOP</td>
<td>25</td>
</tr>
<tr>
<td>AFR_COMBREAK</td>
<td>25</td>
</tr>
<tr>
<td>AFR_COMCLOSE</td>
<td>26</td>
</tr>
<tr>
<td>AFR_COMCTRL</td>
<td>26</td>
</tr>
<tr>
<td>AFR_COMGETBAUD</td>
<td>27</td>
</tr>
<tr>
<td>AFR_COMGETBITS</td>
<td>27</td>
</tr>
<tr>
<td>AFR_COMGETPARITY</td>
<td>28</td>
</tr>
<tr>
<td>AFR_COMGETSTOPBITS</td>
<td>28</td>
</tr>
<tr>
<td>AFR_COMGETSTR</td>
<td>29</td>
</tr>
<tr>
<td>AFR_COMINPUTBYTES</td>
<td>29</td>
</tr>
<tr>
<td>AFR_COMJOURNALSTART</td>
<td>29</td>
</tr>
<tr>
<td>AFR_COMJOURNALSTOP</td>
<td>30</td>
</tr>
<tr>
<td>AFR_COMOPEN</td>
<td>32</td>
</tr>
<tr>
<td>AFR_COMOUTPUTBYTES</td>
<td>33</td>
</tr>
<tr>
<td>AFR_COMPUTSTR</td>
<td>34</td>
</tr>
<tr>
<td>AFR_COMSETBAUD</td>
<td>35</td>
</tr>
<tr>
<td>AFR_COMSETLINEC</td>
<td>36</td>
</tr>
<tr>
<td>AFR_COMSTATUS</td>
<td>36</td>
</tr>
<tr>
<td>AFR_CONSOLE_TYPE</td>
<td>37</td>
</tr>
<tr>
<td>AFR_CURSOR</td>
<td>38</td>
</tr>
<tr>
<td>AFR_EMIT_UA_SNMP_TRAP</td>
<td>38</td>
</tr>
<tr>
<td>AFR_EMAIL_LOGSIZE</td>
<td>40</td>
</tr>
<tr>
<td>AFR_EMAIL_REOPEN_LOG</td>
<td>40</td>
</tr>
<tr>
<td>AFR_ERROR</td>
<td>41</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------</td>
</tr>
<tr>
<td>AFR_GET_LINE</td>
<td>42</td>
</tr>
<tr>
<td>AFR_GET_STATUS</td>
<td>42</td>
</tr>
<tr>
<td>AFR_GETENV</td>
<td>43</td>
</tr>
<tr>
<td>AFR_GETOWNER</td>
<td>46</td>
</tr>
<tr>
<td>AFR_GMT</td>
<td>47</td>
</tr>
<tr>
<td>AFR_KBHIT</td>
<td>48</td>
</tr>
<tr>
<td>AFR_KEY_ET</td>
<td>49</td>
</tr>
<tr>
<td>AFR_LOAD</td>
<td>50</td>
</tr>
<tr>
<td>AFR_LOAD_COLOR</td>
<td>50</td>
</tr>
<tr>
<td>AFR_LOCK</td>
<td>52</td>
</tr>
<tr>
<td>AFR_LOG</td>
<td>53</td>
</tr>
<tr>
<td>AFR_LOGSIZE</td>
<td>54</td>
</tr>
<tr>
<td>AFR_MC_LASTMSGNUM</td>
<td>54</td>
</tr>
<tr>
<td>AFR_MC_LASTMSGTEXT</td>
<td>55</td>
</tr>
<tr>
<td>AFR_MC_LASTMSGTIME</td>
<td>55</td>
</tr>
<tr>
<td>AFR_MC_LOGSIZE</td>
<td>56</td>
</tr>
<tr>
<td>AFR_MC_REOPEN_LOG</td>
<td>56</td>
</tr>
<tr>
<td>AFR_MC_SEND</td>
<td>57</td>
</tr>
<tr>
<td>AFR_MODEMANSWER</td>
<td>58</td>
</tr>
<tr>
<td>AFR_MODEMCALL</td>
<td>58</td>
</tr>
<tr>
<td>AFR_MODEMDISCONNECT</td>
<td>60</td>
</tr>
<tr>
<td>AFR_MODEMSTATUS</td>
<td>60</td>
</tr>
<tr>
<td>AFR_NOTIFY</td>
<td>61</td>
</tr>
<tr>
<td>AFR_NOTIFY_ACK</td>
<td>62</td>
</tr>
<tr>
<td>AFR_NOTIFY_EXIT</td>
<td>64</td>
</tr>
<tr>
<td>AFR_NOTIFY_INIT</td>
<td>64</td>
</tr>
<tr>
<td>AFR_NOTIFY_LASTERROR</td>
<td>64</td>
</tr>
<tr>
<td>AFR_NOTIFY_SET_RESULT</td>
<td>65</td>
</tr>
<tr>
<td>AFR_OP_ERROR</td>
<td>66</td>
</tr>
<tr>
<td>AFR_PEER_CLOSE</td>
<td>67</td>
</tr>
<tr>
<td>AFR_PEER_GETLASTERROR</td>
<td>68</td>
</tr>
<tr>
<td>AFR_PEER_OPEN</td>
<td>69</td>
</tr>
<tr>
<td>AFR_PEER_QUERY</td>
<td>70</td>
</tr>
<tr>
<td>AFR_PEER_RCV</td>
<td>73</td>
</tr>
<tr>
<td>AFR_PEER_SEND</td>
<td>75</td>
</tr>
<tr>
<td>AFR_PEER_SSENDRCV</td>
<td>77</td>
</tr>
<tr>
<td>AFR_PROMPT</td>
<td>79</td>
</tr>
<tr>
<td>AFR_PROMPT_USER</td>
<td>80</td>
</tr>
<tr>
<td>AFR_REOPEN_LOG</td>
<td>81</td>
</tr>
<tr>
<td>AFR_RXQUEUE_CREATE</td>
<td>82</td>
</tr>
<tr>
<td>AFR_RXQUEUE_DELETE</td>
<td>83</td>
</tr>
<tr>
<td>AFR_RXQUEUE_EXISTS</td>
<td>83</td>
</tr>
<tr>
<td>AFR_RXQUEUE_EXTERNAL</td>
<td>84</td>
</tr>
<tr>
<td>AFR_RXQUEUE_GET</td>
<td>84</td>
</tr>
<tr>
<td>AFR_RXQUEUE_PURGE</td>
<td>85</td>
</tr>
<tr>
<td>AFR_RXQUEUE_SET</td>
<td>85</td>
</tr>
<tr>
<td>AFR_SAY</td>
<td>86</td>
</tr>
<tr>
<td>AFR_SCREEN_WAIT</td>
<td>86</td>
</tr>
<tr>
<td>AFR_SELECT</td>
<td>87</td>
</tr>
<tr>
<td>AFR_SELECTED</td>
<td>88</td>
</tr>
<tr>
<td>AFR_SEND</td>
<td>89</td>
</tr>
<tr>
<td>AFR_SEND_3270</td>
<td>90</td>
</tr>
<tr>
<td>AFR_SEND_ASCII</td>
<td>92</td>
</tr>
<tr>
<td>AFR_SEND_EMAIL</td>
<td>92</td>
</tr>
<tr>
<td>AFR_SEND_EMAIL</td>
<td>92</td>
</tr>
<tr>
<td>AFR_SEND_MESSAGE</td>
<td>94</td>
</tr>
<tr>
<td>AFR_SEND_RPAGE</td>
<td>95</td>
</tr>
</tbody>
</table>
Tables

1. REXX functions, according to their use ........................................... 7
2. AFR_GETENV Variables .................................................................. 44
3. AFR_MODEMCALL Dialstring characters ........................................ 59
4. Modem status events ..................................................................... 59
5. AFR_SEND_3270 Modified HLLAPI codes ...................................... 91
6. AFR_VOICEDIAL Dialstring characters ........................................ 124
7. Notify return codes ..................................................................... 133
About this document

This document provides detailed information about IBM-supplied REXX functions that you can use with IBM Tivoli System Automation for Integrated Operations Management (SA IOM).

Who should read this document

If you want to write scripts to automate and customize SA IOM features, you should read this document to become familiar with REXX functions that will help you do this. You need to be familiar with IBM PC-compatible computers, the Windows environment, and the hardware and software of the host computer systems that will be connected to SA IOM.

How this document is organized

This document contains the following sections:

- “New in this release” on page xi lists the SA IOM REXX functions that are newly documented in this release.
- “Introduction” on page 1 provides an introduction to REXX, describes important concepts that you must know in order to effectively use REXX with this product, and groups the SA IOM REXX functions according to their use.
- “Functions” on page 17 details all the SA IOM REXX functions, in alphabetical order.

Notices and statements used in this document

The following types of notices and statements are used in this document:

- **Note:** These notices provide important tips, guidance, or advice.
- **Important:** These notices provide information or advice that might help you avoid inconvenient or problem situations.
- **Attention:** These notices indicate possible damage to programs, devices, or data. An attention notice is placed just before the instruction or situation in which damage could occur.
- **Caution:** These statements indicate situations that can be potentially hazardous to you. A caution statement is placed just before the description of a potentially hazardous procedure step or situation.
- **Danger:** These statements indicate situations that can be potentially lethal or extremely hazardous to you. A danger statement is placed just before the description of a potentially lethal or extremely hazardous procedure step or situation.

Using the documentation

IBM provides the following set of documentation for v2.1:

- *System Automation for Integrated Operations Management REXX Functions Reference*
- *System Automation for Integrated Operations Management Quick Start Guide*

These documents are available as PDF files on the product media.
Printing this document

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

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

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

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

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

Printing problems

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

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

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

Contacting Adobe

If additional information is needed about Adobe Acrobat Reader or printing problems, see the Readme.pdf file that ships with Adobe Acrobat Reader or contact Adobe at (www.adobe.com).

Adding annotations to PDF files

If you have purchased the Adobe Acrobat application, you can add annotations to IBM documentation in .PDF format. See the Adobe product for instructions on using the Acrobat annotations tool and its features.
New in this release

IBM has added the following REXX functions to those already written for Tivoli SA IOM V2.1.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFR_AUDIT_LOGSIZE</td>
<td>Retrieves the number of bytes in the audit log.</td>
</tr>
<tr>
<td>AFR_AUDIT_REOPEN_LOG</td>
<td>Reallocates the current audit log under a specified file name extension and starts a fresh log file.</td>
</tr>
<tr>
<td>AFR_CNTL_TAPI_START</td>
<td>Initializes the SA IOM server's TAPI usage after stopping it with AFR_CNTL_TAPI_STOP.</td>
</tr>
<tr>
<td>AFR_CNTL_TAPI_STOP</td>
<td>Stops the SA IOM server's TAPI usage.</td>
</tr>
<tr>
<td>AFR_EMIT_UA_SNMP_TRAP</td>
<td>Use this function only to send an SNMP trap from SA IOM to the IBM Tivoli Universal Agent.</td>
</tr>
<tr>
<td>AFR_EMAIL_LOGSIZE</td>
<td>Retrieves the number of bytes in the e-mail log.</td>
</tr>
<tr>
<td>AFR_EMAIL_REOPEN_LOG</td>
<td>Reallocates the current e-mail log under a specified file name extension and starts a fresh log file.</td>
</tr>
<tr>
<td>AFR_GET_STATUS</td>
<td>Obtains the status of the active emulation session.</td>
</tr>
<tr>
<td>AFR_LOGSIZE</td>
<td>Retrieves the number of bytes in the server log.</td>
</tr>
<tr>
<td>AFR_NOTIFY</td>
<td>Starts an alert escalation process based on the data configured in the alert escalation database.</td>
</tr>
<tr>
<td>AFR_NOTIFY_ACK</td>
<td>Acknowledges the receipt of an alert message and indicates your status.</td>
</tr>
<tr>
<td>AFR_NOTIFY_EXIT</td>
<td>Logs off from the alert escalation database.</td>
</tr>
<tr>
<td>AFR_NOTIFY_INIT</td>
<td>Logs on to the alert escalation database.</td>
</tr>
<tr>
<td>AFR_NOTIFY_LASTERROR</td>
<td>Retrieves information about the last recorded error for this alert escalation script.</td>
</tr>
<tr>
<td>AFR_NOTIFY_SET_RESULT</td>
<td>Reports the completion status of the alert escalation script.</td>
</tr>
<tr>
<td>AFR_PROMPT_USER</td>
<td>Produces a pop-up message with an input field for a response from the specified SA IOM user.</td>
</tr>
<tr>
<td>AFR_SELECTED</td>
<td>Use this function to retrieve the name of the currently selected emulation session.</td>
</tr>
<tr>
<td>AFR_UA_WRITE</td>
<td>Writes a record to saiom2ua.txt, which is monitored by the Universal Agent.</td>
</tr>
<tr>
<td>AFR_USER_CHANGEPSW</td>
<td>Changes the specified userid's password when the old password is known.</td>
</tr>
<tr>
<td>AFR_USER_LOGONTYPE</td>
<td>Retrieves or sets the password authentication method SA IOM uses to validate log on to the server.</td>
</tr>
<tr>
<td>AFR_USER_SETPSW</td>
<td>Sets the specified userid's password.</td>
</tr>
</tbody>
</table>

Also, the "New in this release" section of the System Automation for Integrated Operations Management User's Guide provides a feature-oriented explanation of what is new.
Introduction

This part of the document provides information you need to successfully use REXX with this product.

REXX and SA IOM

What is REXX?

The REstructured eXtended eXecutor language (REXX) is a full-featured programming language that can be used to write applications.

This structured procedural language was designed to make programming easier for programmers of all levels of experience from novice through expert. One of the main features of REXX is the ease with which it can issue commands to its host environment, like a batch command file. Another feature is its ability to call external functions, which can be written in any language.

REXX terminology

This document uses several terms that have a specific meaning when discussing REXX.

External Functions

Functions that are not a built-in part of REXX. IBM supplies a set of external functions for use with SA IOM. These are collectively called the SA IOM REXX API (Application Programming Interface).

Clause

A single instruction. Clauses are separated either by carriage-return line-feed pairs or by semicolons.

Literal Strings

Symbols enclosed in single or double quotes are literal strings. If nothing is enclosed between the quotes, then it is called a null string. Literal strings are always constants; therefore, if a literal string is composed solely of numeric digits, it can be used as a number in arithmetic calculations.

Variables

All variables evaluate to literal strings. Variable names can be up to 250 characters long and must start with any character from a-z, A-Z, an exclamation mark (!), or a question mark (?). The period (.) has a special meaning in variable names; do not use this character unless you understand the concept of compound symbols. Note that REXX always translates lowercase letters that aren't enclosed in quotes to upper case before using them. If you have a clause creating a variable called apple, for instance, REXX renames the variable to APPLE when interpreting the clause.

Constants

Symbols that start with the characters 0-9, -, or + are constants. All valid numbers are constants, but not all constants are valid numbers. The symbol 8mylunch, for example, is not a valid number, and it cannot be used as a variable name. Its value, 8MYLUNCH, is always constant.

For more information

For information about specific REXX versions supported by this product, see the System Automation for Integrated Operations Management User’s Guide.
For more information about the REXX language itself, refer to the Open Object
REXX Web site at:
http://www.oorexx.org/

There are also many excellent third-party books on programming with the REXX
language.

---

Enhancing SA IOM with REXX

This section gives you some examples of why you would want to enhance SA IOM
with REXX.

Automating tasks

The SA IOM external REXX functions, when used in a REXX script, allow you to
automate virtually any task an operator at a console can do. The scope of these
functions includes:
- Alert escalation functions
- Serial communications port and modem functions
- Environment control functions
- Message log write functions
- Message collector functions
- REXX program control functions
- Session control/read/write/trap functions
- Shared memory control/read/write functions
- Voice functions
- Peer functions
- Queue operation functions
- Telnet functions

You can also execute any Windows command or a batch file from within a REXX
script and send the output to the session window.

Customizing scripts

Several SA IOM features, such as beeper paging and automated time acquisition,
do little more than pass user-specified parameters to REXX sample scripts provided
with SA IOM. It is these separate scripts that actually initiate the beeper page or
acquire the time. An experienced programmer can modify these scripts to customize
the way the beeper paging or time acquisition features work.

But this is only one way to use REXX with SA IOM. You can control and interact
with any REXX script without leaving SA IOM. You can write custom SA IOM
applications and execute them from within SA IOM. You have complete
programmability of nearly every conceivable remote automation task.

Automatically executing scripts at startup

SA IOM makes it easy for you to automatically execute scripts at SA IOM startup
time. (That is, if REXX support is configured.)
The AUTOEXEC.REX script
Whenever you start SA IOM, it looks for a REXX script named AUTOEXEC.REX. When it is found, SA IOM automatically executes this script, which contains product initialization files to program specific startup tasks such as initializing serial ports, modems, and other REXX scripts.

The AUTOEXEC.REX script looks for and, if found, calls the AUTOUSER.REX script. AUTOUSER.REX is a convenient place for you to add your customized startup actions without modifying the supplied version of AUTOEXEC.REX.

The AUTOUSER.REX script
If you have custom initialization files or custom-made REXX scripts that you want to start at SA IOM startup, you should create a file called AUTOUSER.REX for them.

This keeps them separate from the AUTOEXEC.REX SA IOM initialization script and prevents the possibility of overwriting modifications to your AUTOEXEC.REX file during migration to a new version of SA IOM.

The REXX script owner
All SA IOM REXX scripts have an associated “owner”. This is usually the person who started the script. (If the SA IOM server started a script automatically, as in the case of AUTOEXEC.REX described above, then “operator” is listed as the owner.) Only the owner of a REXX script, or a user belonging to a user group with script management authority, can cancel a running script.

REXX and shared memory
Several of the SA IOM external REXX functions are provided to allow scripts written in any language to acquire data from your SA IOM server emulation sessions.

The shared memory functions allow a REXX script to gather data from your sessions and put them in a named shared memory file. Programs written in C, or other languages that support shared memory, can then access this file and use the data put there by REXX. These other programs can also pass data back to REXX through the same shared file.

ANSI escape sequence support
This section provides a brief explanation of and a list of SA IOM-supported ANSI escape sequences.

SA IOM supports the use of ANSI escape sequences in user-written REXX scripts for changing the text format, text color, and background colors that appear on your screen during REXX sessions and to position the cursor to any point on a REXX session screen. In other words, SA IOM supports using ANSI escape sequences for the purpose of display enhancements. You can accomplish this in your scripts by using the syntax that is described next, and by using either the SAY REXX primitive or the AFR_SAY REXX function.

SA IOM does not support the use of ANSI escape sequences to reprogram the keys on your keyboard or to set modes. These functions are reserved for product use.
Syntax

The ANSI escape sequences follow this pattern:
ESC[n;n;nc

where:
ESC
The ASCII escape character code 27. You can SAY (or AFR_SAY) this in your REXX scripts as D2C(27). For example, the following instruction homes the cursor:
SAY D2C(27)||"[H"

[  
A left bracket character follows the escape character.

n  
A numeric parameter. The number of n parameters depends on the command.

c  
A command character.

Usage notes

Adding the following line to your script:
ESC = D2C(27) /* ANSI escape character */
defines a string variable, ESC, which you can use as an argument for subsequent SAY or AFR_SAY functions. This saves many keystrokes and makes your script easier to read.

Omitted parameters in ANSI escape sequences default to zero, except as noted for specific commands. Excess parameters are ignored. An unexpected character terminates a sequence, and it and any remaining characters appear on the screen. For example, the command:
ESC[1;1H
homes the cursor, but the command:
ESC[1,1H
shows the characters:
,1H
at the current cursor position since the comma (,) is an invalid sequence character.

Examples

Here are some examples of REXX code that uses ANSI commands.
• The following example changes the text color screen attributes to yellow foreground and blue background:
  nu1 = AFR_SAY(D2C(27)||"[33;44m")
• The following example moves the cursor to the coordinates line 10, column 60, then SAYs some text.
  nu1 = AFR_SAY(D2C(27)||"[10;60H"||"Hello")
• The following example changes the screen scrolling parameter from 12 lines (the SA IOM default) to 1 line (the system default):
  nu1 = AFR_SAY(D2C(27)||"[1S")
Emulated commands

The following commands are supported by SA IOM. Capitalization is significant; these commands are case-sensitive.

```
ESC[l;cH  Home the cursor
ESC[l;cf  Cursor position
```

Both commands move the cursor to a specified position (coordinate), where:

- `l` is the line number, from 1-24.
- `c` is the column number, from 1-80.

If you do not specify a coordinate, 1 is assumed. If you specify a number too large for a coordinate, the maximum is assumed. The home position is line 1, column 1, in the upper-left corner of the screen. The lower-right corner of the screen is line 24, column 80.

```
ESC[ nA  Cursor up
ESC[ nB  Cursor down
ESC[ nC  Cursor right
ESC[ nD  Cursor left
```

The 4 commands above move the cursor the specified number of lines (or columns) in the requested direction. If you do not specify a number `n`, 1 is assumed. Movement stops when the cursor reaches the screen border.

```
ESC[ s  Save cursor position.
ESC[ u  Restore cursor position. Returns the cursor to the position stored by the last ESC[s command. If no ESC[s command has been issued, it moves the cursor to the home position.
ESC[2J  Erase display (homes cursor).
ESC[K  Erase to end of line.
ESC[ a;fbm  Set attribute, where:
  
a
  Sets the text attribute. See Text attributes on page 6.
  
f
  Sets the foreground color. See Foreground colors on page 6.
  
b
  Sets the background color. See Background colors on page 6.
```

You can specify more than one attribute by separating each attribute with semicolons. For colors, only the last specified will take effect. There is an internal limit of eight parameters per command.
Text attributes
0  All attributes turned off; sets green foreground on black background.
1  Bold on.
4  Underscore on.
5  Blink on.
7  Reverses the foreground and background colors.
8  Sets the foreground color the same as the background color.

Foreground colors
30  Black
31  Red
32  Green
33  Yellow (brown on PC)
34  Blue
35  Magenta
36  Cyan
37  White

Background colors
40  Black
41  Red
42  Green
43  Yellow (brown on PC)
44  Blue
45  Magenta
46  Cyan
47  White

Enhanced commands
The following commands are supported by SA IOM.

ESC[1K  Erase from beginning of line to cursor.
ESC[2K  Erase entire line containing cursor.
ESC[J  Erase from cursor to end of screen.
ESC[1J  Erase from beginning of screen to cursor.

The four commands above are compatible with the DEC VT100.

ESC[:;nH  Homes the cursor based on a 0-origin offset from home. The cursor position, \( n \), ranges from 0 to 1919.
ESC[:;nf  Moves cursor based on a 0-origin offset from home. The cursor position, \( n \), ranges from 0 to 1919.
Addressing Note: SA IOM supports more than one way to address the cursor. For example, all commands on the following line are equivalent and home the cursor.

ESC[H ESC[;H ESC[1;1H ESC[0;0;0H

Both commands on the following line are equivalent and move the cursor to the bottom right corner of the REXX session screen.

ESC[24;80H ESC[;;1919H

ESC[nS Set scrolling. Where n, the number of lines, ranges from 0 to 23. Specifying an n of 0 disables scrolling. With scrolling disabled, when the cursor reaches the bottom of the screen, the next line printed will start at the top of the screen. The session startup default is 12 (scroll half the screen).

REXX functions, by their use

The following table lists all of this product's REXX functions, grouped according to their use. This is intended to give you an idea of which functions work together with other functions and the type of work they do.

Table 1. REXX functions, according to their use

<table>
<thead>
<tr>
<th>Event Driven Automation Functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AFR_ASYNC_DELETE</td>
<td>Deletes the trap set by AFR_ASYNC_WTOR.</td>
</tr>
<tr>
<td>AFR_ASYNC_WTOR</td>
<td>Starts a REXX script when a WTOR message is found.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Communications Port and Modem Functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AFR_COMBREAK</td>
<td>Generates a break signal on a specified communications port.</td>
</tr>
<tr>
<td>AFR_COMCLOSE</td>
<td>De-allocates a serial communications port.</td>
</tr>
<tr>
<td>AFR_COMCTRL</td>
<td>Sets the method of flow control used by a serial port.</td>
</tr>
<tr>
<td>AFR_COMGETBAUD</td>
<td>Returns the current baud rate of the specified serial port.</td>
</tr>
<tr>
<td>AFR_COMGETBITS</td>
<td>Returns the current number of databits for the specified serial port.</td>
</tr>
<tr>
<td>AFR_COMGETPARITY</td>
<td>Returns the current type of parity for the specified serial port.</td>
</tr>
<tr>
<td>AFR_COMGETSTOPBITS</td>
<td>Returns the current number of stopbits for the specified serial port.</td>
</tr>
<tr>
<td>AFR_COMGETSTR</td>
<td>Returns a string from a serial port.</td>
</tr>
<tr>
<td>AFR_COMINPUTBYTES</td>
<td>Returns the number of bytes waiting in a serial port’s input buffer.</td>
</tr>
<tr>
<td>AFR_COMJOURNALSTART</td>
<td>Starts a journal file of a specified serial communication port.</td>
</tr>
<tr>
<td>AFR_COMJOURNALSTOP</td>
<td>Stops a journal file of a specified serial communication port.</td>
</tr>
<tr>
<td>AFR_COMOPEN</td>
<td>Allocates a serial port for use by the communications port and modem functions.</td>
</tr>
<tr>
<td>AFR_COMOUTPUTBYTES</td>
<td>Returns the number of unused bytes currently available in a serial port’s output buffer.</td>
</tr>
<tr>
<td>AFR_COMPUTSTR</td>
<td>Writes a string to a serial port output buffer.</td>
</tr>
<tr>
<td>AFR_COMSETBAUD</td>
<td>Sets the baud rate of a serial port.</td>
</tr>
</tbody>
</table>
Table 1. REXX functions, according to their use (continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFR_COMSETLINEC</td>
<td>Sets the line characteristics of a serial port.</td>
</tr>
<tr>
<td>AFR_COMSTATUS</td>
<td>Returns the status of a serial port's input and output lines.</td>
</tr>
</tbody>
</table>

**Note:** the following modem functions require a serial communications port configured for script use.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFR_MODEMANSWER</td>
<td>Puts a modem in a state ready to answer a call.</td>
</tr>
<tr>
<td>AFR_MODEMCALL</td>
<td>Commands a modem to dial a phone number.</td>
</tr>
<tr>
<td>AFR_MODEMDISCONNECT</td>
<td>Hangs up a modem.</td>
</tr>
<tr>
<td>AFR_MODEMSTATUS</td>
<td>Returns the status of the AFR_MODEMCALL or the AFR_MODEMANSWER function.</td>
</tr>
</tbody>
</table>

**Environment Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFR_GETENV</td>
<td>Returns the value of a server configuration data item.</td>
</tr>
<tr>
<td>AFR_GMT</td>
<td>Sets the server PC's clock.</td>
</tr>
<tr>
<td>AFR_SETENV</td>
<td>Assigns a string to a server configuration data item.</td>
</tr>
</tbody>
</table>

**Message and Logging Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFR_AUDIT_LOGSIZE</td>
<td>Retrieves the number of bytes in the audit log.</td>
</tr>
<tr>
<td>AFR_AUDIT_REOPEN_LOG</td>
<td>Reallocates the current audit log under a specified file name extension and starts a fresh log file.</td>
</tr>
<tr>
<td>AFR_BROADCAST</td>
<td>Sends a string to the server log and to all users where it is displayed in a pop-up window.</td>
</tr>
<tr>
<td>AFR_EMAIL_LOGSIZE</td>
<td>Retrieves the number of bytes in the e-mail log.</td>
</tr>
<tr>
<td>AFR_EMAIL_REOPEN_LOG</td>
<td>Reallocates the current e-mail log under a specified file name extension and starts a fresh log file.</td>
</tr>
<tr>
<td>AFR_ERROR</td>
<td>Sends a string to the server log and to the REXX script owner where it is displayed in a pop-up window.</td>
</tr>
<tr>
<td>AFR_LOG</td>
<td>Writes a string only to the server log.</td>
</tr>
<tr>
<td>AFR_LOGSIZE</td>
<td>Retrieves the number of bytes in the server log.</td>
</tr>
<tr>
<td>AFR_OP_ERROR</td>
<td>Sends a string to the server log and to all users with authority to respond to client messages.</td>
</tr>
<tr>
<td>AFR_PROMPT</td>
<td>Produces a pop-up message with an input field for a response from the REXX script owner.</td>
</tr>
<tr>
<td>AFR_PROMPT_USER</td>
<td>Produces a pop-up message with an input field for a response from the specified SA IOM user.</td>
</tr>
<tr>
<td>AFR_REOPEN_LOG</td>
<td>Reallocates the server log.</td>
</tr>
</tbody>
</table>

**Message Collector Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFR_MC_LASTMSGNUM</td>
<td>Retrieves the number of messages sent to the Message Collector.</td>
</tr>
<tr>
<td>AFR_MC_LASTMSGTEXT</td>
<td>Retrieves the last message received by the Message Collector.</td>
</tr>
<tr>
<td>AFR_MC_LASTMSGTIME</td>
<td>Retrieves the time of the last message received by the Message Collector.</td>
</tr>
<tr>
<td>AFR_MC_LOGSIZE</td>
<td>Retrieves the number of bytes in the Message Collector log.</td>
</tr>
<tr>
<td>AFR_MC_REOPEN_LOG</td>
<td>Reallocates the current Message Collector log under a specified name extension and starts a fresh log.</td>
</tr>
</tbody>
</table>
### Table 1. REXX functions, according to their use (continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AFR_MC_SEND</strong></td>
<td>Use this function to send a message to the Message Collector.</td>
</tr>
<tr>
<td><strong>Notify Functions (used for alert escalation processing)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>AFR_NOTIFY</strong></td>
<td>Starts an alert escalation process based on the data configured in the alert escalation database.</td>
</tr>
<tr>
<td><strong>AFR_NOTIFY_ACK</strong></td>
<td>Acknowledges the receipt of an alert message and indicates your status.</td>
</tr>
<tr>
<td><strong>AFR_NOTIFY_EXIT</strong></td>
<td>Logs off from the alert escalation database.</td>
</tr>
<tr>
<td><strong>AFR_NOTIFY_INIT</strong></td>
<td>Logs on to the alert escalation database.</td>
</tr>
<tr>
<td><strong>AFR_NOTIFY_LASTERROR</strong></td>
<td>Retrieves information about the last recorded error for this alert escalation script.</td>
</tr>
<tr>
<td><strong>AFR_NOTIFY_SET_RESULT</strong></td>
<td>Reports the completion status of the alert escalation script.</td>
</tr>
<tr>
<td><strong>Peer Functions</strong></td>
<td></td>
</tr>
<tr>
<td><strong>AFR_PEER_CLOSE</strong></td>
<td>Terminates a conversation with another SA IOM server, or AF/OPERATOR address space, or other peer system.</td>
</tr>
<tr>
<td><strong>AFR_PEER_GETLASTERROR</strong></td>
<td>Finds the return code for the last peer function that executed.</td>
</tr>
<tr>
<td><strong>AFR_PEER_OPEN</strong></td>
<td>Starts a conversation with another SA IOM server, or AF/OPERATOR address space, or other peer system.</td>
</tr>
<tr>
<td><strong>AFR_PEER_QUERY</strong></td>
<td>Finds out various peer-related status information.</td>
</tr>
<tr>
<td><strong>AFR_PEER_RCV</strong></td>
<td>Receives data from a peer system.</td>
</tr>
<tr>
<td><strong>AFR_PEER_SEND</strong></td>
<td>Sends data to a peer system.</td>
</tr>
<tr>
<td><strong>AFR_PEER_SENDRCV</strong></td>
<td>Sends data to a peer system and waits for response data to be received back from the peer system.</td>
</tr>
<tr>
<td><strong>Queue Operation Functions</strong></td>
<td></td>
</tr>
<tr>
<td><strong>AFR_RXQUEUE_CREATE</strong></td>
<td>Creates a new named REXX queue.</td>
</tr>
<tr>
<td><strong>AFR_RXQUEUE_DELETE</strong></td>
<td>Deletes the internal named REXX queue.</td>
</tr>
<tr>
<td><strong>AFR_RXQUEUE_EXISTS</strong></td>
<td>Inquires if the specified REXX queue exists.</td>
</tr>
<tr>
<td><strong>AFR_RXQUEUE_EXTERNAL</strong></td>
<td>Declare a specified queue as external.</td>
</tr>
<tr>
<td><strong>AFR_RXQUEUE_GET</strong></td>
<td>Retrieve the name of the current REXX queue.</td>
</tr>
<tr>
<td><strong>AFR_RXQUEUE_PURGE</strong></td>
<td>Empty an internal queue.</td>
</tr>
<tr>
<td><strong>AFR_RXQUEUE_SET</strong></td>
<td>Make the specified REXX queue current.</td>
</tr>
<tr>
<td><strong>Program Control Functions</strong></td>
<td></td>
</tr>
<tr>
<td><strong>AFR_ANSI</strong></td>
<td>Enables or disables ANSI support, or returns the current value of the ANSI setting.</td>
</tr>
<tr>
<td><strong>AFR_CLS</strong></td>
<td>Clears a REXX program output screen.</td>
</tr>
<tr>
<td><strong>AFR_GETOWNER</strong></td>
<td>Retrieves a REXX script owner name.</td>
</tr>
<tr>
<td><strong>AFR_KBHIT</strong></td>
<td>Returns ASCII values from the keyboard buffer.</td>
</tr>
<tr>
<td><strong>AFR_LOCK</strong></td>
<td>Obtains exclusive access to a resource controlled by SA IOM.</td>
</tr>
<tr>
<td><strong>AFR_SAY</strong></td>
<td>Produces formatted screen output. May contain ANSI escape sequences.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AFR_START_PROGRAM</td>
<td>Starts a REXX script.</td>
</tr>
<tr>
<td>AFR_STOP_PROGRAM</td>
<td>Stops a REXX script.</td>
</tr>
<tr>
<td>AFR_SUSPEND</td>
<td>Forces a REXX script to pause.</td>
</tr>
<tr>
<td>AFR_UNLOCK</td>
<td>Frees access to a resource controlled by SA IOM.</td>
</tr>
<tr>
<td>AFR_WHOAMI</td>
<td>Returns the script ID assigned to a REXX program.</td>
</tr>
<tr>
<td>AFR_WHOIS</td>
<td>Returns the status line information for the specified REXX script.</td>
</tr>
</tbody>
</table>

**Session Control Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFR_CHECK_TRAP</td>
<td>Returns the string trapped by the AFR_SET_TRAP function.</td>
</tr>
<tr>
<td>AFR_CHECK_WTOR</td>
<td>Returns the string trapped by the AFR_SET_WTOR function.</td>
</tr>
<tr>
<td>AFR_CLEAR_TRAP</td>
<td>Clears a trap.</td>
</tr>
<tr>
<td>AFR_CLEAR_WTOR</td>
<td>Clears a WTOR trap.</td>
</tr>
<tr>
<td>AFR_CONSOLE_TYPE</td>
<td>Returns the console type of the current emulation session.</td>
</tr>
<tr>
<td>AFR_CURSOR</td>
<td>Returns the cursor position in the current screen.</td>
</tr>
<tr>
<td>AFR_GET_LINE</td>
<td>Returns the next line of data from a currently selected Glass Teletype or Message Collector session.</td>
</tr>
<tr>
<td>AFR_GET_STATUS</td>
<td>Obtains the status of the active emulation session.</td>
</tr>
<tr>
<td>AFR_KEY_ET</td>
<td>Returns the elapsed time, in seconds, since a keystroke was last given to the currently selected emulation session.</td>
</tr>
<tr>
<td>AFR_LOAD</td>
<td>Returns a string from the current screen.</td>
</tr>
<tr>
<td>AFR_LOAD_COLOR</td>
<td>Returns the colors of the text on the current screen.</td>
</tr>
<tr>
<td>AFR_SCREEN_WAIT</td>
<td>Halts program execution until the selected emulation session screen changes or until the specified timeout expires.</td>
</tr>
<tr>
<td>AFR_SELECT</td>
<td>Makes an emulation session the current session for all other session functions described in this section.</td>
</tr>
<tr>
<td>AFR_SELECTED</td>
<td>Use this function to retrieve the name of the currently selected emulation session.</td>
</tr>
<tr>
<td>AFR_SEND</td>
<td>Sends a string to the specified session.</td>
</tr>
<tr>
<td>AFR_SEND_3270</td>
<td>Sends a modified HLLAPI string to the current session.</td>
</tr>
<tr>
<td>AFR_SEND_ASCII</td>
<td>Sends an ASCII string to the current session.</td>
</tr>
<tr>
<td>AFR_SEND_SCAN</td>
<td>Sends a single value directly to the 3270 emulation adapter.</td>
</tr>
<tr>
<td>AFR_SET_TRAP</td>
<td>Sets a trap for a string on the screen.</td>
</tr>
<tr>
<td>AFR_SET_TRAP2</td>
<td>Sets a trap for a string on the screen.</td>
</tr>
<tr>
<td>AFR_SET_WTOR</td>
<td>Sets a WTOR string trap in the current session.</td>
</tr>
</tbody>
</table>

**Send Message To Other System Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFR_EMIT_UA_SNMP_TRAP</td>
<td>Use this function only to send an SNMP trap from SA IOM to the IBM Tivoli Universal Agent.</td>
</tr>
</tbody>
</table>
### Table 1. REXX functions, according to their use (continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFR_SEND_EMAIL</td>
<td>Sends e-mail to the specified recipient or to multiple recipients.</td>
</tr>
<tr>
<td>AFR_SEND_MESSAGE</td>
<td>Sends a message to a Message Collector session via TCP/IP.</td>
</tr>
<tr>
<td>AFR_UA_WRITE</td>
<td>Writes a record to saiom2ua.txt, which is monitored by the Universal Agent.</td>
</tr>
</tbody>
</table>

#### Shared Memory Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFR_SHRACCESS</td>
<td>Gains access to a pre-allocated shared memory file.</td>
</tr>
<tr>
<td>AFR_SHRALLOC</td>
<td>Allocates a shared memory file.</td>
</tr>
<tr>
<td>AFR_SHRFREE</td>
<td>De-allocates a shared memory file.</td>
</tr>
<tr>
<td>AFR_SHRREAD</td>
<td>Returns a string from a shared memory file.</td>
</tr>
<tr>
<td>AFR_SHRWRI TE</td>
<td>Writes a string to a shared memory file.</td>
</tr>
</tbody>
</table>

#### Telnet Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFR_TELNET_CLOSE</td>
<td>Terminates a connection with a Telnet host.</td>
</tr>
<tr>
<td>AFR_TELNET_GETLASTERROR</td>
<td>Obtains the return code from the last executed Telnet REXX function.</td>
</tr>
<tr>
<td>AFR_TELNET_OPEN</td>
<td>Starts a connection to a Telnet host.</td>
</tr>
<tr>
<td>AFR_TELNET_QUERY</td>
<td>Obtains detailed information, including current connect and disconnect status, of any configured Telnet session.</td>
</tr>
<tr>
<td>AFR_TELNET_SET</td>
<td>Changes the protocol and local echo setting of a Telnet session.</td>
</tr>
</tbody>
</table>

#### Voice Functions

**Note:** The following functions can be used only if a supported voice adapter is installed in the SA IOM server. For information about specific voice adapters supported by this product, see the *System Automation for Integrated Operations Management User's Guide*.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFR_TTS_FILE</td>
<td>Speaks the text contained in a file to a user dialed into a Dialogic voice card.</td>
</tr>
<tr>
<td>AFR_TTS_STRING</td>
<td>Speaks the text contained in a string to a user dialed into a Dialogic voice card.</td>
</tr>
<tr>
<td>AFR_VOICEANSWER</td>
<td>Answers the phone.</td>
</tr>
<tr>
<td>AFR_VOICEBEEP</td>
<td>Generates beep tones.</td>
</tr>
<tr>
<td>AFR_VOICEDIAL</td>
<td>Dials a phone number.</td>
</tr>
<tr>
<td>AFR_VOICEGETTONES</td>
<td>Returns a touch-tone string.</td>
</tr>
<tr>
<td>AFR_VOICEHANGUP</td>
<td>Hangs up the phone and de-allocates the voice adapter.</td>
</tr>
<tr>
<td>AFR_VOICEINIT</td>
<td>Tests if a voice adapter is accessible by this application.</td>
</tr>
<tr>
<td>AFR_VOICERECORD</td>
<td>Records a voice message.</td>
</tr>
<tr>
<td>AFR_VOICESPEAK</td>
<td>Plays a voice message.</td>
</tr>
</tbody>
</table>

#### Miscellaneous Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFR_CNTL_TAPI_START</td>
<td>Initializes the SA IOM server's TAPI usage after stopping it with AFR_CNTL_TAPI_STOP.</td>
</tr>
</tbody>
</table>
Table 1. REXX functions, according to their use (continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFR_CNTL_TAPI_STOP</td>
<td>Stops the SA IOM server's TAPI usage.</td>
</tr>
<tr>
<td>AFR_USER_CHANGEPSW</td>
<td>Changes the specified userid's password when the old password is known.</td>
</tr>
<tr>
<td>AFR_USER_LOGONTYPE</td>
<td>Retrieves or sets the password authentication method.</td>
</tr>
<tr>
<td>AFR_USER_SETPSW</td>
<td>Sets the specified userid's password.</td>
</tr>
</tbody>
</table>

**Programming guidance**

**Important:** Many of the SA IOM external functions are interdependent. The value returned from one function is often used as a parameter in other functions. You should always, therefore, test a function's return before calling subsequent functions.

**About the Telnet REXX functions**

The Telnet REXX functions allow users to programmatically manage their Telnet host sessions. These functions are not mandatory for users who wish to configure and utilize Telnet sessions. Connects, disconnects, echo toggling, and so on, can continue to be performed using the Telnet graphical user interface that SA IOM provides.

The Telnet REXX functions provide a set of programming tools for detecting lost sessions, reconnecting sessions, automatically opening and logging on to Telnet hosts, and building other useful applications.

**About the peer REXX functions**

**Peer functions background**

In SA IOM there are a group of REXX peer functions that utilize TCP/IP. These functions can be divided into the following categories:

- Connection management (AFR_PEER_OPEN and AFR_PEER_CLOSE).
- Data transmission (AFR_PEER_SEND, AFR_PEER_SENDRCV, and AFR_PEER_RCV).
- Status reporting (AFR_PEER_GETLASTERROR and AFR_PEER_QUERY).

A typical SA IOM REXX script which makes use of these peer functions would begin with a call to AFR_PEER_QUERY to determine whether a particular link is active. If it is not active, an AFR_PEER_OPEN call would be made to start a conversation with the peer. This would be followed by one or more data transmission calls, such as AFR_PEER_SEND, to transmit data and/or REXX execs to the peer system. The script might then conclude with a call to AFR_PEER_CLOSE or it might leave the link active for subsequent REXX scripts to use.

As a rule, both SA IOM and AF/OPERATOR require a linkid parameter to be specified in all peer-to-peer commands and REXX API calls. The linkid is synonymous with the *conversation ID*. It is a handle that the peer who initiates the conversation assigns, and it uniquely identifies that conversation.

**Note:** When a peer system triggers a REXX script to execute on SA IOM, the triggered REXX script does not need to specify a linkid parameter in any
In addition to a linkid parameter, all outbound data transmissions require a DATATYPE parameter to signify the type of data being sent. DATATYPE can have one of the following string values:

- **EXEC**  
  Indicates that the SENDBUFF contains an OMEGACENTER Gateway for z/OS command or a REXX exec that is to be run on a peer system.

- **DATARPLY**  
  Indicates that the SENDBUFF contains response data that a peer system requested. Only use this if the other peer did a SENDRCV call and is waiting for a response.

- **USRDATA**  
  Indicates that the SENDBUFF contains user-specified data, such as input to a REXX exec running on a connected peer, or some other type of miscellaneous conversational data. Only use this if the other peer is in a RECEIVE state, such as waiting for data.

  Always use the USRDATA value if you are using the special link ID: "NONAF"

These DATATYPE values correspond to the ones supported by the OMEGACENTER Gateway for z/OS COMSDRCV REXX API function. SA IOM also supports an "ID" DATATYPE record, which allows a peer to identify attributes about itself, such as whether it will send ASCII or EBCDIC data.

Explicitly sending an ID record is not allowed in any "AF peer" to "AF peer" conversation, because SA IOM and AF/OPERATOR both generate their own internal ID records when a conversation starts.

**Note:** The defaults are that AF/OPERATOR sends EBCDIC text and integers in "big endian" order, SA IOM sends ASCII text and integers in "little endian" order, and there is no idle timeout limit.

### Error handling techniques

When an SA IOM REXX script terminates unexpectedly while utilizing any of the TCP/IP peer functions, it is important that the REXX script clean up, for example, clean up after a syntax shutdown. Otherwise, conversations could be left dangling and resources will not be released. This could prevent a conversation from being successfully restarted with either the same peer system or the same LINKID.

Therefore, always use the SIGNAL ON ERROR, SIGNAL ON FAILURE, and SIGNAL ON SYNTAX REXX procedures in SA IOM scripts that contain one of the peer REXX functions. It is always a good programming practice to use these error handling services, but it is particularly important when writing communications-based scripts.

### About the queue operation REXX functions

The queue operation REXX functions provide comprehensive RXQUEUE support for REXX scripts running under SA IOM. You can use these functions in conjunction with the REXX PUSH and PULL commands to perform queueing operations.

**Note:** The “current” queue is used as the target for PUSH and PULL commands.
About STDIN and STDOUT in REXX functions under SA IOM

There are a number of functions that work on file handles. When used with the default STDIN and STDOUT file handles, one would expect the output to go to the selected REXX console and screen input to come from typing on the keyboard that is connected to the selected REXX console. As documented in the Open Object REXX Programming Guide, these are not intercepted by REXX and go straight to the operating system. When running REXX scripts under SA IOM, output sent to STDOUT will just disappear, and attempting to get STDIN input will hang the script as no input will ever arrive.

In your REXX programs running under SA IOM.

- The functions CHARIN, LINEIN, CHARS, LINES, and PARSE LINEIN - when intended to be used with the default STDIN file handle - should be avoided. Use PULL and PARSE PULL instead.
- The function LINEOUT - when intended to be used with the default STDOUT file handle - should be avoided. Use SAY instead.
- The function CHAROUT - when intended to be used with the default STDOUT file handle - should be avoided. Use AFR_SAY instead.

These are the default functions supplied with REXX. However, it is possible to extend REXX and this caution holds. For example, the RexxUtil package that comes with REXX and has a SysDumpVariables function that writes to STDOUT. This function produces no output when running under SA IOM.

Internal versus external REXX queues

Use the AFR_RXQUEUE functions to manage “internal” REXX queues. Internal queues are only for communication between REXX programs running under SA IOM. You cannot use internal queues for communicating with other processes or REXX programs running outside of SA IOM.

You should use standard REXX RXQUEUE function calls to manage “external” REXX queues. External queues are for communication between REXX programs running under SA IOM and programs running outside of SA IOM.

If a REXX program running inside SA IOM needs to use a REXX queue to communicate with a REXX program or other process running outside of SA IOM, it must use the AFR_RXQUEUE_EXTERNAL function to declare this queue as external to SA IOM. SA IOM will then pass all operations on this queue to REXX.

Note: There can be only one current external queue at a given time for all REXX programs running under SA IOM, since REXX manages queue currency at the process level.

Using RXQUEUE under SA IOM

SA IOM emulates the following REXX commands and RXQUEUE options for internal queues. For external queues, SA IOM passes the command or RXQUEUE option to REXX.

PUSH
Push a line on to the current queue

PULL
Pull a line from the current queue or from console input if the queue is empty

LINEIN("QUEUE:")
Same as PULL (see Note 3 on page 15)

LINEOUT("QUEUE:")
Same as PUSH
QUEUED() Retrieve number of lines on the current queue (see Note 8)
RXQUEUE("SET") Set the specified queue to be current (see Note 4)

SA IOM does not emulate the following RXQUEUE options. You must use the corresponding AFR_RXQUEUE functions instead.

RXQUEUE("CREATE") Create a new queue (see Note 5 below)
RXQUEUE("DELETE") Delete a queue (see Note 6 below)
RXQUEUE("GET") Retrieve name of the current queue (see Note 2 below)

Note the following when you are using RXQUEUE under SA IOM:
1. When a REXX program is started under SA IOM, it is automatically given a default queue name of SESSION.
2. REXX does not support an exit for the RXQUEUE("GET") function. Therefore, you should use AFR_RXQUEUE_GET to retrieve the name of the current internal REXX queue.
3. Because of REXX exit behavior, LINEIN("QUEUE") is handled identically to PULL. That is, if the current queue is empty when LINEIN("QUEUE") is called, the REXX program will wait for user input in the console window. This differs from the documented behavior of LINEIN, where it waits for a line to appear on the queue if the queue is currently empty.
4. If the specified queue does not exist, RXQUEUE("SET") creates a new internal REXX queue by that name, except if the queue name is previously declared as external using AFR_RXQUEUE_EXTERNAL.
5. You do not need to call RXQUEUE("CREATE") to create internal REXX queues. If you do call it, a REXX queue is created without SA IOM awareness.
6. RXQUEUE("DELETE") does not affect internal SA IOM queues, but can be used to delete an existing queue created using RXQUEUE("CREATE") and declared as external to SA IOM. If RXQUEUE("DELETE") is called, a REXX queue is deleted without SA IOM awareness. It will not affect the integrity of the corresponding internal REXX queue.
7. According to REXX rules, when you use RXQUEUE("SET") for a queue declared as external, the REXX interpreter, not SA IOM, will handle subsequent PUSH, PULL, LINEIN, and LINEOUT operations. This behavior will persist until AFR_RXQUEUE_SET is called specifying a non-declared queue. When this is done, SA IOM will handle PUSH, PULL, LINEIN, and LINEOUT operations.
8. SA IOM cannot emulate LINES("QUEUE:"); use QUEUED instead.
Functions

This chapter provides detailed information about each IBM-supplied external REXX function. Here all functions are listed in alphabetical order for ease of reference. The Remarks section of each function notes if that function requires a particular configuration on the SA IOM server before that function is available for use.

AFR_AUDIT_LOGSIZE

Use this function to retrieve the number of bytes in the audit log.

Format

AFR_AUDIT_LOGSIZE()

This function accepts no parameters.

Returns

This function returns the number of bytes in the audit log.

Example

Num = AFR_AUDIT_LOGSIZE()

AFR_AUDIT_REOPEN_LOG

Use this function to reallocate the current audit log under a specified file name extension and start a fresh log file.

When you call AFR_AUDIT_REOPEN_LOG:
1. The current RPAUDIT. extension (if present) is deleted.
2. RPAUDIT.LOG is closed.
3. RPAUDIT.LOG is named to RPAUDIT. extension.
4. A new, empty RPAUDIT.LOG is opened for recording audit trails.

Format

AFR_AUDIT_REOPEN_LOG(extension)

extension Up to 3 characters used as a file name extension when renaming the current RPAUDIT.LOG file. If you omit this parameter, the default extension .BAK is used.

Returns

0 The log file was reallocated successfully.
1 The specified extension is invalid.

Remarks

• When the renaming process is successful, the last message in the old log file (now renamed as RPAUDIT. extension) and the first message in the new RPAUDIT.LOG file records the successful reallocation of the log.
AFR_AUDIT_REOPEN_LOG

- If either RPAUDIT.LOG or RPAUDIT. extension is in use by another process when this function is called, the renaming process fails and AFR_AUDIT_REOPEN_LOG returns an internal return code, most likely:

  5 Access denied (RPAUDIT. extension is in use).
  32 Sharing violation (RPAUDIT.LOG is in use).

Should this occur, SA IOM continues to use RPAUDIT.LOG and AFR_AUDIT_REOPEN_LOG records the rename failure in the log. In no case does SA IOM "lose" a message.

SA IOM includes a sample REXX program, MONTHLY.REX, that invokes AFR_AUDIT_REOPEN_LOG to rename the log file on the first day of the month. For descriptions of this and other sample programs, see the appendix, "SA IOM Sample Scripts" in the System Automation for Integrated Operations Management User's Guide.

AFR_ANSI

Use this function to enable or to disable ANSI support, or to return the current value of the ANSI setting.

Format

AFR_ANSI(setting) or AFR_ANSI(' ')

setting The setting parameter accepts one of the following values:

  '1' or 'Y' enables ANSI support.
  '0' or 'N' disables ANSI support.

Note: The second form, which passes a single blank character as an argument, returns the current ANSI setting.

Returns

This function returns the current value of the ANSI setting.

1 indicates that ANSI support is enabled.

0 indicates that ANSI support is disabled.

Example

/* REXX */
n = afr_comctrl(Handle,1,1) /* set flow control of comm port */
ansiSave = afr_ansi(' ') /* save current setting */
n = afr_ansi('0')

See also

"ANSI escape sequence support" on page 3

AFR_ASYNC_DELETE

Use this function to delete the trap set by AFR_ASYNC_WTOR.
AFR_ASYNC_DELETE

Format

AFR_ASYNC_DELETE(trap)

trap This parameter identifies the trap to be deleted. It is a unique identifier returned by the AFR_ASYNC_WTOR function.

Returns

0 The trap deleted successfully.
1 The trap does not exist or there is an error.
-1 The trap has already been cleared.

Example

RET = AFR_ASYNC_DELETE(trap)

See also

"AFR_ASYNC_WTOR"

AFR_ASYNC_WTOR

Use this function to start a REXX script when a WTOR is found.

Format

AFR_ASYNC_WTOR(target,begin,end,scriptname,timeout)

target This parameter specifies a string to search for within the WTORs on the currently active emulation session. The string must be enclosed in quotes. Note: The emulation session must be defined as an MVS console or an MVS console/beeper in order for this target string to be found.

begin This parameter specifies the beginning of the search area in the current emulation session’s buffer. The size of the buffer depends on the screen size of the selected emulator. If your emulation session screen size is model 2, the session buffer spans 2000 characters in the range 0-1999 (80 characters per row, times 25 rows). If your emulation session screen size is model 4, the session buffer spans 3520 characters in the range 0-3519 (80 characters per row, times 44 rows).

dend This parameter specifies the end of the search area in the current emulation session’s buffer. The session buffer spans either 2000 characters in the range 0-1999 or 3520 characters in the range 0-3519.

scriptname This parameter specifies the name of a REXXX program executed when a WTOR containing the target string is found. The entire WTOR line is passed as the argument to this REXXX script. The script name must be enclosed in quotes.

timeout This parameter specifies the time in milliseconds that the function will wait for the trap to occur. The maximum value for this parameter is 2 ^ 31 - 1 milliseconds. If -1 is specified, the wait is indefinite.

Returns

Upon success, this function returns a unique identifier (also called a “handle”) that identifies the trapped data item.
AFR_ASYNC_WTOR

≥0  A handle that identifies the trap.
-1  An error occurred.
-2  Invalid parameter.

Remarks

The function AFR_ASYNC_WTOR captures one occurrence of the specified WTOR. The function must be called again after that WTOR has been answered if subsequent occurrences of that same WTOR are to be captured.

Once the WTOR is found, the indicated REXX script is started and is passed the entire WTOR line as its argument or parameter. The script can then parse the argument for whatever information is needed.

If the program calling AFR_ASYNC_WTOR terminates before the WTOR is found, the indicated REXX script is never started.

The function AFR_ASYNC_WTOR is terminated when any of the following conditions occur:
- timeout time expires
- WTOR is found and scriptname starts and finishes
- script with the AFR_ASYNC_WTOR call terminates

Example

handle = AFR_ASYNC_WTOR('!AOBEEP',0,1999,'script1.rex',6000)

The unique handle returned by this function is assigned to a variable called handle. In this example, if a WTOR containing !AOBEEP is found within 6 seconds, the REXX program SCRIPT1.REX is started with the full WTOR line as its argument.

See also

"AFR_SELECT" on page 87
"AFR_ASYNC_DELETE" on page 18

AFR_BROADCAST

Use this function to display a pop-up text message to all users and send the message to the server log.

Format

AFR_BROADCAST(string,color)

string  This parameter specifies the characters to be written. They must be enclosed in quotes. The maximum length is 240 characters.
color   This parameter specifies the color of the pop-up message box:
0       red
1       green
2       yellow
AFR_BROADCAST

Returns

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The string is successfully logged.</td>
</tr>
<tr>
<td>1</td>
<td>There is an error.</td>
</tr>
</tbody>
</table>

Remarks

This function places the specified string in the server log and on the screens of all connected users in a pop-up message box.

Example

```r
RET = AFR_BROADCAST('testing',0)
```

See also

“AFR_ERROR” on page 41
“AFR_LOG” on page 53
“AFR_OP_ERROR” on page 66

AFR_CHECK_TRAP

Use this function to examine the status of a trap that was previously set by an AFR_SET_TRAP function.

Format

```
AFR_CHECK_TRAP(trap,timeout)
```

- **trap**
  - This parameter identifies the trap to be checked. It is a unique identifier returned by the AFR_SET_TRAP function.

- **timeout**
  - This parameter specifies the time, in milliseconds, to continue to check for the trap to occur. If timeout is equal to zero, the trap is only checked once. The maximum value for this parameter is $2^{31}-1$ milliseconds.

Returns

If the specified trap has been satisfied, this function returns the string placed in memory by the function described in “AFR_SET_TRAP” on page 96.

This function returns a null string if the trap has not yet been satisfied.

If the specified trap is not active, a REXX syntax error results rather than an error return code being returned.

Remarks

This function immediately checks the specified trap then re-checks every two seconds until either a non-null string is returned or the timeout expires.

Once this function returns a non-null string, the specified trap is cleared and must be reset with another AFR_SET_TRAP function.

Example

```r
trap = AFR_SET_TRAP('AOP3001',0,1999,13,9)
RET = AFR_CHECK_TRAP(trap,0)
```
AFR_CHECK_WTOR

Use this function to examine the status of a WTOR that was previously set by an AFR_SET_WTOR function.

Format

AFR_CHECK_WTOR(trap,timeout)

trap This parameter identifies the WTOR to be checked. It is a unique identifier returned by the AFR_SET_WTOR function.

timeout This parameter specifies the time, in milliseconds, to continue to check for the WTOR to occur. If timeout is equal to zero, the WTOR is only checked once. The maximum value for this parameter is $2^{31}$-1 milliseconds.

Returns

If the specified trap has been satisfied, this function returns the string placed in memory by the function described in "AFR_SET_WTOR" on page 101. The string returned starts at the beginning of the line that the WTOR appears on.

This function returns a null string if the trap has not yet been satisfied.

If the specified trap is not active, a REXX syntax error results rather than an error return code being returned.

Remarks

The AFR_CHECK_WTOR function allows you to trap WTORs on the MVS console. This function checks WTORs for the string that you specified with the AFR_SET_WTOR function.

This function immediately checks the specified trap then checks again every two seconds until either a non-null string is found or the until timeout expires. If the string is found, the entire 80-character WTOR line is returned.

Once this function returns a non-null string, the specified trap is cleared and must be reset with another AFR_SET_WTOR function.

The sample program WTORSCAN.REX, included with the product media, is an example of how you can use this and related SA IOM functions in REXX scripts that you write.

Example

RET = AFR_CHECK_WTOR(trap,0)

This example returns the string which is used in the example for AFR_SET_WTOR:

!AOBEEP T9,1-818-555-1211

See also

"AFR_SET_WTOR" on page 101
AFR_CLEAR_TRAP

Use this function to clear a trap that was previously set by an AFR_SET_TRAP function.

Format

AFR_CLEAR_TRAP(trap)

trap This parameter identifies the trap to be cleared. It is a unique identifier returned by the AFR_SET_TRAP function.

Returns

0 The trap is successfully cleared.
1 The trap does not exist or there is an error.
-1 The trap has already been cleared.

Remarks

A trap is also cleared when

• AFR_CHECK_TRAP is successful. If you are using AFR_CHECK_TRAP to look for a text string, you can omit AFR_CLEAR_TRAP from your program.
• AFR_SELECT is used successfully.

Example

RET = AFR_CLEAR_TRAP(trap)

See also

"AFR_SET_TRAP" on page 96

AFR_CLEAR_WTOR

Use this function to clear a WTOR that was previously set by an AFR_SET_WTOR function.

Format

AFR_CLEAR_WTOR(trap)

trap This parameter identifies the trap to be cleared. It is a unique identifier returned by the AFR_SET_WTOR function.

Returns

0 The trap is successfully cleared.
1 The trap does not exist or there is an error.
-1 The trap has already been cleared.

Remarks

A trap is also cleared when
AFR_CLEAR_WTOR

- AFR_CHECK_WTOR is successful. If you are using AFR_CHECK_WTOR to look for a WTOR, you can omit AFR_CLEAR_WTOR from your program.
- AFR_SELECT is used successfully.

Example

RET = AFR_CLEAR_WTOR(trap)

See also

“AFR_SET_WTOR” on page 101

AFR_CLS

Use this function to clear the output screen for the REXX program.

Format

AFR_CLS()

Returns

0 The output screen is successfully cleared.
1 There is an error.

Example

RET = AFR_CLS()

AFR_CNTL_TAPI_START

Use this debugging function to initialize the SA IOM server's TAPI usage after stopping it with AFR_CNTL_TAPI_STOP.

Format

AFR_CNTL_TAPI_START()

This function accepts no parameters.

Returns

This function returns 0.

Remarks

This is a debug function to be used after AFR_CNTL_TAPI_STOP to re-initialize TAPI.

Example

RET = AFR_CNTL_TAPI_STOP()
RET = AFR_SUSPEND(3000)
RET = AFR_CNTL_TAPI_START()

See also

“AFR_CNTL_TAPI_STOP” on page 25
AFR_CNTL_TAPI_STOP

Use this debugging function to stop the SA IOM server’s TAPI usage.

Format

AFR_CNTL_TAPI_STOP()

Returns

This function returns 0.

Remarks

This is a debug function to be used before AFR_CNTL_TAPI_START to re-initialize TAPI.

Example

RET = AFR_CNTL_TAPI_STOP()
RET = AFR_SUSPEND(3000)
RET = AFR_CNTL_TAPI_START()

See also

AFR_CNTL_TAPI_START* on page 24

AFR_COMBREAK

Use this function to generate a break signal on the specified serial communications port.

Format

AFR_COMBREAK(handle)

handle This parameter identifies the serial communications port. It is a unique handle returned by the AFR_COMOPEN function.

Returns

0 The break signal was successfully generated.
1 There is an error (the port is not available).

Remarks

This function will work even if the specified port is currently in use by another function or program, so use it with caution.

Note: This function does not break a connection. If you want to hang up your modem connection, use AFR_COMCTRL or AFR_MODEMDISCONNECT.

Example

RET = AFR_COMBREAK(handle)
AFR_COMCLOSE

Use this function to deallocate a serial communications port that was previously
opened by an AFR_COMOPEN function.

Format

AFR_COMCLOSE(handle)

*handle* This parameter identifies the serial communications port. It is a unique handle
returned by the AFR_COMOPEN function.

Returns

0 The port is already closed or there is an error.

\(n\) The indicated port number is successfully closed. Valid values for \(n\) range from
1 to 32. (The actual number of serial communication ports depends on the
number of serial port expansion adapters installed on the server.)

Remarks

This function will work even if the specified port is currently in use by another
function or program, so use it with caution.

Example

RET = AFR_COMCLOSE(handle)

See also

*AFR_COMOPEN* on page 33

AFR_COMCTRL

Use this function to set the flow control of the specified serial communications port.

Format

AFR_COMCTRL(handle,dtr,rts)

*handle* This parameter identifies the serial communications port. It is a unique handle
returned by the AFR_COMOPEN function.

*dtr* This parameter must be one of the following:

\(0\) forces Data Terminal Ready (DTR) off

\(1\) sets DTR on

*rts* This parameter must be one of the following:

\(0\) forces Request To Send (RTS) off

\(1\) sets RTS on
AFR_COMCTRL

Returns

0  The flow control values were successfully set.
1  There is an error.

Remarks

This function will work even if the specified port is currently in use by another
function or program, so use it with caution.

If the specified serial port is configured for flow control type RTS/CTS, this function
will only work for DTR signal switches and the RTS setting will be ignored.

Example

RET = AFR_COMCTRL(handle, 1, 1)

See also

"AFR_COMOPEN" on page 33

AFR_COMGETBAUD

Use this function to return the current baud rate of the specified serial port.

Format

AFR_COMGETBAUD(handle)

handle  This parameter identifies the serial communications port. It is a unique handle
         returned by the AFR_COMOPEN function.

Returns

This function returns the current baud rate, or 0 if the specified port is not available.

Example

RET = AFR_COMGETBAUD(handle)

See also

"AFR_COMOPEN" on page 33
"AFR_COMSETBAUD" on page 35

AFR_COMGETBITS

Use this function to return the current number of databits for the specified serial
port.

Format

AFR_COMGETBITS(handle)

handle  This parameter identifies the serial communications port. It is a unique handle
         returned by the AFR_COMOPEN function.
AFR_COMGETBITS

Returns
This function returns the current number of databits, either 7 or 8, or it returns 0 if the specified port is not available.

Example
RET = AFR_COMGETBITS(handle)

See also
"AFR_COMOPEN" on page 33
"AFR_COMSETLINEC" on page 36

AFR_COMGETPARITY

Use this function to return the current type of parity for the specified serial port.

Format
AFR_COMGETPARITY(handle)

handle This parameter identifies the serial communications port. It is a unique handle returned by the AFR_COMOPEN function.

Returns
This function returns a single character for the current type of parity. It can be:

E Even parity
N No parity
O Odd parity
M Mark parity
S Space parity
? Unknown (the port is not available)

Example
RET = AFR_COMGETPARITY(handle)

See also
"AFR_COMOPEN" on page 33
"AFR_COMSETLINEC" on page 36

AFR_COMGETSTOPBITS

Use this function to return the current number of stopbits for the specified serial port.

Format
AFR_COMGETSTOPBITS(handle)

handle This parameter identifies the serial communications port. It is a unique handle returned by the AFR_COMOPEN function.
AFR_COMGETSTOPBITS

Returns
This function returns the current number of stopbits, either 1 or 2, or it returns 0 if the specified port is not available.

Example
RET = AFR_COMGETSTOPBITS(handle)

See also
"AFR_COMOPEN" on page 33
"AFR_COMSETLINEC" on page 36

AFR_COMGETSTR

Use this function to retrieve the specified number of bytes from the specified serial communications port input buffer.

Format
AFR_COMGETSTR(handle,bytes)

handle
This parameter identifies the serial communications port. It is a unique handle returned by the AFR_COMOPEN function.

bytes
This parameter specifies, in bytes, the length of the input buffer string retrieved. It can be the value returned by the AFR_COMINPUTBYTES function.

Returns
A string of the length specified is returned.

On error, this function returns a NULL.

Remarks
This function will work even if the specified port is currently in use by another function or program, so use it with caution.

Before using AFR_COMGETSTR, you can use AFR_COMINPUTBYTES to find how many bytes are available, as shown in the following example.

Example
bytes = AFR_COMINPUTBYTES(handle)
if (bytes > 0) then
    string = AFR_COMGETSTR(handle,bytes)

See also
"AFR_COMOPEN" on page 33
"AFR_COMOUTPUTBYTES" on page 34

AFR_COMINPUTBYTES

Use this function to return the number of bytes currently in the specified serial communication port’s input buffer. You retrieve these bytes using the function described in “AFR_COMGETSTR.”
AFR_COMINPUTBYTES

Format

AFR_COMINPUTBYTES(handle)

handle This parameter identifies the serial communications port. It is a unique handle returned by the AFR_COMOPEN function.

Returns

The function returns the number of bytes in the serial port's input buffer.

Returns 0 if no bytes or an error occurred.

Remarks

This function will work even if the specified port is currently in use by another function or program, so use it with caution.

Example

RET = AFR_COMINPUTBYTES(handle)

See also

"AFR_COMOPEN" on page 33
"AFR_COMGETSTR" on page 29

AFR_COMJOURNALSTART

Use this function to start a journal file of the specified serial communication port.

Format

All parameters are positional. If any of the optional parameters are omitted, commas must indicate the proper position. The journal file is automatically closed and the journal stopped if the COM port being journaled is closed.

AFR_COMJOURNALSTART(handle[,type][,fileinfo][,autostop1])
AFR_COMJOURNALSTART(handle[,type][,fileinfo][,autostop1][,autostop2])

handle This parameter identifies the serial communications port. It is a unique handle returned by the AFR_COMOPEN function.

Type This optional parameter identifies the type of journal entries to produce.

'IO' produces records with IxIxIx as input characters followed by the actual hexadecimal character read from the communications port. Output records have the format OxOxOx. 'IO' is the default.

'HEX' produces records with a time stamp type (input or output) block address and length, followed by a hexadecimal dump format of the input or output data read or written to the communications port.
fileinfo

This optional parameter identifies how to handle existing journal files.

'A' appends the new journal information to an existing journal file. 'A' is the default.

'N' creates a new journal file. Any existing journal file is deleted before the new file is created. If you append to an existing file and stop after xxx records, the file will probably contain more than xxx records since existing records are not counted. If you append to an existing file and stop after xxx bytes when xxx bytes of input or output data have been added to the file the journal is stopped. The journal may contain MORE than xxx bytes.

autostop1

When autostop parameters are used, maxbytes refers to the size of the input/output transmit data sizes, not the file size. The file size can be double or more the requested size depending on the type of output requested. Also, maxrecs and maxbytes are for each open of the journal file. This protects against loss of data if SA IOM restarts and you request journaling with autostop parameters to journal activity during startup. Since each start has a timestamp the starts can be separated easily.

These parameters can be specified in pairs or alone as autostop1. If specified in pairs, the first parameter must be either size in bytes or record count. The second value, autostop2, is the time limit in minutes.

'maxbytes=xxxM|xxxK|xxx'

where xxxM is xxx million bytes, xxxK is xxx thousand bytes (1000), or xxx is bytes.

'maxrecs=xxxM|xxxK|xxx'

where xxxM is xxx million records, xxxK is xxx thousand records (1000), or xxx records.

'time=MMM'

where MMM is mmm minutes.

The default is: autostop is not checked and the file size is not limited.

Returns

This function returns a single character for the current type of parity. If journaling errors occur during the actual writing of the journal file, they are logged in the server log and the journal is stopped. This function will work even if the specified handle (port) is currently in use by another function or program, so use it with caution.

0   The journal was started.
1   The journal could not be started because the handle is not open.
2   The journal could not be started because the type is incorrect.
3   The journal could not be started because the fileinfo is incorrect.
4   The journal could not be started because autostop1 is incorrect.
5   The journal could not be started because autostop2 is incorrect.
7   The journal file could not be opened.
8   The journal file is already active.
AFR_COMJOURNALSTART

**Remarks**

Journaling provides low-level serial communication data traffic information that you can use as a diagnostic aid when working with serial ports. To start journaling, `AFR_COMJOURNALSTART` must be called from your REXX application after the `AFR_COMOPEN` call has completed. Or, it must be called only for a serial port opened by `AFR_COMOPEN`.

**Note:** Journaling can be voluminous so should not be used regularly. It cannot be used to diagnose errors in SA IOM Client/Server connections.

**Examples**

In the following example, journal COM1 is in 'IO' format and is appended to an existing journal. Journaling stops when the size reaches 50000000 bytes or two hours expires.

```
AFR_COMJOURNALSTART(1,,,,'maxbytes=50M','time=120')
```

In the next example, journal COM1 is in hex format and is appended to an existing journal. Journaling stops after 30 minutes.

```
AFR_COMJOURNALSTART(1,'HEX','A','time=30')
```

Shown next, the journal COM3 is in hex format. A new journal is created, deleting any existing journal file for COM3. Journaling stops when 10000 records have been written.

```
AFR_COMJOURNALSTART(3,'HEX','N','maxrecs=10K')
```

The next example journal COM4 is in 'IO' format and is appended to an existing journal. Journaling stops only when `AFR_COMJOURNALSTOP` is called.

```
AFR_COMJOURNALSTART(4)
```

**See also**

- "AFR_COMCLOSE" on page 26
- "AFR_COMJOURNALSTOP"
- "AFR_COMOPEN" on page 33

AFR_COMJOURNALSTOP

Use this function to stop a journal file of the specified serial communication port.

**Format**

```
AFR_COMJOURNALSTOP(handle)
```

**handle**

This parameter identifies the serial communications port. It is a unique handle returned by the `AFR_COMOPEN` function.

**Returns**

0

The journal was stopped.

1

The journal was not stopped. The handle is invalid, the port is not open, or the journal has already been stopped.
**AFR_COMJOURNALSTOP**

**Remarks**

Use this function to stop journaling for a serial communication port that was started with AFR_COMJOURNALSTART.

**Example**

In the following example the journal on COM1 is stopped.

AFR_COMJOURNALSTOP(1)

**See also**

"AFR_COMCLOSE" on page 26
"AFR_COMJOURNALSTART" on page 30
"AFR_COMOPEN"

---

**AFR_COMOPEN**

Use this function to allocate a serial communications port for use by the AFR_MODEM functions.

**Format**

AFR_COMOPEN(request)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>request</td>
<td>This parameter either searches for or specifies, the port to use for communications.</td>
</tr>
</tbody>
</table>

- **0**  
  Tells this function to look for any available serial communications port configured for script use. This is the recommended method.  
  **Note:** The AFR_MODEM functions require a serial communication port configured for script use.

- If you want, you can allocate a specific port (request it by number). Valid values for **n** range from 1 to 32. (The actual number of serial communication ports depends on the number of serial port expansion adapters installed on the server.)

**Returns**

This function returns a short string that both identifies the port allocated and serves as a unique handle for subsequent AFR_MODEMCALL, AFR_MODEMDISCONNECT, AFR_MODEMANswer, and AFR_MODEMSTATUS functions, and for subsequent AFR_COM functions.

- **0**  
  No port is allocated.

- **n**  
  The corresponding port (the one you requested above) is successfully allocated.

**Remarks**

Note that when using values other than 0 for a port request, you can allocate a port that may already be in use by another function or program. To be sure you are allocating an unused port, use the value 0. This allows you to have several programs using the same port, but it also requires you to be careful that your programs don't get in each other's way.
AFR_COMOPEN

Example

```python
handle = AFR_COMOPEN(0)
```

The unique handle returned by the function is assigned to a variable called `handle`.

AFR_COMOUTPUTBYTES

Use this function to return the number of unused bytes currently available in the specified serial communication port's output buffer. You add bytes to this buffer using the function described in AFR_COMPUTSTR.

Format

```python
AFR_COMOUTPUTBYTES(handle)
```

`handle` - This parameter identifies the serial communications port. It is a unique handle returned by the AFR_COMOPEN function.

Returns

This function returns the number of bytes currently available in the serial port's output buffer.

Remarks

This function will work even if the specified port is currently in use by another function or program, so use it with caution.

Example

```python
RET = AFR_COMOUTPUTBYTES(handle)
```

See also

"AFR_COMOPEN" on page 33
"AFR_COMPUTSTR"

AFR_COMPUTSTR

Use this function to write the specified string to the specified serial communications port output buffer.

Format

```python
AFR_COMPUTSTR(handle,string,crfeed)
```

`handle` - This parameter identifies the serial communications port. It is a unique handle returned by the AFR_COMOPEN function.

`string` - This parameter specifies the characters to be written. They must be enclosed in quotes.

`crfeed` - This parameter specifies whether (1) or not (0) to append an ASCII carriage return character (CR) to the end of the string.
AFR_COMPUTSTR

**Returns**
This function returns the number of characters successfully written to the buffer.
Returns 0 or short byte count if errors occur.

**Remarks**
The maximum size of the communications port output buffer is 8K.
This function will work even if the specified port is currently in use by another function or program, so use it with caution.
Before writing to a port output buffer, check if space is available by using AFR_COMOUTPUTBYTES.

**Example**
```
string = "This is a test string"
size = length(string)
output_bytes = AFR_COMOUTPUTBYTES
if (size < output_bytes) then
    ret = AFR_COMPUTSTR(handle,string,0)
```

**See also**
"AFR_COMOPEN" on page 33
"AFR_COMOUTPUTBYTES" on page 34

AFR_COMSETBAUD

**Use this function to set the baud rate of the specified serial communications port.**

**Format**
```
AFR_COMSETBAUD(handle,baud)
```

**handle**
This parameter identifies the serial communications port. It is a unique handle returned by the AFR_COMOPEN function.

**baud**
This parameter must be one of the following numbers:
- 1200
- 9600
- 14400
- 19200
- 38400
- 57600
- 115200

**Returns**
This function returns either the baud rate (one of the above) of the successfully set port or 0 if a failure occurred.

**Remarks**
This function will work even if the specified port is currently in use by another function or program, so use it with caution.

**Example**
```
RET = AFR_COMSETBAUD(handle,1200)
```
AFR_COMSETLINEC

Use this function to set the line characteristics of the specified serial communications port.

Format

AFR_COMSETLINEC(handle,databits,stopbits,parity)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>This parameter identifies the serial communications port. It is a unique handle returned by the AFR_COMOPEN function.</td>
</tr>
<tr>
<td>databits</td>
<td>This parameter specifies the number of databits, 7 or 8.</td>
</tr>
<tr>
<td>stopbits</td>
<td>This parameter specifies the number of stopbits, 1 or 2.</td>
</tr>
<tr>
<td>parity</td>
<td>This parameter specifies the type of parity:</td>
</tr>
<tr>
<td></td>
<td>E</td>
</tr>
<tr>
<td></td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>S</td>
</tr>
</tbody>
</table>

Returns

0       The line characteristics were successfully set.
1       There is an error.

Remarks

This function will work even if the specified port is currently in use by another function or program, so use it with caution.

Example

RET = AFR_COMSETLINEC(handle,8,1,'N')

See also

"AFR_COMOPEN" on page 33
"AFR_COMGETBAUD" on page 27
"AFR_COMGETBITS" on page 27
"AFR_COMGETPARITY" on page 28
"AFR_COMGETSTOPBITS" on page 28

AFR_COMSTATUS

Use this function to get the status of various output and input lines for the specified serial communications port.
AFR_COMSTATUS

Format

AFR_COMSTATUS(handle)

handle This parameter identifies the serial communications port. It is a unique handle returned by the AFR_COMOPEN function.

Returns

The function returns 0 if the port is not available.

Otherwise, the returned number is a single byte value ranging from 0 to 255. Break down the number to determine which bits are on, then refer to the following table to find out which lines have been asserted:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>Line</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>DTR</td>
<td>Output</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>RTS</td>
<td>Output</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>CTS</td>
<td>Input</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>DSR</td>
<td>Input</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>RI</td>
<td>Input</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>DCD</td>
<td>Input</td>
</tr>
</tbody>
</table>

Remarks

This function will work even if the specified port is currently in use by another function or program, so use it with caution.

Example

RET = AFR_COMSTATUS(handle)

A return value of 35 would be 00100011 in binary, meaning DTR, RTS, and DSR asserted.

AFR_CONSOLE_TYPE

Use this function to return a string designating the console type of the active emulation session.

Format

AFR_CONSOLE_TYPE()

Remarks

The desired emulation session must have been previously made active with the function described in "AFR_SELECT" on page 87.

The string returned by this function identifies the exact configuration of the emulation session defined on the SA IOM server. However, the string does not in all cases exactly match the information listed on the Host Sessions page of the Server Configuration Properties window. The string returned is a mapping of the
AFR_CONSOLE_TYPE

values defined for **Type** (in the second configuration dialog) and for **Usage** (in the third configuration dialog). For example, a session of type "MVS console" can be configured either with or without the option "/Beeper" which was presented as a checkbox in the third configuration dialog when the session was defined. So the string returned can be "MVS Console" or "MVS Console/Beeper" depending on how the session was defined.

**Example**

```plaintext
RET = AFR_CONSOLE_TYPE()
```

This example returns the string:

MVS Console/Beeper

**See also**

"AFR_SELECT" on page 87

AFR_CURSOR

Use this function to return the cursor position in the active screen.

**Format**

```plaintext
AFR_CURSOR()
```

**Returns**

This function returns the cursor position as a number from 0-1999 or from 0-3519. The size of the buffer depends on the screen size of the selected emulator. If your emulation session screen size is 3270 model 2, the session buffer spans 2000 characters in the range 0-1999 (80 characters per row, times 25 rows).

If your emulation session screen size is 3270 model 4, the session buffer spans 3520 characters in the range 0-3519 (80 characters per row, times 44 rows).

AFR_CURSOR will return -1 if you did not first select the session with AFR_SELECT.

**Remarks**

The desired emulation session buffer must have been previously made active with the function described in "AFR_SELECT" on page 87.

**Example**

```plaintext
RET = AFR_CURSOR()
```

**See also**

"AFR_SELECT" on page 87

AFR_EMIT_UA_SNMP_TRAP

Use this function only to send an SNMP trap from SA IOM to the IBM Tivoli Universal Agent. Ignore this function if you do not have a correctly configured IBM Tivoli Universal Agent in your environment.
AFR_EMIT_UA_SNMP_TRAP

Format

AFR_EMIT_UA_SNMP_TRAP(destination, severity, category, message)

destination  TCP/IP name of the target SNMP Manager host name (defaults to local machine).
severity      SNMP Trap message severity, standard defined severities are:
              "Cleared"
              "Indeterminate"
              "Warning"    (default)
              "Minor"
              "Critical"
              "Major"
category      SNMP Trap message category, standard defined categories are:
              "Threshold_Events"
              "Network_Topology_Events"
              "Error_Events"
              "Status_Events"
              "Node_Configuration_Events"
              "Application_Aler_Event"  (default)
              "All_Category_Events"
              "Log_Only_Events"
              "Map_Events"
              "Ignore_Events"
message       The free form content of the SNMP trap alert message text.

Returns

0         OK
-1        Could not load the DLL (KUMS1API.DLL or a DLL it needs is probably missing).
-2        Emit function not found (probably wrong version of KUMS1API.DLL).
-5        Emit call attempted while SA IOM server shutting down.
-10       Reserved (should never see this).
other     Any other value as returned from the emit function in KUMS1API.DLL.

Not passing four parameters will cause the standard REXX error 40.

Remarks

The IBM Tivoli Universal Agent must be configured.

Beware of firewall software (such as Zone Alarm) blocking the request on either or both the sending or receiving machines.

Notes:

1. The defaults values listed for Destination and Category (above) depend upon which Tivoli Universal Agent DLL you are using.
2. The parameters of the AFR_EMIT_UA_SNMP_TRAP function are only displayed in the "value list" section and not elsewhere in the SNMP Manager Trap panel of a Tivoli Enterprise Portal (TEP).

Example

RET = AFR_EMIT_UA_SNMP_TRAP(“host ip name”, “Critical”, ”Error_Events”, ”Oops: ICP failed”)
AFR_EMIT_UA_SNMP_TRAP

See also
IBM Tivoli Universal Agent documentation.

AFR_EMAIL_LOGSIZE
Use this function to retrieve the number of bytes in the e-mail log.

Format
AFR_EMAIL_LOGSIZE()
This function accepts no parameters.

Returns
This function returns the number of bytes in the e-mail log.

Example
Num = AFR_EMAIL_LOGSIZE()

AFR_EMAIL_REOPEN_LOG
Use this function to reallocate the current e-mail log under a specified file name extension and start a fresh log file.

When you call AFR_EMAIL_REOPEN_LOG:
1. The current RPEMAIL. extension (if present) is deleted.
2. RPEMAIL.LOG is closed.
3. RPEMAIL.LOG is named to RPEMAIL. extension.
4. A new, empty RPEMAIL.LOG is opened for recording e-mail messages.

Format
AFR_EMAIL_REOPEN_LOG(extension)

extension Up to 3 characters used as a file name extension when renaming the current RPEMAIL.LOG file. If you omit this parameter, the default extension .BAK is used.

Returns
0 The log file was reallocated successfully.
1 The specified extension is invalid.

Remarks
- When the renaming process is successful, the last message in the old log file (now renamed as RPEMAIL. extension) and the first message in the new RPEMAIL.LOG file records the successful reallocation of the log.
- If either RPEMAIL.LOG or RPEMAIL. extension is in use by another process when this function is called, the renaming process fails and AFR_EMAIL_REOPEN_LOG returns an internal return code, most likely:
  5 Access denied (RPEMAIL. extension is in use).
Sharing violation (RPEMAIL.LOG is in use).

Should this occur, SA IOM continues to use RPEMAIL.LOG and AFR_EMAIL_REOPEN_LOG records the rename failure in the log. In no case does SA IOM “lose” a message.

SA IOM includes a sample REXX program, MONTHLY.REX, that invokes AFR_EMAIL_REOPEN_LOG to rename the log file on the first day of the month. For descriptions of this and other sample programs, see the appendix, “SA IOM Sample Scripts” in the System Automation for Integrated Operations Management User's Guide.

AFR_ERROR

Use this function to send a pop-up text message to the REXX script owner and send the message to the server log.

Format

AFR_ERROR(string,color)

string This parameter specifies the characters to be written. They must be enclosed in quotes. The maximum length is 240 characters.

color This parameter specifies the color of the pop-up message box:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>red</td>
</tr>
<tr>
<td>1</td>
<td>green</td>
</tr>
<tr>
<td>2</td>
<td>yellow</td>
</tr>
</tbody>
</table>

Returns

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The string is successfully logged.</td>
</tr>
<tr>
<td>1</td>
<td>There is an error.</td>
</tr>
</tbody>
</table>

Remarks

This function places the specified string in the server log and in a pop-up message box on your (the script owner’s) client screen. You can use this function to send a message to yourself. This can be useful when developing scripts.

If the script owner is not logged on at the time the message is generated, the message is written to the log only.

Example

RET = AFR_ERROR('SYS A IPL INITIATED',0)

See also

*AFR_BROADCAST* on page 20
*AFR_LOG* on page 53
*AFR_OP_ERROR* on page 66
AFR_GET_LINE

Use this function to retrieve a line of data from an active Glass Teletype or Message Collector session.

Format

AFR_GET_LINE()

Returns

This function returns a line of characters from the Glass Teletype and Message Collector sessions' buffers. The line is retrieved when a carriage return is detected. The next AFR_GET_LINE call will retrieve the next line.

AFR_GET_LINE returns a null string if a complete line (up to the carriage return) is not available when the function is called or if you did not first select the session with AFR_SELECT.

Remarks

Because of the scrollable nature of these sessions, SA IOM provides these sessions with an 80K wraparound collection buffer. This prevents loss of data before it can be viewed or trapped. Use the AFR_GET_LINE function to retrieve data from this buffer.

The desired session must have been previously made active with the function described in "AFR_SELECT" on page 87.

Example

```
line = AFR_GET_LINE()
if substr(line,1,7) = 'IEF1234' then ...
```

This program fragment will retrieve one line of data from the active session buffer and search for the string IEF1234.

See also

"AFR_SELECT" on page 87
"AFR_SEND_MESSAGE" on page 94
"AFR_MC_LASTMSGTEXT" on page 55

AFR_GET_STATUS

Use this function to get the status of the active emulation session.

Format

AFR_GET_STATUS(option, offset, length)

All parameters are optional and the typical use is to call the function without parameters.

option

Must be specified if offset is specified. The only valid value is 1.

offset

Must be specified if length is specified. The parameter specifies an offset into the emulation screen. Valid values are between 0 and the size of the screen minus any length.
AFR_GET_STATUS

Returns

The function returns a string with one of the following possible screen status indicators. If offset was specified, the string will also have the screen characters starting at offset into the screen for length characters if the indicator begins with "AVAILABLE-" appended after "::" characters.

- "UNAVAILABLE-NOTSELECTED"
- "UNAVAILABLE-DISCONNECTED"
- "UNAVAILABLE-CONNECTING"
- "UNAVAILABLE-NEGOTIATING"
- "UNAVAILABLE-FAILED"
- "UNAVAILABLE-ALLOCATING"
- "UNAVAILABLE-STARTING"
- "UNAVAILABLE-STOPPING"
- "UNAVAILABLE-STOPPED"
- "UNAVAILABLE-INVALID"
- "AVAILABLE-RUNNING"
- "AVAILABLE-KEYBOARDLOCKED"
- "AVAILABLE-KEYBOARDUNLOCKED"

Remarks

The desired emulation session buffer must have been previously made active with the AFR_SELECT() function, see "AFR_SELECT" on page 87.

Example

```
RET = AFR_GET_STATUS()
```

See also

"AFR_LOAD" on page 50
"AFR_SELECT" on page 87

AFR_GETENV

Use this function to return the value of a server configuration data item.

Format

```
AFR_GETENV(variable)
```

variable A descriptive name that identifies the server configuration data item. See the following table.

Returns

This function returns the value of the server configuration data item that is identified by the specified variable name. See the following table.

This function returns a null string if the variable does not exist, if the name of the variable is invalid, or if the name is good but the variable contains no data.

Remarks

The order in which certain resources become available determines the order in which the corresponding En_TITLE variable is created. This order can vary from
one SA IOM server execution to the next. This is particularly common with
TN3270E sessions, due to network traffic, activity within the TN3270E server, and
so forth.

The order of the configured sessions as listed on the Host Sessions page of the
Server Configuration Properties window is not necessarily the order in which the
variables are returned by the AFR_GETENV function. The order in which variables
are returned can vary from one SA IOM server execution to the next. If the
configuration has changed then the order of elements in the configuration file will be
different.

The string returned by the En_USE variable identifies the exact configuration of the
emulation session defined on the SA IOM server. However, the string does not in
all cases exactly match the information listed on the Host Sessions page of the
Server Configuration Properties window. The string returned is a mapping of the
values defined for Type (in the second configuration dialog) and for Usage (in the
third configuration dialog). For example, a session of type "MVS console" can be
configured either with or without the option "/Beeper" which was presented as a
checkbox in the third configuration dialog when the session was defined. So the
string returned can be "MVS Console" or "MVS Console/Beeper" depending on
how the session was defined.

Example

```plaintext
RET = AFR_GETENV('TA_RATE')
```

<table>
<thead>
<tr>
<th>AFR_GETENV Variable Name</th>
<th>Typical Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Acquisition Values</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TA_PORT</td>
<td>COM1</td>
<td>Time Acquisition COM port</td>
</tr>
<tr>
<td>TA_RATE</td>
<td>38400</td>
<td>Time Acquisition port baud rate</td>
</tr>
<tr>
<td>TA_BITS</td>
<td>8</td>
<td>Time Acquisition port data bits</td>
</tr>
<tr>
<td>TA_PARITY</td>
<td>None</td>
<td>Time Acquisition port parity</td>
</tr>
<tr>
<td>TA_STOP</td>
<td>1</td>
<td>Time Acquisition port stop bits</td>
</tr>
<tr>
<td>TA_PHONE</td>
<td>9,1(303)494-4774</td>
<td>Time Acquisition service phone number</td>
</tr>
<tr>
<td>TA_CALL</td>
<td>06-03-29 12:21</td>
<td>Time Acquisition next date/time to call</td>
</tr>
<tr>
<td></td>
<td>(yy-mm-dd-hh:mm)</td>
<td></td>
</tr>
<tr>
<td>TA_INTERVAL</td>
<td>10 00:00</td>
<td>Time Acquisition interval between calls</td>
</tr>
<tr>
<td></td>
<td>(dd hh:mm)</td>
<td>Here dd indicates elapsed days in the range 1-99.</td>
</tr>
<tr>
<td>TA_AUTOSTART</td>
<td>No</td>
<td>Time Acquisition REXXSTART autostart flag</td>
</tr>
</tbody>
</table>
### Table 2. AFR_GETENV Variables (continued)

<table>
<thead>
<tr>
<th>AFR_GETENV Variable Name</th>
<th>Typical Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA_REXXSTART</td>
<td>TIME.REX</td>
<td>Time Acquisition REXX program to be started by AUTOEXEC.REX if autostart flag = 'Y'</td>
</tr>
<tr>
<td>TA_REXXCALL</td>
<td>TIMECALL.REX</td>
<td>Time Acquisition REXX program that acquires system time</td>
</tr>
</tbody>
</table>

#### Beep Paging Values

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BP_PORT</td>
<td>COM1</td>
<td>Beeper Paging COM port</td>
</tr>
<tr>
<td>BP_RATE</td>
<td>38400</td>
<td>Beeper Paging port baud rate</td>
</tr>
<tr>
<td>BP_BITS</td>
<td>8</td>
<td>Beeper Paging port data bits</td>
</tr>
<tr>
<td>BP_PARITY</td>
<td>E</td>
<td>Beeper Paging port parity</td>
</tr>
<tr>
<td>BP_STOP</td>
<td>1</td>
<td>Beeper Paging port stop bits</td>
</tr>
<tr>
<td>BP_TRIGGER</td>
<td>!AOBEEP</td>
<td>Beeper Paging WTOR trigger pattern</td>
</tr>
<tr>
<td>BP_AUTOSTART</td>
<td>No</td>
<td>Beeper Paging REXXSTART autostart flag</td>
</tr>
<tr>
<td>BP_REXXSTART</td>
<td>BEEP.REX</td>
<td>Beeper Paging REXX program to be started by AUTOEXEC.REX if autostart flag = 'Y'</td>
</tr>
<tr>
<td>BP_REXXCALL</td>
<td>BEEPCALL.REX</td>
<td>Beeper Paging REXX program that sends a beeper page</td>
</tr>
</tbody>
</table>

#### Managed by System

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS_KEYBOARD</td>
<td>101-Key IBM Enhanced</td>
<td>PC keyboard type</td>
</tr>
<tr>
<td>SYS_LANGUAGE</td>
<td>US</td>
<td>PC keyboard language type</td>
</tr>
<tr>
<td>SYS_DATE</td>
<td>06-03-21</td>
<td>Current system date (yy-mm-dd)</td>
</tr>
<tr>
<td>SYS_TIME</td>
<td>08:24:56</td>
<td>Current system time (hh:mm:ss)</td>
</tr>
<tr>
<td>SYS_HOSTNAME</td>
<td>SAIOM1</td>
<td>The server name as entered in the configuration</td>
</tr>
<tr>
<td>SYS_DATETIME</td>
<td>06-03-21 08:24:56</td>
<td>Current system date and time (yy-mm-dd hh:mm:ss)</td>
</tr>
<tr>
<td>SYS_TIMEZONE</td>
<td>480</td>
<td>Current GMT to local time difference in minutes</td>
</tr>
<tr>
<td>SYS_DST</td>
<td>Pacific Daylight Time</td>
<td>Daylight Savings Time</td>
</tr>
<tr>
<td>SYS_VERSION</td>
<td>210</td>
<td>Product version</td>
</tr>
</tbody>
</table>

**Note:** In the following communication port variable names, n is the port number. Valid values for n range from 1 to 32. (The actual number of serial communication ports depends on the number of serial port expansion adapters installed on the server.)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pn_USE</td>
<td>Terminal Emulation</td>
</tr>
<tr>
<td>Pn_CLASS</td>
<td>CLASSB</td>
</tr>
</tbody>
</table>
### AFR_GETENV

**Table 2. AFR_GETENV Variables (continued)**

<table>
<thead>
<tr>
<th>AFR_GETENV Variable Name</th>
<th>Typical Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pn_TITLE</td>
<td>Host SessionB</td>
<td>Name of port-connected host session</td>
</tr>
<tr>
<td>Pn_LANGUAGE</td>
<td>English (U.S.)</td>
<td>Language of the 3270 device connected to the port</td>
</tr>
<tr>
<td>Pn_SCREEN</td>
<td>1920</td>
<td>Screen size of port-connected host session</td>
</tr>
<tr>
<td>Pn_TYPE</td>
<td>3090 Console</td>
<td>Type of host session that port is being used for</td>
</tr>
<tr>
<td>Pn_LINE</td>
<td>Dial-up</td>
<td>Port connection hardware</td>
</tr>
<tr>
<td>Pn_RATE</td>
<td>38400</td>
<td>Port baud rate</td>
</tr>
<tr>
<td>Pn_BITS</td>
<td>8</td>
<td>Port data bits</td>
</tr>
<tr>
<td>Pn_PARITY</td>
<td>N</td>
<td>Port parity</td>
</tr>
<tr>
<td>Pn_STOP</td>
<td>1</td>
<td>Port stop bits</td>
</tr>
<tr>
<td>Pn_FLOW</td>
<td>None</td>
<td>Port flow control type</td>
</tr>
<tr>
<td>Pn_IDLETO</td>
<td>0</td>
<td>Dial-in connect idle timeout, minutes</td>
</tr>
</tbody>
</table>

**Note:** In the following emulator variable names, *n* represents the relative position of the configured host session as it appears in the Host Sessions list of the SA IOM Server Configuration dialog. Valid values for *n* range from 1-9999.

<table>
<thead>
<tr>
<th>En_USE</th>
<th>MVS Console/Beeper</th>
<th>Emulator usage description</th>
</tr>
</thead>
<tbody>
<tr>
<td>En_CLASS</td>
<td>Production Console Class</td>
<td>Name of session class that host session belongs to</td>
</tr>
<tr>
<td>En TITLE</td>
<td>SystemA Console</td>
<td>Name of the configured host session</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> This is the Name value supplied when the Host Session was defined.</td>
</tr>
<tr>
<td>En_LANGUAGE</td>
<td>English (U.S.)</td>
<td>Language of the 3270 host session</td>
</tr>
<tr>
<td>En_SCREEN</td>
<td>3440</td>
<td>Screen size of the host session</td>
</tr>
</tbody>
</table>
| En_ENABLED               | 1                 | Indicates whether or not the configured session is started automatically at server initialization time.  
|                          |                   | A value of 1 indicates yes.                                                |
|                          |                   | A value of 0 indicates no.                                                 |

**See also**

“AFR_SETENV” on page 102

### AFR_GETOWNER

Use this function to retrieve a script owner name for a specified script. You can then restrict user authorization for starting some REXX programs.
AFR_GETOWNER

Format

AFR_GETOWNER([script_id])

script_id  This optional parameter is the SA IOM script ID number assigned to the REXX program. The script ID is displayed after the REXX script name on the Scripts control panel and also on the title bar of an opened script session window. If you call this function without the parameter, the script ID of the currently selected REXX session is used.

Returns

The function returns a character string which is the script owner name.

Remarks

For scripts started by a person, the owner name is the logon ID of the user.

Scripts started by the AFR_START_PROGRAM function inherit the REXX script owner name from the call process.

Example

/* This is a sample program that can only be started by Sysadmin */
authorized_user = "Sysadmin"  /* define an authorized user name */
caller_name = AFR_GETOWNER()  /* get name of the user who */  /* started this program (remotely) */

if (caller_name <> authorized_user) then
  do
    nul = AFR_ERROR("You are not authorized to run this program.",1)
  exit
  end
/* else continue... */

See also

"AFR_ERROR" on page 41
"AFR_START_PROGRAM" on page 107

AFR_GMT

Use this function to set the server PC's clock.

Format

AFR_GMT('gmtadjust date time') or AFR_GMT(gmtadjust,date,time)

gmtadjust  This parameter is the number of minutes (or the number of hours and minutes) your local time is West of GMT. Los Angeles, for example, is 8 hours west of GMT, and so would use a value of either 480 (minutes) or 08:00 (hours) for this parameter.
date  This parameter uses the formats:

  yy-mm-dd or yy/mm/dd. It is the GMT date.
AFR_GMT

**time**

This parameter uses the formats:

- \( hh:mm:ss \) or \( hh:mm \)
- or
- \( hh.mm.ss \) or \( hh.mm \)

It is the GMT.

**Note:** The second form sets time to zero seconds.

The parameters are separated either with spaces or with commas.

**Returns**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The clock is successfully set.</td>
</tr>
<tr>
<td>1</td>
<td>An unspecified error was encountered.</td>
</tr>
</tbody>
</table>

**Remarks**

This function is used by the TIMECALL.REX program to update the system date and time of your PC.

**Example**

```
RET = AFR_GMT('480 98-03-22 14:55:30')
```

AFR_KBHIT

This function is used to retrieve the ASCII value of the current keystroke in the keyboard input buffer.

**Format**

```
AFR_KBHIT()
```

This function accepts no parameters.

**Returns**

This function returns one of the following:

-1 No keyboard input, no key has been pressed.

or, the ASCII value of the key that was pressed.

**Remarks**

This function is useful in monitoring situations where keyboard input is possible, but you do not want a REXX script to stop and wait for keyboard input. Use the AFR_KBHIT function in a script to check if a user has entered any keys.

This function is similar to the KBHIT() function in the C language runtime library.
Example

/* loop forever, getting and sending data */
do forever
   /* if a key has been pressed, grab it and send to the COM port */
   /* (convert the ASCII Value returned by afr_KBHit() to CHAR) */
   key = afr_KBHit()
   if (key > 0) then do
      rc = afr_ComPutStr(handle, d2c(key), 0)
      end /* end key checking */
   end /* end forever loop */
See also
   "AFR_SAY" on page 86
   "ANSI escape sequence support" on page 3

AFR_KEY_ET

Use this function to return the elapsed time, in seconds, since a keystroke was last given to the currently selected emulator.

Format

AFR_KEY_ET()

This function accepts no parameters.

Returns

This function returns
   -1  No emulator is selected.

If an emulator is selected, this function returns
   • the elapsed time in seconds since a keystroke was last given to the selected emulator
   • or, if no keystrokes have ever been sent, the elapsed time in seconds since the server was started.

Remarks

The keystroke may originate from a user selecting the emulator and typing on a keyboard or from another REXX script executing one of the functions that begin with AFR_SEND.

Example

RET = AFR_KEY_ET()
AFR_LOAD

Use this function to return a string from the active emulation session.

Format

```c
AFR_LOAD(offset, length)
```

- **offset**
  - This parameter specifies where the string starts in the buffer. It is specified as a number from 0-1999 or from 0-3519. The size of the buffer depends on the screen size of the selected emulator. If your emulation session screen size is 3270 model 2, the session buffer spans 2000 characters in the range 0-1999 (80 characters per row, times 25 rows). If your emulation session screen size is 3270 model 4, the session buffer spans 3520 characters in the range 0-3519 (80 characters per row, times 44 rows).

- **length**
  - This parameter specifies the length of the string in bytes. Attribute bytes are ignored.

Returns

This function returns a string of the specified offset and length. If there is no active emulation session, a null string is returned.

Remarks

The desired emulation session buffer must have been previously made active with the function described in "AFR_SELECT" on page 87.

Example

```c
RET = AFR_LOAD(1460, 40)
```

This example returns 40 bytes from the middle of row 19, characters 1460-1499, indexed from 0.

See also

"AFR_SELECT" on page 87
"AFR_LOAD_COLOR"
"AFR_SEND_MESSAGE" on page 94
AFR_LOAD_COLOR

offset  This parameter specifies where the string starts in the buffer. It is specified as a number from 0-1999 or from 0-3519. The size of the buffer depends on the screen size of the selected emulator. If your emulation session screen size is 3270 model 2, the session buffer spans 2000 characters in the range 0-1999 (80 characters per row, times 25 rows). If your emulation session screen size is 3270 model 4, the session buffer spans 3520 characters in the range 0-3519 (80 characters per row, times 44 rows).

length  This parameter specifies the length of the string, in bytes.

Returns

This function retrieves the color attribute of the string of the specified offset and length and converts it to a two-digit HEX representation of the foreground and background colors. See the table that follows. If there is no active emulation session, a null string is returned.

Remarks

The desired emulation session buffer must have been previously made active with the function described in "AFR_SELECT" on page 87.

Each text character returns one byte (8 bits) of color attribute information. The first 4 bits (0-3) represent the background color, the second 4 bits (4-7) represent the foreground color of a position on the PC screen.

The following table lists the bits that correspond to the 16 standard PC colors (BIOS color set) and the HEX values (0 through F) assigned to them by AFR_LOAD_COLOR.

<table>
<thead>
<tr>
<th>Hex Value</th>
<th>Bits</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>black</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>blue</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>green</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>cyan</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>red</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>magenta</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>brown</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>white</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>gray</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>light blue</td>
</tr>
<tr>
<td>A</td>
<td>1010</td>
<td>light green</td>
</tr>
<tr>
<td>B</td>
<td>1011</td>
<td>light cyan</td>
</tr>
<tr>
<td>C</td>
<td>1100</td>
<td>light red</td>
</tr>
<tr>
<td>D</td>
<td>1101</td>
<td>light magenta</td>
</tr>
<tr>
<td>E</td>
<td>1110</td>
<td>yellow</td>
</tr>
<tr>
<td>F</td>
<td>1111</td>
<td>bright yellow</td>
</tr>
</tbody>
</table>

Example

RET = AFR_LOAD_COLOR(1460,1)
AFR_LOAD_COLOR

This example returns 1 byte from the middle of row 19. If the text at this position is green on black, AFR_LOAD_COLOR returns Hex: 02.

See also

“AFR_LOAD” on page 50
“AFR_SELECT” on page 87

AFR_LOCK

Use this function to obtain exclusive access to a specified emulation session or to a serial port configured for script use.

Format

AFR_LOCK(resource, time)

resource

The name of a resource controlled by SA IOM. This can be an emulation session or a serial port configured for script use. To specify an emulation session, use the session name. The name is case sensitive and may include imbedded blanks. Enclose this parameter in quotes. To specify serial ports configured for script use, use AFR_LOCK("REXXPOOL nn",0), where nn is a standard COM port number or a serial communication port expansion adapter port number.

time

This parameter specifies the time that the REXX program will wait for the session to become available if the session is currently locked by another REXX program. Valid values follow:

>1  Specifies the wait time in milliseconds.
0   Specifies that your call does not wait for the resource to become available if the session currently is locked by another program.
-1  Specifies that your call will wait indefinitely for the session to be freed.

Returns

0   The session lock is successful.
1   Unable to obtain lock (the session is already held by another program).
4   The session name is not defined.

Remarks

Use this serialization mechanism to prevent multiple REXX programs from writing simultaneously to the same terminal emulation session.

Call AFR_LOCK immediately before you enter a section of code that needs exclusive use of a terminal emulation session. Then, remember to free the session when your program no longer needs it exclusively. Failure to free a session can prevent other programs that need access to that session from executing successfully.

A program that calls AFR_LOCK is guaranteed exclusive access to a session only if all other programs that use the session call AFR_LOCK.
Example

In the following example, AFR_LOCK is used by a program to serialize access to a session (named “MVS_Console”) before sending a reply to a WTOR.

```rexx
/* Sample REXX code */

Session = 'MVS_Console'
WTORNum = ... /* WTOR Number */

ret = AFR_SELECT(Session)
if (ret <> 0) then do
   say "Cannot select " || Session
   exit
end

do forever
   lock = AFR_LOCK(Session, 0)
   if lock = 0 then do
      send = AFR_SEND_3270("R" || WTORNum || ",1 @E")
      free = AFR_UNLOCK(Session)
      leave
   end
   nul = AFR_SUSPEND(2000)
end
```

The session is locked only when the program needs to write a message to the console. Once the reply is sent, AFR_UNLOCK is issued, which frees the session to other REXX programs.

If the session is unavailable, the program suspends itself for 2 seconds and then checks again for availability.

The following example specifies that the call will wait indefinitely for session TSO to become available.

AFR_LOCK(TSO",-1)

See also

"AFR_UNLOCK" on page 116

AFR_LOG

Use this function to send a string to the server log.

Format

AFR_LOG(string)

`string` This parameter specifies the characters to be written. The maximum length is 240 characters.

Returns

0 The string is successfully logged.
1 An unspecified error was encountered.
AFR_LOG

Remarks
This function places the specified string in the server log. No associated pop-up text message is displayed.

Example
RET = AFR_LOG('SYS A IPL INITIATED')

See also
*AFR_BROADCAST* on page 20
*AFR_ERROR* on page 41
*AFR_OP_ERROR* on page 66

AFR_LOGSIZE

Use this function to retrieve the number of bytes in the server log.

Format
AFR_LOGSIZE()

This function accepts no parameters.

Returns
This function returns the number of bytes in the server log.

Example
Num = AFR_LOGSIZE()

AFR_MC_LASTMSGNUM

Use this function to retrieve the number of messages sent to the Message Collector.

Format
AFR_MC_LASTMSGNUM()

This function accepts no parameters.

Returns
The function returns the number of messages sent to the Message Collector.

Remarks
The Message Collector must be configured.

Example
RET = AFR_MC_LASTMSGNUM()

See also
*AFR_MC_LASTMSGTEXT* on page 55
*AFR_MC_LASTMSGTIME* on page 55
AFR_MC_LASTMSGTEXT

Use this function to retrieve the last message received by the Message Collector.

Format

AFR_MC_LASTMSGTEXT()

This function accepts no parameters.

Returns

The function returns the text sent to the Message Collector.

This function returns a null string if no message has been received since the server’s startup.

Remarks

The Message Collector must be configured.

Example

RET = AFR_MC_LASTMSGTEXT()

See also

“AFR_MC_LASTMSGNUM” on page 54
“AFR_MC_LASTMSGTIME”
“AFR_GET_LINE” on page 42

AFR_MC_LASTMSGTIME

Use this function to get the time of the last message received by the Message Collector.

Format

AFR_MC_LASTMSGTIME()

This function accepts no parameters.

Returns

The function returns the time of the last message sent to the Message Collector.

The format of the return string is: YYYY MM-DD HH:MM

If no message has been received then "0000" is returned.

Remarks

The Message Collector must be configured.

Example

RET = AFR_MC_LASTMSGTIME()
AFR_MC_LOGSIZE

Use this function to retrieve the number of bytes in the Message Collector log.

Format

AFR_MC_LOGSIZE()

This function accepts no parameters.

Returns

This function returns the number of bytes in the Message Collector log.

Example

Num = AFR_MC_LOGSIZE()

See also

“AFR_MC_LASTMSGNUM” on page 54
“AFR_MC_LASTMSGTEXT” on page 55

AFR_MC_REOPEN_LOG

Use this function to reallocate the current Message Collector log under a specified name extension and start a fresh log.

Format

AFR_MC_REOPEN_LOG(extension)

See also

“AFR_MC_LASTMSGNUM” on page 54
“AFR_MC_LASTMSGTEXT” on page 55

Returns

0  The file is reallocated successfully.
1  The specified extension is invalid.

Remarks

When you call AFR_MC_REOPEN_LOG:
1. The current MsgClect.extension (if present) is deleted.
2. MsgClect.LOG is closed.
3. MsgClect.LOG is named to MsgClect.extension.
4. A new, empty MsgClect.LOG is opened for recording messages.
AFR_MC_REOPEN_LOG

Example

RET = AFR_MC_REOPEN_LOG(BU1)

See also

“AFR_MC_LOGSIZE” on page 56
“AFR_REOPEN_LOG” on page 81

AFR_MC_SEND

Use this function to send a message to the Message Collector.

Format

AFR_MC_SEND([attribute|color][,text][,option])

attribute or
color This optional parameter identifies either a color or an escape sequence.

Color keywords are: BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE

If the parameter is specified and it is not one of the above keywords, then an escape character is prefixed to the parameter. For example, to send Esc[H use AFR_MC_SEND("\[H")

text This optional parameter identifies text to send to the Message Collector window.

option This optional parameter, when specified, further instructs the Message Collector about the previously specified text.

Keyword: NOCRLF

When keyword NOCRLF is specified, the Message Collector will not append a carriage return line feed character to the text.

Returns

0 The message is successfully sent.
1 An error occurred while initiating.
3 An error occurred while resolving the host name.
5 An error occurred while connecting to the host.
9 An invalid option is specified.

Remarks

The Message Collector must be configured.

This function internally calls the AFR_SEND_MESSAGE function.

To call a Message Collector on a different server use the AFR_SEND_MESSAGE function.

Example

RET = AFR_MC_SEND("RED","Alert from session LA_01")
AFR_MODEMANSWER

Use this function to put a modem in a ready state to answer a call.

Format

AFR_MODEMANSWER(handle,delay)

handle This parameter identifies the serial communications port. It is a unique handle returned by the AFR_COMOPEN function.
delay This parameter specifies the time to wait, in milliseconds. The maximum value for this parameter is $2^{31} - 1$ milliseconds.

Returns

This function returns a string indicating one of the modem status events in Table 4 on page 59. If the time expires and nothing has happened, the function returns a 1 (delay expired), and the answer-ready state remains active.

Remarks

This function requires a serial communication port configured for script use.

This function remains active until one of the modem status events listed in Table 4 on page 59 is returned or until the AFR_MODEMDISCONNECT function is issued. The only exception is if DelayExpired is returned, in which case the answer-ready state remains active.

The delay parameter plays an important role in this function. See "AFR_MODEMCALL" for a complete description of the delay parameter.

Example

RET = AFR_MODEMANSWER(handle,0)

See also

"AFR_COMOPEN" on page 33
"AFR_MODEMCALL"

AFR_MODEMCALL

Use this function to command the modem to dial the specified number.

Format

AFR_MODEMCALL(handle,dialstring,delay)

handle This parameter identifies the serial communications port. It is a unique handle returned by the AFR_COMOPEN function.
dialstring This parameter specifies the number to dial. It must evaluate to a literal string 30 characters long or less. Table 3 on page 59 lists the characters you can use in an AFR_MODEMCALL dialing string.
AFR_MODEMCALL

**delay**  This parameter specifies, in milliseconds, the time to wait for the call action to complete. The maximum value for this parameter is $2^{31}-1$ milliseconds. If -1 is specified, the function will wait as long as is necessary for the modem to reply.

**Returns**

This function returns a string indicating one of the modem status events in Table 4. If the time expires and nothing has happened, the function returns a 1 (delay expired), and the modem continues to process the call. You can use the AFR_MODEMSTATUS function to examine the status later.

**Remarks**

This function requires a serial communication port configured for script use.

This function remains active until one of the modem status events listed in Table 4 is returned or until the AFR_MODEMDISCONNECT function is issued. The only exception is if *DelayExpired* is returned, in which case the answer-ready state remains active.

**Example**

```plaintext
RET = AFR_MODEMCALL(handle, '9,1(310)555-1212', 10000)
```

**See also**

"AFR_COMOPEN" on page 33
"AFR_MODEMSTATUS" on page 60

**Table 3. AFR_MODEMCALL Dialstring characters**

<table>
<thead>
<tr>
<th>Char</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-9</td>
<td>Numbers to be dialed</td>
</tr>
<tr>
<td>( ) /</td>
<td>No effect; used only to make dialstring more readable</td>
</tr>
<tr>
<td>,</td>
<td>2 second delay</td>
</tr>
</tbody>
</table>

**Table 4. Modem status events**

<table>
<thead>
<tr>
<th>Return String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No handle; the specified handle does not exist</td>
</tr>
<tr>
<td>1</td>
<td>Delay expired; the specified delay time has expired, but the modem call is still active</td>
</tr>
<tr>
<td>4</td>
<td>Error; an undefined error has occurred</td>
</tr>
<tr>
<td>6</td>
<td>Connected; the modem has made connection</td>
</tr>
<tr>
<td>7</td>
<td>Path fail; the function is unable to communicate with the modem</td>
</tr>
<tr>
<td>8</td>
<td>Disconnected; the modem has been disconnected</td>
</tr>
<tr>
<td>9</td>
<td>In progress; the modem is attempting to make connection</td>
</tr>
<tr>
<td>10</td>
<td>COM port not opened</td>
</tr>
<tr>
<td>13</td>
<td>No carrier; the modem cannot detect a carrier signal</td>
</tr>
<tr>
<td>14</td>
<td>No dial; the modem cannot detect a dial tone</td>
</tr>
<tr>
<td>15</td>
<td>Busy; the number called is busy</td>
</tr>
<tr>
<td>16</td>
<td>No answer; the number called did not answer within the modem's time out period</td>
</tr>
</tbody>
</table>
AFR_MODEMDISCONNECT

Use this function to disconnect a modem.

Format

AFR_MODEMDISCONNECT(handle)

handle  This parameter identifies the serial communications port. It is a unique handle
         returned by the AFR_COMOPEN function. The modem must have been
         previously opened with the AFR_MODEMCALL function.

Returns

This function always returns the modem status event “Disconnected” as listed in
Table 4 on page 59.

Remarks

This function requires a serial communication port configured for script use.

Example

RET = AFR_MODEMDISCONNECT(handle)

See also

"AFR_COMOPEN" on page 33
"AFR_MODEMANSWER" on page 58

AFR_MODEMSTATUS

Use this function to return the current status of a modem.

Format

AFR_MODEMSTATUS(handle)

handle  This parameter identifies the serial communications port. It is a unique handle
         returned by the AFR_COMOPEN function.

Returns

Table 4 on page 59 lists the modem status events that may be returned by this
function.

Remarks

This function requires a serial communication port configured for script use.

The modem must have been previously opened with the AFR_MODEMANSWER or
AFR_MODEMCALL function.

Example

RET = AFR_MODEMSTATUS(handle)
AFR_NOTIFY

Use this function to start an alert escalation process based on the data configured in the alert escalation database.

Format

AFR_NOTIFY(escalation_id,priority,message)

- `escalation_id`: This parameter identifies a notification policy that is defined in the alert escalation database. Be aware that the escalation ID is case-sensitive.
- `priority`: This parameter is a user-assigned decimal number, in the range from 1 to 999, that is intended to express the relative importance of the alert.
- `message`: This parameter is the message text to be passed to the recipient of the alert.

Note: The maximum length of the message that can be forwarded using the various notification methods depends upon the hardware and service providers being used.

Returns

Upon successful completion, this function returns the Alert ID. This is a unique positive number that is assigned by SA IOM and is used to track the alert message.

If there is an error, this function returns one of the negative notify return codes listed in Table 7 on page 133.

Remarks

The SA IOM alert escalation feature must be configured.

Always call the AFR_NOTIFY_INIT function before calling the AFR_NOTIFY function. Be aware that the escalation ID is case-sensitive.

Executing the AFR_NOTIFY function with the correct number of parameters creates an alert message that is delivered as specified in the alert escalation notification policy. The Alert ID that is returned by the AFR_NOTIFY function uniquely identifies the alert message, is used to track the alert message, and is used as input in subsequent notify functions.

Example

Example 1:

```plaintext
alertid = AFR_NOTIFY("MyEscalationID",2,"test message")
```

Example 2:

```plaintext
/* Logon the script on the SA IOM Alert Escalation database system */
rc = Afr_Notify_Init()
if (rc \jour 0) then do
   say "Afr_Notify_Init error "rc
   say "Last Error: "Afr_Notify_LastError()
exit
```
AFR_NOTIFY

end

/* Launch the "SampleEscl" escalation as defined in SA IOM */
alertid=afr_notify("SampleEscl",222,'This is a test alert')
if (alertid < 1) then do
   say "Afr_Notify error "alertid
   say "Last Error: "Afr_Notify_LastError()
afr_notify_exit()
end

/* Log the script off */
rc=afr_notify_exit()
if (rc \= 0) then do
   say "Afr_Notify_Exit error "rc
   say "Last Error: "Afr_Notify_LastError()
   exit
end

AFR_NOTIFY_ACK

Use this function to acknowledge the receipt of an alert message and to indicate
your status. This function can be used to stop the alert escalation process.

Format

AFR_NOTIFY_ACK(alert_id,ack_level,user_id,description)

alert_id  The alert ID as assigned by SA IOM.
ack_level Specify one of the following keywords to indicate your status.

   RECEIVE - Indicates that you received the alert message. The alert
   escalation process continues (counting down the time until escalation to the
   next escalation level).

   REJECT - Indicates that you received the alert message and will not
   attempt to fix the problem. The alert escalation process continues.

   ACCEPT - Indicates that you are taking ownership of the problem that
   prompted this alert message. The alert escalation process stops upon
   receipt of this keyword.

user_id   This parameter is a user ID that is used to identify a person who is
          interacting with the alert escalation feature. It does not need to be a user ID
          that is defined in the alert escalation database. For example, it can be the
          initials or nickname of the recipient of the alert message.

description This parameter is for descriptive text that indicates the status of the
          situation.

Returns

Upon successful completion, this function returns 0.

Not passing four parameters will cause the standard REXX error 40.

If there is an error, this function returns one of the negative notify return codes
listed in Table 7 on page 133.

Remarks

The SA IOM alert escalation feature must be configured.
Always call the AFR_NOTIFY_INIT function before calling the AFR_NOTIFY_ACK function.

The ack_level keyword that you supply to the AFR_NOTIFY_ACK function determines whether or not alert escalation processing continues on to the next escalation level as defined in the notification policy in the alert escalation database.

**Note:** If all recipients at a one notification level of a policy REJECT the alert message, then the notification level escalates immediately to the next level as defined in the policy.

**Example**

**Example 1:**

```plaintext
id = AFR_NOTIFY_ACK("MyAlertID","RECEIVE","Jim","Testing MyAlertID")
```

**Example 2:**

```plaintext
Arg alertid,acklevel,userid,desc /* pickup input parameters */
lrc = 0 /* assume good completion */

/*-----------------------------------------------------------------*/
/* Tell SA IOM who is acknowledging the alert... */
/*-----------------------------------------------------------------*/
rc = Afr_Notify_Init()
if rc \= 0 then /* bad return code from service */
do /* ..Yes, report error */
   say "Afr_Notify_Init error" rc
   say "Last Error:" Afr_Notify_LastError()
lrc = 8 /* remember bad completion */
end /* of reporting error */

/*-----------------------------------------------------------------*/
/* Acknowledge the alert... */
/*-----------------------------------------------------------------*/
if lrc = 0 then /* everything OK ? */
do /* ..Yes, continue */
   rc = afr_Notify_ACK(alertid,acklevel,userid,desc)
   if rc \= 0 then /* good completion ? */
do /* ..No, report error */
      say "Afr_Notify_ACK error" rc
      say "Last Error:" Afr_Notify_LastError()
lrc = 8 /* remember bad completion */
end /* of reporting error */
end

/*-----------------------------------------------------------------*/
/* Allow SA IOM to cleanup... */
/*-----------------------------------------------------------------*/
if lrc = 0 then /* everything OK ? */
do /* ..Yes, continue */
   rc = afr_Notify_Exit()
   if rc \= 0 then /* bad return code from service */
do /* ..Yes, report error */
      say "Afr_Notify_Exit error" rc
      say "Last Error:" Afr_Notify_LastError()
lrc = 8 /* remember bad completion */
end /* of reporting error */
end

exit lrc /* return to caller w/ RC */
```
AFR_NOTIFY_EXIT

AFR_NOTIFY_EXIT

Use this function to log off from the alert escalation database.

Format

AFR_NOTIFY_EXIT()

Returns

Upon successful completion, this function returns 0.

Example

/* Log the script off */
rc = afr_notify_exit()
if (rc \= 0) then do
   say "Afr_Notify_Exit error " rc
   say "Last Error: " Afr_Notify_LastError()
   exit
end

AFR_NOTIFY_INIT

AFR_NOTIFY_INIT

Use this function to log on to the alert escalation database.

Format

AFR_NOTIFY_INIT()

Returns

Upon successful completion, this function returns a zero.

Remarks

If there is an error, this function returns one of the negative notify return codes listed in Table 7 on page 133.

Example

rc = AFR_NOTIFY_INIT()
if (rc \= 0) then do
   say "Afr_Notify_Init error " rc
   say "Last Error: "Afr_Notify_LastError()
   exit
end

AFR_NOTIFY_LASTERROR

AFR_NOTIFY_LASTERROR

Use this function to retrieve information about the last recorded error for this alert escalation script.
AFR_NOTIFY_LASTERROR

Format

AFR_NOTIFY_LASTERROR()

Returns

This function returns a character string which is the last recorded error for the alert escalation script. The format is an unformatted text string explaining the error. For example "Invalid Escalation ID specified"

If no error has been recorded for the script, this function returns a "no information available" character string.

Remarks

The SA IOM alert escalation feature must be configured.

Example

If you use the following REXX code fragment in a script that you write, and if you specify a value for ESCID that does not exist, then the value returned by AfrNotify_LastError() is: "Invalid Escalation ID specified"

/* Launch the "SampleEsc1" escalation as defined in SA IOM */
alertid=afr_notify(ESCID,222,'This is a test alert')
if (alertid < 1) then do
   say "Afr_Notify error "alertid
   say "Last Error: "Afr_Notify_LastError()
   afr_notify_exit()
end

AFR_NOTIFY_SET_RESULT

Use this function (from an alert escalation helper script only) to report the completion status of the script.

Format

AFR_NOTIFY_SET_RESULT(alert_id,escalation_id,process_handle,script_name,compcode,description)

alert_id
This parameter is the alert ID as passed on input to the helper script.

escalation_id
This parameter is the escalation ID as passed on input to the helper script.

process_handle
This parameter is a handle that uniquely identifies this helper script invocation; it is passed on input to the helper script.

script_name
This parameter is the name of the current REXX helper script.
**AFR_NOTIFY_SET_RESULT**

**compcode**
This parameter is the completion code which indicates the success of the helper script in delivering the notification message. It can be one of the following.

0= OK - The notification was successfully delivered.

1= WARNING - The notification was delivered but a warning was given by the service provider.

2= DATA ERROR - The notification could not be delivered. (Result: keep the next level's delay.)

3= DATA ERROR - The notification could not be delivered. (Result: bypass the next level's delay.)

4= DEVICE or HARDWARE ERROR (Result: The current notification method is marked "disabled" and is no longer usable. To "enable" the notification method again, the device or hardware error must be corrected and SA IOM must be recycled.)

**description**
This parameter is for descriptive text that indicates the status of the situation. The maximum length of the description parameter is 100 characters. The text will be truncated if longer.

**Returns**
Upon successful completion, this function returns 0.

Upon error, this function returns one of the negative notify return codes listed in Table 7 on page 133.

**Remarks**
The SA IOM alert escalation feature must be configured.

This function is to be used from SA IOM alert escalation helper scripts only.

**Example**

```c
rc = AFR_NOTIFY_SET_RESULT(alert_id,escalation_id,process_handle,gbl.0me,,
   err_rc,err_msg)
If rc <> 0 then /* good completion ? */
   do /* ..No, report error */
      err_msg = 'Bad return code from AFR_NOTIFY_SET_RESULT, RC='rc
      rc = Afr_Log(gbl.0me': err_msg)
   end
```

---

**AFR_OP_ERROR**

Use this function to display a pop-up text message to all users with authority to respond to client messages. The message is also sent to the server log.

**Format**

AFR_OP_ERROR(string,color)

**string**
This parameter specifies the characters to be written. They must be enclosed in quotes. The maximum length is 240 characters.
AFR_OP_ERROR

**color**
This parameter specifies the color of the pop-up message box:

<table>
<thead>
<tr>
<th>Color</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>red</td>
</tr>
<tr>
<td>1</td>
<td>green</td>
</tr>
<tr>
<td>2</td>
<td>yellow</td>
</tr>
</tbody>
</table>

**Returns**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The string is successfully logged.</td>
</tr>
<tr>
<td>1</td>
<td>There is an error.</td>
</tr>
</tbody>
</table>

**Remarks**
This function places the specified `string` in the server log and in a pop-up message box to all connected users with authority to respond to client messages. You can use this function to send a message to all SA IOM operators and to the system administrator.

**Example**

```
RET = AFR_OP_ERROR('testing',0)
```

**See also**

- [AFR_BROADCAST](#) on page 20
- [AFR_ERROR](#) on page 41
- [AFR_LOG](#) on page 53

AFR_PEER_CLOSE

Use this function to allow an SA IOM REXX script to terminate a conversation with another SA IOM server, or AF/OPERATOR address space, or other peer system.

The specified linkid cannot currently be in use with another REXX script.

**Format**

```
AFR_PEER_CLOSE(linkid)
```

**linkid**
A 1 to 8 character, user-specified alphanumeric designation that uniquely identifies the conversation. The linkid may already be coded in a successful AFR_PEER_OPEN or initiated by another peer, whose linkid was obtained using AFR_PEER_QUERY.

You can omit this parameter when AFR_PEER_CLOSE is invoked in a REXX script that was started by a peer system. In this context, the conversation linkid is already known to SA IOM. For inbound non AF peer connections, either omit this parameter or use the link ID passed to the PeerNSvr.rex program.

For outbound non AF peer connections only, specify the special link ID: "NONAF"

**Returns**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The conversation was successfully terminated.</td>
</tr>
</tbody>
</table>
**AFR_PEER_CLOSE**

1. An invalid parameter was specified on the AFR_PEER_CLOSE.
2. An invalid peer environment exists; unable to perform the AFR_PEER_CLOSE operation.
3. The linkid specified is not known to this SA IOM system.
4. The AFR_PEER_CLOSE failed to complete.
5. The specified linkid is currently being used by another REXX script; unable to perform the AFR_PEER_CLOSE operation.
10. A TCP/IP error occurred while closing the connection to the specified host.

**Remarks**

Because a REXX script may not always be able to predict whether other REXX scripts are currently using the same conversation (as represented by the linkid), there could be situations where both RC = 0 and RC = 10 are acceptable return codes. In these situations, the last REXX user of the conversation should call AFR_PEER_CLOSE to ensure that the linkid is closed.

To successfully use the special "NONAF" linkid, the PEER_PORT_NON_AFPACKET keyword must be defined in the server's RpSvrPrf.txt file. Also, currently only one active "NONAF" session is allowed. Be sure to close a session before starting a new one.

**Example**

In the following example, the OGMVS1 conversation is terminated if there are no other users of the conversation. The OGMVS1 linkid can then be reused to start another conversation.

```rexx
/* REXX */
ret = AFR_PEER_CLOSE("OGMVS1")
if ret = 0 then
   say "Connection successfully terminated"
else
   say "AFR_PEER_CLOSE failed with rc:" ret
```

**See also**

"AFR_PEER_OPEN" on page 69
"AFR_PEER_QUERY" on page 70

---

**AFR_PEER_GETLASTERROR**

Use this function to allow an SA IOM REXX script to find the return code from the last peer function (AFR_PEER_ xxx) that executed. This function is useful in determining the reason that AFR_PEER_RCV or AFR_PEER_SENDRCV returned a null string.

**Format**

```rexx
AFR_PEER_GETLASTERROR()
```

**Returns**

This function returns the error code from the last peer function call. If the last peer function call completed successfully, this value can be zero.
AFR_PEER_OPEN

Use this function to allow an SA IOM REXX script to start a conversation with another SA IOM server, AF/OPERATOR address space, or other peer system. This function attempts to make a TCP/IP connection to the hostname and listening port specified in the function's parameter list.

The linkid cannot already be in use on this SA IOM system or on the destination peer.

Format

AFR_PEER_OPEN(linkid, hostname, portnumber[, ip-family])

linkid A 1-8 character, user-specified alphanumeric designation that uniquely identifies the conversation on both peer systems.

For outbound non-AF peer connections only, specify the special link ID: "NONAF"

hostname The IP address of the remote host that you want to establish a session with. The IP address can be either a dotted decimal IP address or a hostname defined to your network's domain name server. If it is a dotted decimal IP address, none of the four numeric levels can exceed 255.

portnumber The listening port of the remote host that you want to establish a session with. The port number can be any integer greater than or equal to 1025 and less than or equal to 32767.

ip-family The type of IP connection. It must be either "IPV6" or "IPV4". This parameter is optional.

Returns

0 The conversation was successfully started.
1 An invalid parameter was specified on AFR_PEER_OPEN.
2 An invalid peer environment exists; unable to start a conversation.
3 A conversation already exists with the specified linkid.
5 The AFR_PEER_OPEN failed to complete.
7 An invalid ID record was sent during AFR_PEER_OPEN processing.
100xx A TCP/IP error occurred while connecting to the specified host.

Remarks

The TCP/IP error code value will usually be a very important piece of diagnostic data in any failed AFR_PEER_OPEN call. The TCP/IP error codes, per the Windows socket specification, are documented in the Winsock.h file and, by convention, are 5-digit numbers in the range of 10000 to 11000. For example, an attempt to connect to a server that is not running will fail with a 10061 error code, which is defined in Winsock.h as WSAECONNREFUSED.
AFR_PEER_OPEN

For a list of common TCP/IP error codes, see the appendix “TCP/IP Error Codes” in the System Automation for Integrated Operations Management User’s Guide.

To successfully use the special "NONAF" linkid, the PEER_PORT_NON_AFPACKET keyword must be defined in the server's RpSvrPrf.txt file.

Example

In the following example, an attempt is made to start a conversation with an AF/OPERATOR address space whose IP address is 129.0.44.55 and which normally listens for client connections on port 1234.

If the AFR_PEER_OPEN is successful, a linkid of AFOP1 is assigned to the conversation. Once the conversation is established, other REXX scripts running on the AF/OPERATOR peer can use linkid AFOP1 when communicating with the local SA IOM peer. Similarly, other REXX scripts running on the local SA IOM can use AFOP1 when communicating with the AF/OPERATOR peer.

Several TCP/IP error codes, common to failed connections, are tested for and reported if encountered.

```rexx
/* REXX */
ret = AFR_PEER_OPEN("AFOP1","129.0.44.55",1234)
if ret = 10049 then /*Is IP address not available from machine? */
    say "AF/OPERATOR cannot be reached from the local machine"
if ret = 10060 then /*Connection timed out? */
    say "Connection attempt to AF/OPERATOR timed out"
if ret = 10061 then /*Is the AF/OPERATOR TCP/IP server down? */
    say "AF/OPERATOR not accepting connections"
/* Also check for a common SA IOM error condition */
if ret = 3 then /*Is this linkid already in use? */
    say "Linkid AFOP1 is already active"
if ret = 0 then
do
    say "Connection to AF/OPERATOR successfully started!"
    /*Perform data transmission to/from AF/OPERATOR using "AFOP1"*/
end
exit
/* Linkid AFOP1 is left in active state for other REXX scripts to use*/
```

See also

“AFR_PEER_CLOSE” on page 67

AFR_PEER_QUERY

Use this function to allow an SA IOM REXX script to find out various types of peer-related status information, such as whether a particular linkid is active.

Format

AFR_PEER_QUERY(keyword, searchitem)

Keyword | The type of peer data to return. Valid keywords are:
--- | ---
HOSTNAME |
LINKID |
LINKIDEXISTS |
**AFR_PEER_QUERY**

**searchitem** An optional parameter to qualify the search criteria. For example, when you specify a hostname value with LINKID, the linkids that are active with that hostname are returned. If you specify a linkid with HOSTNAME, the IP address of the peer communicating over that linkid will be returned. You must specify a linkid searchitem with the LINKIDEXISTS keyword.

If you specify LINKID but omit searchitem, the current linkid (if there is one) will be returned; otherwise, a null string is returned.

**Note:** A REXX script only has a current linkid if it was started by a peer system. In this case, the linkid assigned by the peer when it initiated the conversation that kicked off the REXX script is considered to be the current or implied linkid.

**Returns**

Upon success, this function returns a string (or concatenation of strings) of the type requested in the keyword parameter. When returning a hostname, it is in the form of a dotted decimal IP address. When multiple linkids match the search criteria, the individual entries are concatenated with a single blank delimiter between each entry.

Upon error, this function returns a null string. If a null string is returned, you can use AFR_PEER_GETLASTERROR to find the exact reason for the AFR_PEER_QUERY error. Following is a list of possible return codes.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Data was successfully retrieved.</td>
</tr>
<tr>
<td>1</td>
<td>An invalid parameter was specified on the AFR_PEER_QUERY.</td>
</tr>
<tr>
<td>2</td>
<td>An invalid peer environment exists; unable to query peer-related status information.</td>
</tr>
<tr>
<td>5</td>
<td>The AFR_PEER_QUERY failed to complete.</td>
</tr>
</tbody>
</table>

**Remarks**

AFR_PEER_QUERY offers several useful applications to SA IOM scripts:

- A REXX script can find out if a linkid to a particular IP address is active, so that the linkid can be reused to send and/or receive data without having to initiate a new conversation.
- A REXX script can determine if a particular linkid is already active before calling AFR_PEER_OPEN and receiving a “Conversation already exists with the specified linkid”, RC = 3 error code.
- A REXX script that was started by a peer system can find its own current linkid and then use this value as input to a second AFR_PEER_QUERY call, which will return the IP address of the system that started this REXX script running.
- A REXX script that was started by a peer system can find its own current linkid, start another REXX script, and pass it the linkid obtained from the AFR_PEER_QUERY call.

**Example**

**Example 1:**

In the following example, a REXX script determines if there are any active conversations with IP address 129.0.11.22, which corresponds to an OG/MVS peer system. If a non-null linkid is returned, the linkid gets reused to send a REXX exec to run on the OG/MVS peer.
Example 2:

In the following example, a conversation is started by `Script2.rex` on SA IOM Server2, which causes `Script1.rex` to execute on SA IOM Server1.

```rexx
/* REXX */
/* Script2.rex on SA IOM Server2 */
rplink = AFR_PEER_QUERY(LINKIDEXISTS, "RP1")
/* Script2.rex on SA IOM Server2 */
rplink = AFR_PEER_QUERY(LINKIDEXISTS, "RP1")
if rplink = "" then /* RP1 not in use? */
do
  if rplink = "" then /* RP1 not in use? */
do
    ret = AFR_PEER_OPEN("RP1", "133.15.25.35", 1999)
    if ret <> 0 then /* perform application error processing and exit */
      ret = AFR_PEER_SEND("RP1", "EXEC", "Script1.rex")
    /* Wait up to 10 minutes for data to be received from the peer */
    rcvbuf = AFR_PEER_RCV("RP1", 600)
  end
end
```

The REXX script called `Script1.rex` starts executing on SA IOM Server1. Its current linkid is RP1, since that was the conversation name assigned by the `Script2.rex` on Server2. It uses the current linkid to send test data back to `Script2.rex`, which is blocking on an AFR_PEER_RCV.

```rexx
/* REXX */
/* Script1.rex on SA IOM Server1 */
currlink = AFR_PEER_QUERY(LINKID)
  /* No parm supplied means give the current linkid */
if currlink <> " " then
do
  sendbuf = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
  ret = AFR_PEER_SEND(currlink, "USRDATA", sendbuf)
  if ret <> 0 then /* perform application error processing and exit */
end
```

Finally, another REXX script on SA IOM Server1, `Script1a.rex`, joins the conversation.

```rexx
/* REXX */
/* Script1a.rex on SA IOM Server1 */
rplink = AFR_PEER_QUERY(LINKIDEXISTS, "RP1")
if rplink = "RP1" then /* Linkid "RP1" currently active? */
do
  sendbuf = "Scripta.rex parm1 parm2" /* Start a REXX exec on Server2 */
  ret = AFR_PEER_SEND(rplink, "EXEC", sendbuf)
end
```

See also

- "AFR_PEER_CLOSE" on page 67
- "AFR_PEER_CLOSE" on page 69
- "AFR_PEERSEND" on page 75
- "AFR_PEER_RCV" on page 73
AFR_PEER_RCV

Use this function to allow an SA IOM REXX script to receive data from a peer system. This function does not return any reply data to the peer.

Format

AFR_PEER_RCV(linkid, timeout)

linkid  A 1-8 character, user-specified alphanumeric designation that uniquely identifies a conversation. The linkid may already be coded in a successful AFR_PEER_OPEN or initiated by another peer, whose linkid was obtained using AFR_PEER_QUERY.

You can omit this parameter when AFR_PEER_RCV is invoked in a REXX script that a peer system started. In this context, the conversion linkid is already known to SA IOM. For inbound non AF peer connections, either omit this parameter or use the link ID passed to the PeerNSvr.rex program.

timeout  The number of seconds to wait for data to be received before the AFR_PEER_RCV will time out. If you omit this parameter, an infinite wait time is assumed.

Returns

The AFR_PEER_RCV function returns a string, where received data from the peer system has been placed. If no data is received before the timeout value elapses or an error occurs during the receive operation, a null string is returned. A maximum of 32K of data can be received.

If a null string is returned, you can use AFR_PEER_GETLASTERROR to find the exact reason for the AFR_PEER_RCV error. Following is a list of possible return codes:

0  Data was successfully received.
1  An invalid parameter was specified on the AFR_PEER_RCV.
2  An invalid peer environment exists; unable to receive data.
4  The linkid specified is not known to this SA IOM system.
5  The AFR_PEER_RCV failed to complete.
6  The timeout interval expired before any data was received.
8  The received data is invalid. For example, it is EBCDIC instead of ASCII, it does not adhere to SA IOM's TCP/IP data packet format, and so on.
100xx  A TCP/IP error occurred while receiving data from the peer system.

Remarks

When using AFR_PEER_RCV:

- If one peer issues an AFR_PEER_RCV, the other peer must issue an AFR_PEER_SEND or AFR_PEER_SENDRCV (or COMSDRCV on an OMEGACENTER Gateway for z/OS system). If two peers have issued an AFR_PEER_RCV and both are waiting for data in the same conversation, a deadlock situation occurs. In this scenario, only a non-infinite timeout value in one or both of the AFR_PEER_RCV calls, or a user-issued halt of one or both of the REXX scripts, will break the deadlock.
AFR_PEER_RCV

- If multiple AFR_PEER_RCV calls are issued by multiple REXX scripts using the same linkid, the receive operations are fulfilled on a FIFO (first in, first out) basis.
- To successfully use the special "NONAF" linkid, the PEER_PORT_NON_AFPACKET keyword must be defined in the server's RpSvrPrf.txt file.

Example

Example 1:

In the following example, a conversation has already been established with a REXX script on another SA IOM server. A linkid of RMTPROD has been assigned to this conversation. An AFR_PEER_RCV is issued, with an infinite wait time, to pull any incoming data from the other server. The other SA IOM server must not also be waiting on an AFR_PEER_RCV, or the conversation will deadlock.

```rexx
/* REXX */
rcvbuf = AFR_PEER_RCV("RMTPROD")
/* Wait indefinitely for data to be received */
if rcvbuf <> " " then
   say "Received the following data from RMTPROD:" rcvbuf
else
   say "AFR_PEER_RCV failed with rc:" AFR_PEER_GETLASTERROR()
```

Example 2:

In the following example, an OMEGACENTER Gateway for z/OS REXX exec opens a linkid named RMT1 with an SA IOM peer and sends it a buffer of data. This peer-to-peer application has been designed so that there is a special-purpose REXX script on the SA IOM side, called RECEIVE.REX. It is the job of RECEIVE.REX to test for the availability of the RMT1 linkid and, when it becomes available, to keep making AFR_PEER_RCV calls with 60-second wait times, so that it can receive all the data transmissions from its OMEGACENTER Gateway for z/OS peer.

The OMEGACENTER Gateway for z/OS REXX exec specifies a reply length of 100 bytes, which means it will wait for an acknowledgement message from the SA IOM REXX script. By convention, the response data in OMEGACENTER Gateway for z/OS REXX execs is stored in a single compound variable called COMDATA.1.

```rexx
/* REXX */
/* Production OMEGACENTER Gateway for z/OS on System A */
link define linkid(RMT1) type(TCPIP) hostname(129.0.10.20)
   portnumber(1234)
"link start linkid(RMT1)"
DATA = "OMEGACENTER Gateway for z/OS Test Message"
RETURN_CODE = COMSDRCV("RMT1", DATA, "USRDATA", 100)
if RETURN_CODE = 0 then
   say COMDATA.1
else
   say "Error encountered in COMSDRCV, rc =" RETURN_CODE
```

The relevant code from the SA IOM REXX script, RECEIVE.REX, is listed below. In this context, the DATARPLOY acknowledgement message requires a linkid parameter to be supplied. There is no implied current linkid, since RECEIVE.REX was not started by the OMEGACENTER Gateway for z/OS peer system.

```rexx
/* REXX */
do forever
   /*See if production OMEGACENTER Gateway for z/OS system with IP hostname "SYSA"
    * has connected */
linkid = AFR_PEER_QUERY(LINKID, "SYSA")
```
if linkid = "RMT1" then /* Is it the right linkid? */ do
  rcvbuf = AFR_PEER_RCV("RMT1", 60)
  /* Wait up to 60 seconds for data */
  if rcvbuf <> " " then
    /* process the received data */
    tstamp = time()
    sendbuf = "Received message from OMEGACENTER Gateway for z/OS at" ||tstamp
    ret = AFR_PEER_SEND("RMT1", "DATARPLY", sendbuf)
    end
  else
    say "AFR_PEER_RCV failed with rc:" AFR_PEER_GETLASTERROR()
  end
end

See also
"AFR_PEER_QUERY" on page 70
"AFR_PEER_OPEN" on page 69
"AFR_PEER_SEND"
"AFR_PEER_SENDRCV" on page 77

**AFR_PEER_SEND**

Use this function to allow an SA IOM REXX script to send data to a peer system. You can use this function in communication scenarios where a response, directly tied to the data being sent, is not required from the peer system.

**Format**

AFR_PEER_SEND(linkid, datatype, sendbuf)

- **linkid**
  A 1-8 character, user-specified alphanumeric designation that uniquely identifies a conversation. The linkid may already be coded in a successful AFR_PEER_OPEN or initiated by another peer, whose linkid was obtained using AFR_PEER_QUERY.

  You can only omit this parameter when AFR_PEER_SEND is invoked in a REXX script that a peer system started. In this context, the conversation linkid is already known to SA IOM. For inbound non AF peer connections, either omit this parameter or use the link ID passed to the PeerNSvr.rex program.

  For outbound non AF peer connections only, specify the special link ID: "NONAF"

- **datatype**
  The type of data being sent. Valid values are:
  - EXEC - the send buffer consists of an OMEGACENTER Gateway for z/OS command or the name of a REXX exec followed by one or more input arguments.
  - DATARPLY - the send buffer contains response data that was requested by the connected peer.
  - USRDATA - the send buffer contains input data or some other miscellaneous conversational data. Always use this value when using the special link ID: "NONAF"

- **sendbuf**
  The data to be sent. The size of the send buffer cannot exceed 32K. The data can be a text string contained in a REXX variable or a text string enclosed within quotes.
AFR_PEER_SEND

Returns

0 The data was successfully sent.
1 An invalid parameter was specified on the AFR_PEER_SEND.
2 An invalid peer environment exists; unable to send the data.
4 The linkid specified is not known to this SA IOM system.
5 The AFR_PEER_SEND failed to complete.
9 DATARPLY encountered when another data type was expected.

Remarks

AFR_PEER_SEND is best suited for “fire-and-forget” communication situations where, for example, a REXX exec must get triggered to run on a peer system to perform some routine task and no response data is necessary.

Return code 9 pertains to what might be called “conversation context”, such as the relationship between two peers and their changing roles as sender and receiver. When using AFR_PEER_SEND, you need to understand the current state of your peer conversations. In particular, conversations should not send a DATARPLY record when no reply data was requested, and conversely, conversations should not send something other than a DATARPLY record when that is the only type of record that the other peer system is expecting.

To successfully use the special "NONAF" linkid, the PEER_PORT_NON_AFPACKET keyword must be defined in the server's RpSvrPrf.txt file.

Example

Example 1:

In the following example, a conversation is established between two SA IOM servers. A linkid of RMT1 is assigned to the conversation. If the AFR_PEER_OPEN completes successfully, a buffer is sent that will cause a REXX script named beep.rex, with one phone number argument, to start on the other SA IOM server.

```rexx
/* REXX */
ret=AFR_PEER_OPEN("RMT1", "198.200.15.25", 1918)
if ret <> 0 then
   do
      /* perform application error processing and exit */
   end
else
   do
      sendbuf = "beep.rex 9,1(310)555-1212"
      ret = AFR_PEER_SEND("RMT1", "EXEC", sendbuf)
   end
```

Example 2:

In the following example, a simple “fire-and-forget” scenario illustrates an OMEGACENTER Gateway for z/OS REXX exec opening a connection to an SA IOM peer and causing a REXX script to execute. The OMEGACENTER Gateway for z/OS peer does not request any return data.

```rexx
/* REXX */
"link define linkid(AFRP) type(TCPIP) hostname(AFRPROD) portnumber(1234)"
```
Example 3:

In the following example, an OMEGACENTER Gateway for z/OS to SA IOM peer connection utilizes a DATARPLY record type. A REXX exec running on OMEGACENTER Gateway for z/OS has already done a LINK DEFINE and LINK START to an SA IOM peer, assigning IOM1 as the linkid. The exec then issues the COMSDRCV API to start TEST1.REX running on SA IOM. Because a reply length of 10 has been specified, the SA IOM peer knows it must send up to 10 bytes of reply data back. The OMEGACENTER Gateway for z/OS exec is in a wait, until the data is returned by the SA IOM peer.

```rexx
/* REXX */
/* OMEGACENTER Gateway for z/OS */
DATA = "EX TEST1"
RETURN_CODE = COMSDRCV("IOM1", DATA, "EXEC", 10)
```

The TEST1.REX script which starts executing on SA IOM uses AFR_PEER_SEND to return 10 bytes of data to the OMEGACENTER Gateway for z/OS peer. (Because this is a peer-initiated REXX script, and the linkid of the peer being responded to is already known by SA IOM, the linkid parameter can be omitted.) In this context, DATARPLY is the only valid data type, because a reply length was specified in the COMSDRCV call. If reply data had not been requested by the other peer, the AFR_PEER_SEND function call would fail with an RC = 9 error.

```rexx
/* REXX */
/* SA IOM */
sendbuf = "0123456789"
ret = AFR_PEER_SEND("DATARPLY", sendbuf)
```

See also

- "AFR_PEER_QUERY" on page 70
- "AFR_PEER_OPEN" on page 69

AFR_PEER_SENDRCV

Use this function to allow an SA IOM REXX script to send data to a peer system and then wait for response data to be received back from that same peer system.

**Format**

```rexx
AFR_PEER_SENDRCV(linkid, datatype, replylength, sendbuf, timeout)
```

**linkid**

A 1-8 character, user-specified alphanumeric designation that uniquely identifies a conversation. The linkid may already be coded in a successful AFR_PEER_OPEN or initiated by another peer, whose linkid was obtained using AFR_PEER_QUERY.

Omit this parameter when AFR_PEER_SENDRCV is invoked in a REXX script started by a peer system. In this context, the conversation linkid is already known by SA IOM. For inbound non AF peer connections, either omit this parameter or use the link ID passed to the PeerNSvr.rex program.

For outbound non AF peer connections only, specify the special link ID: "NONAF"
**AFR_PEER_SENDRCV**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **datatype** | The type of data being sent. Valid values are:  
EXEC - the send buffer consists of an OMEGACENTER Gateway for z/OS command or the name of a REXX exec followed by one or more input arguments.  
DATARPLY - the send buffer contains response data that was requested by the connected peer.  
USRDATA - the send buffer contains input data or some other miscellaneous conversational data. Always use this value when using the special link ID: "NONAF" |
| **replylength** | The maximum length, in bytes, of response data that will be accepted. This value can be any integer from 1 to 32K.  
**Note:** Zero is not a valid value. If no reply data is needed, use AFR_PEER_SEND instead. |
| **sendbuf** | The data to be sent. The size of the send buffer cannot exceed 32K. The data can be a text string contained in a REXX variable or a text string enclosed within quotes. |
| **timeout** | The number of seconds to wait for data to be received before the AFR_PEER_SENDRCV will time out. If you omit this parameter, an infinite wait time is assumed. |

**Returns**

The AFR_PEER_SENDRCV function returns a string, where received data from the peer system has been placed. If no data is received before the timeout value elapses or an error occurs during the send or receive operations, a null string is returned. A maximum of 32K of data can be received.

If a null string has been returned, you can use AFR_PEER_GETLASTERROR to find the exact reason for the AFR_PEER_SENDRCV error. Following are the possible returns codes:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The data was successfully sent and received.</td>
</tr>
<tr>
<td>1</td>
<td>An invalid parameter was specified on the AFR_PEER_SENDRCV.</td>
</tr>
<tr>
<td>2</td>
<td>An invalid peer environment exists; unable to send or receive the data.</td>
</tr>
<tr>
<td>4</td>
<td>The linkid specified is not known to this SA IOM system.</td>
</tr>
<tr>
<td>5</td>
<td>The AFR_PEER_SENDRCV failed to complete.</td>
</tr>
<tr>
<td>6</td>
<td>The timeout interval expired before any data was received.</td>
</tr>
<tr>
<td>8</td>
<td>The received data is invalid. For example, it is EBCDIC instead of ASCII, it does not adhere to the SA IOM TCP/IP data packet format, and so on.</td>
</tr>
<tr>
<td>9</td>
<td>DATARPLY encountered when another data type was expected.</td>
</tr>
<tr>
<td>100xx</td>
<td>A TCP/IP error occurred while sending/receiving data to or from the peer system.</td>
</tr>
</tbody>
</table>

**Remarks**

Return code 9 pertains to what might be called “conversation context”, such as the relationship between two peers and their changing roles as sender and receiver.

When using AFR_PEER_SENDRCV, you need to understand the current state of your peer conversations. In particular, conversations should not send a DATARPLY
record when no reply data has been requested and conversely, conversations should not send something other than a DATARPLY record when that is the only type of record the other peer is expecting.

To successfully use the special link ID: "NONAF", the PEER_PORT_NON_AFPACKET keyword must be defined in the server's RpSvrPrf.txt file.

Example

In the following example, a conversation is started with an AF/OPERATOR address space. A linkid of OGSYS1 is assigned to the conversation. The buffer being sent causes an OPER command to be issued by AF/OPERATOR. A TCP/IP receive is then done to retrieve up to 500 bytes of response data from the OPER command. The REXX script remains in a wait state, with no timeout, until OPER response data has been received. After displaying the returned data, the REXX script closes the linkid.

```rexx
/* REXX */
ret=AFR_PEER_OPEN("OGSYS1", "155.100.33.44", 2222)
if ret <> 0 then do
    /* perform application error processing and exit */
end
sendbuf = "OPER 'OGDB SHOW STATS' RESP"
rcvbuf = AFR_PEER_SENDRCV("OGSYS1", "EXEC", 500, sendbuf)
if rcvbuf <> "" then do
    say "SHOW STATS returned:" rcvbuf
else
    say "AFR_PEER_SENDRCV failed with rc:" AFR_PEER_GETLASTERROR()
ret = AFR_PEER_CLOSE("OGSYS1")
```

See also

"AFR_PEER_QUERY" on page 70
"AFR_PEER_OPEN" on page 69
"AFR_PEER_SEND" on page 75
"AFR_PEER_CLOSE" on page 67

AFR_PROMPT

Use this function to display a pop-up message that includes an input field for a response. Typically, the message is sent to the REXX script owner. Optionally, you can use this function to send a message to all SA IOM operators and to the system administrator.

Format

```
AFR_PROMPT(message, timeout [, OPERATOR])
```

message This parameter lets you provide message text for the pop-up prompt. The maximum length is 240 characters.

timeout This parameter specifies the length of time, in seconds, that the message prompt remains displayed at the client. However, with each keystroke of data that is typed into the input field, the timeout is reset. If -1 (minus 1) is entered, an indefinite wait occurs for input data.

OPERATOR This optional parameter lets you specify that the message prompt is addressed to all users who are authorized to receive messages addressed to the "operator" of the system (users having authority to "respond to client messages". The literal string "operator" is the expected value of this parameter.
AFR_PROMPT

Returns

This function returns a character string from the input field entered by the user. The maximum length is 240 characters.

Remarks

The pop-up message with the input field appears on the top of the targeted user's screen. The message disappears when any of the following occurs:

- The targeted user presses Enter.
- The timeout value expires.
- The calling REXX thread is interrupted by another user.

The user's response is case-sensitive. If your REXX script requires this input then ensure that you code it so that it gets the correct response.

If “operator” was not specified as the optional third parameter, and if the REXX script owner is not logged onto SA IOM when this function is called, the function returns an empty string.

If multiple users having authority to "respond to client messages" are logged on, only the first response is returned to the REXX program.

Examples

Here is a typical example.

```rexx
name = AFR_PROMPT("Enter your name:",30)
say "Your name is " || name
```

The following example shows how you can use this function's optional third parameter.

```rexx
response = AFR_PROMPT("Is there an operator who can help me?",30,"OPERATOR")
```

See also

"AFR_BROADCAST" on page 20
"AFR_ERROR" on page 41
"AFR_OP_ERROR" on page 66
other message and logging functions listed in "REXX functions, by their use" on page 7

AFR_PROMPT_USER

Use this function to send a pop-up message window to a specified SA IOM user. The message window includes an input field in which the user can type a response.

Format

`AFR_PROMPT_USER(message, timeout, userid)`

- **message**: This parameter lets you provide message text for the prompt. The maximum length is 240 characters.
- **timeout**: This parameter specifies the length of time, in seconds, that the message prompt remains displayed at the client. However, with each keystroke of data that is typed into the input field, the timeout is reset. If -1 (minus 1) is entered, an indefinite wait occurs for input data.
AFR_PROMPT_USER

**userid**  This parameter specifies the SA IOM user identifier (user ID) of the user to whom the message is addressed.

**Returns**

This function returns a character string from the input field entered by the user. The maximum length is 240 characters.

**Remarks**

The pop-up message with the input field appears on the top of the targeted user’s screen. The message disappears when any of the following occurs:

- The targeted user presses Enter.
- The timeout value expires.
- The calling REXX thread is interrupted by another user.

The user's response is case-sensitive. If your REXX script requires this input then ensure that you code it so that it gets the correct response.

If the specified user is not logged on to SA IOM when this function is called, the function immediately returns a null string.

If multiple instances of the same user are logged on at the same time (possible if the server has been customized to allow it) then only the first response is returned to the REXX program.

**Examples**

In this example, the prompt is sent to the SA IOM user whose ID is Tortuga2.

```plaintext
response = AFR_PROMPT_USER("Are you ready to test it now?",30, "Tortuga2")
say "Tortuga2 says:"||response
```

**See also**

- `AFR_BROADCAST` on page 20
- `AFR_ERROR` on page 41
- `AFR_OP_ERROR` on page 66

Other message and logging functions listed in "REXX functions, by their use" on page 7.

---

**AFR_REOPEN_LOG**

Use this function to reallocate the current server log under a specified file name extension and start a fresh log file.

When you call AFR_REOPEN_LOG:

1. The current RPSERVER. extension (if present) is deleted.
2. RPSERVER.LOG is closed.
3. RPSERVER.LOG is named to RPSERVER. extension.
4. A new, empty RPSERVER.LOG is opened for recording messages.

**Format**

```plaintext
AFR_REOPEN_LOG(extension)
```
AFR_REOPEN_LOG

extension  Up to 3 characters used as a file name extension when renaming the current RPSERVER.LOG file. If you omit this parameter, the default extension .BAK is used.

Returns

0  The log file was reallocated successfully.
1  The specified extension is invalid.

Remarks

- When the renaming process is successful, the last message in the old log file (now renamed as RPSERVER. extension) and the first message in the new RPSERVER.LOG file records the successful reallocation of the log.
- If either RPSERVER.LOG or RPSERVER. extension is in use by another process when this function is called, the renaming process fails and AFR_REOPEN_LOG returns an internal return code, most likely:
  5  Access denied (RPSERVER. extension is in use).
  32 Sharing violation (RPSERVER.LOG is in use).

Should this occur, SA IOM continues to use RPSERVER.LOG and AFR_REOPEN_LOG records the rename failure in the log. In no case does SA IOM “lose” a message.

SA IOM includes a sample REXX program, MONTHLY.REX, that invokes AFR_REOPEN_LOG to rename the log file on the first day of the month. For descriptions of this and other sample programs, see the appendix, “SA IOM Sample Scripts” in the System Automation for Integrated Operations Management User’s Guide.

AFR_RXQUEUE_CREATE

Use this function to create a new named REXX queue.

Format

AFR_RXQUEUE_CREATE(queuename)

queuename  A 1-32 alphanumeric parameter that specifies the name of a queue.

Returns

If a failure occurs, a null value is returned.

Remarks

If the named queue exists, a new queue with the generated name is created. The created queue is an internal queue and cannot be used to communicate with programs running outside of SA IOM.

Example

NEWQ = AFR_RXQUEUE_CREATE(CMD_QUEUE)
AFR_RXQUEUE_DELETE

Use this function to delete the internal named REXX queue.

Format

```plaintext
AFR_RXQUEUE_DELETE(queuename)
```

`queuename` A 1-32 alphanumeric parameter that specifies the name of a queue.

Returns

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>5</td>
<td>Invalid queue name</td>
</tr>
<tr>
<td>9</td>
<td>Named queue does not exist</td>
</tr>
<tr>
<td>10</td>
<td>Queue in use (current for other REXX program)</td>
</tr>
</tbody>
</table>

Remarks

If the named queue exists and is an internal queue, the queue is deleted. If the queue is an external queue, the named queue does not exist; use RXQUEUE("DELETE") to delete an external queue.

You cannot delete an internal named queue if the queue is current (in use) for some REXX program.

Example

```plaintext
RET = AFR_RXQUEUE_DELETE(CMD_QUEUE)
```

See also

"AFR_RXQUEUE_EXISTS" on page 85

AFR_RXQUEUE_EXISTS

Use this function to inquire if the specified REXX queue exists.

Format

```plaintext
AFR_RXQUEUE_EXISTS(queuename)
```

`queuename` A 1 to 32 character alphanumeric parameter that specifies the name of a queue.

Returns

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Queue currently exists</td>
</tr>
<tr>
<td>1</td>
<td>Queue does not exist</td>
</tr>
</tbody>
</table>
AFR_RXQUEUE_EXISTS

Remarks
Determines whether the specified internal queue currently exists.

Example

RET = AFR_RXQUEUE_EXISTS(CMD_QUEUE)

AFR_RXQUEUE_EXTERNAL

Use this function to declare the specified queue to be defined as “external” and ignored by SA IOM.

Format

AFR_RXQUEUE_EXTERNAL(queueName)

queueName  A 1 to 32 character alphanumeric parameter that specifies the name of a queue.

Returns

0    Success
1    Already declared to be external
2    Already exists as an internal queue

Remarks
Declares the specified queue to be defined as an external queue. Use RXQUEUE('CREATE') to create an external queue.

Example

RET = AFR_RXQUEUE_EXTERNAL(CMD_QUEUE)

AFR_RXQUEUE_GET

Use this function to retrieve the name of the current REXX queue.

Format

AFR_RXQUEUE_GET()

Returns

If a failure occurs, a null value is returned.

Remarks
The name of the REXX program’s current internal queue is returned. If the current queue is the default queue, the queue name SESSION is returned.

Example

CURQUE = AFR_RXQUEUE_GET()
AFR_RXQUEUE_PURGE

Use this function to empty an internal queue.

Format

AFR_RXQUEUE_PURGE(queuename)

queuename A 1 to 32 character alphanumeric parameter that specifies the name of a queue.

Returns

0 Success
1 Queue not found

Remarks

Empties the specified internal queue, whether or not it is current for some REXX program.

Example

RET = AFR_RXQUEUE_PURGE(CMD_QUEUE)

AFR_RXQUEUE_SET

Use this function to make the specified REXX queue current.

Format

AFR_RXQUEUE_SET(queuename)

queuename A 1 to 32 character alphanumeric parameter that specifies the name of a queue.

Returns

If a failure occurs, a null value is returned.

Remarks

If the internal or external named queue exists, it is made current and the name of the previously current queue is returned. If the specified queue does not exist it is created, unless it was previously declared as an external queue using AFR_RXQUEUE_EXTERNAL.

External queues can be used to communicate with programs running outside of SA IOM.

Example

OLDQ = AFR_RXQUEUE_SET(CMD_QUEUE)

See also

"AFR_RXQUEUE_EXTERNAL" on page 84
AFR_SAY

Use this function to produce formatted screen output.

**Format**

AFR_SAY(data)

- **data** This parameter is an ASCII character string for screen output. It may contain ANSI escape commands.

**Returns**

This function displays data on the active REXX session screen.

**Remarks**

Use this function to format screen output (in a ANSI terminal emulation program, for example).

AFR_SAY is similar to the SAY REXX instruction. SAY adds a line feed character to the end of the string, AFR_SAY does not.

**Example**

```plaintext
nul = AFR_SAY(data)
```

**See also**

“AFR_CLS” on page 24,  
“AFR_KBHIT” on page 48,  
“ANSI escape sequence support” on page 3

AFR_SCREEN_WAIT

Use this function to pause program execution until the selected emulation session screen changes or until the specified timeout value expires.

**Format**

AFR_SCREEN_WAIT(delay)

- **delay** This parameter specifies the time in milliseconds that the function will wait for a change in the emulation session screen. The maximum value for this parameter is $2^{31}-1$ millisecond.

**Returns**

- **-1** No emulator session is selected.
- **0** A screen change occurred.
- **1** The specified timeout value expired without a screen change.
- **2** The script thread is terminating due to a server shutdown or script Halt command.

**Remarks**

This function waits for the screen to change as follows:
AFR_SCREEN_WAIT

- If the screen change occurs before the timeout expires, 0 (zero) is returned to the calling program.
- If the timeout expires before a screen change occurs, 1 is returned to the calling program.

This function will return when either the content of the screen changes by at least one character or the timeout delay expires.

This function begins its wait for the next screen update at the time it is called.

The desired emulation session must have been previously made active with the function described in “AFR_SELECT” on page 119 (“AFR_SELECT”).

Example

RET = AFR_SCREEN_WAIT(30000)

This example specifies a timeout value of 30 seconds (30000 milliseconds).

See also

“AFR_LOAD” on page 50
“AFR_SELECT”

AFR_SELECT

Use this function to make the specified server emulation session the current session. Session currency is necessary to use session read/write functions such as AFR_SEND_ASCII.

Format

AFR_SELECT(session)

session The name of the SA IOM emulation session. The name is case sensitive and may include imbedded blanks. Enclose this parameter in quotes.

Note: This is the Name value supplied when the Host Session was defined.

Returns

0 The session is successfully selected.
1 An unspecified error was encountered.

Remarks

Any function (in the session control group - refer to "REXX functions, by their use" on page 7) that performs a read or write to a session must first select the session using AFR_SELECT.

Within a REXX script, selecting a second session using AFR_SELECT deselects the first.

All subsequent functions that deal with reading or writing to a session buffer will operate only on this session (except AFR_SEND when used with another session name).
AFR_SELECT

If any traps or WTORs are outstanding, these will be cleared, as if AFR_CLEAR_TRAP functions were called for each.

Example

    RET = AFR_SELECT('A')
    RET = AFR_SEND_3270("LOGON")

In the example above, the string LOGON would be sent to the 3270 adapter named A.

In the example below, the first 1920 bytes of the Message Collector session are searched for the 'system down' character string.

    RET = AFR_SELECT("Message Collector")
    screen = AFR_LOAD(0,1920)
    if pos('system down', screen) > 0 then
        do...

See also

"AFR_SET_TRAP" on page 96
"AFR_ASYNC_WTOR" on page 19
"AFR_SEND" on page 89
"AFR_SEND_3270" on page 90
"AFR_SEND_ASCII" on page 92
"AFR_SET_WTOR" on page 101

AFR_SELECTED

Use this function to retrieve the name of the currently selected emulation session.

Format

    AFR_SELECTED()

This function accepts no parameters.

Returns

Upon successful completion, this function returns the name of the currently selected emulation session.

If no emulation session is selected, a null string is returned.

Passing any parameters will cause the standard REXX error.

Remarks

Any function (in the session control group - refer to "REXX functions, by their use" on page 7) that performs a read or write to a session must first select the session using AFR_SELECT.

Within a REXX script, selecting a second session using AFR_SELECT deselects the first.

You can use the AFR_SELECTED function to check that the session you are about to perform some operation on is the correct session.
AFR_SEND

Use this function to send a string to a specified emulation session. When you use AFR_SEND, you do not have to specify the session receiving the string with a previous AFR_SELECT function call.

Format

AFR_SEND(session,string)

session  The name of the SA IOM emulation session. The name is case sensitive and may include imbedded blanks. Enclose this parameter in quotes.

To send a string to the currently selected session, use a null parameter (""") for session. The specified string will be sent to the session last specified by an AFR_SELECT call. This is the recommended method.

string  This parameter specifies the characters to be written. Enclose the string in quotes. The maximum length is 255 characters. The string can also include the following HLLAPI codes:

HLLAPI Codes supported by AFR_SEND

@5 = ESCAPE     @7 = PF7     @g = PF16
@E = ENTER      @8 = PF8     @h = PF17
@@ = @          @9 = PF9     @i = PF18
@1 = PF1        @a = PF10    @j = PF19
@2 = PF2        @b = PF11    @k = PF20
@3 = PF3        @c = PF12    @l = PF21
@4 = PF4        @d = PF13    @m = PF22
@5 = PF5        @e = PF14    @n = PF23
@6 = PF6        @f = PF15    @o = PF24

Returns

0  The string is successfully sent.
1  An unspecified error was encountered.

Remarks

If you use the AFR_SEND function with a valid session name, the session name previously selected with an AFR_SELECT function call is deselected.

After an AFR_SEND function call, you may have to wait depending on the response time of the host system to which you are sending a message. When using AFR_SEND on an MVS session, monitor the Operator Information Area (OIA) for the X-clock to clear before sending another AFR_SEND command.
AFR_SEND

Example

Example 1:

The following example sends strings to two host sessions.
- a 3270 VTAM console defined as VTAM.
- a VT100 terminal emulation session on a COM port attached to a modem defined as terminal.

```rexx
/* REXX */

Console_3270 = "VTAM" /* 3270 Console Name */
Console_COM1 = terminal /* Terminal Console Name */
UserId = "TDSM12" /* data string */
rc = AFR_SEND(Console_3270, `@$k' || UserID)
    /* sends `ESC k' characters to 3270 */
    /* VTAM Console to unlock keyboard, */
    /* then send some data (UserID). */

rc = AFR_SEND(Console_COM1, `@1") /* sends keystroke data defined for */
    /* F1 key to the terminal. */
    /* If F1 contains a string */
    /* (`ATDT9.333-1212~<`), this string */
    /* will be dialed. */
```

Example 2:

The following example emphasizes you do not need to specify the session name for every AFR_SEND call.

```rexx
/* REXX */
rc = AFR_SELECT("SESA")
DO 10
    rc = AFR_SEND("","TEXT")
END
```

See also

"AFR_SELECT" on page 87
"AFR_SEND_3270"
"AFR_SEND_ASCII" on page 92

AFR_SEND_3270

Use this function to send a string to a 3270 emulation session.

Format

```rexx
AFR_SEND_3270(string)
```

string  This parameter specifies the characters to be written. Enclose the string in quotes. The string can include special commands or functions to a hardware console. The maximum length is 255 characters. See "AFR_SEND" on page 89 for the modified HLLAPI format.

Returns

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The string is successfully sent to a 3270 emulation session.</td>
</tr>
<tr>
<td>1</td>
<td>There is an error or the active session is not a 3270 emulation session.</td>
</tr>
</tbody>
</table>
Remarks

The desired emulation session buffer must have been previously made current with the function described in "AFR_SELECT" on page 87.

You can use the AFR_SEND_3270 function from within a REXX script to send a command or function that you would normally type at a hardware console. Many such commands or functions are accomplished at the hardware console by escape sequences: key combinations of the escape key (Esc) followed by a letter key.

Once you have found the correct escape key sequence, and have tested it by typing it at the console, begin the string with the characters @$ to send the Esc character, followed by the letter. For example, to simulate pressing “Esc” then “A” at the console, code the line AFR_SEND_3270('@$A') in your REXX script.

After an AFR_SEND_3270 function call, you may have to wait depending on the response time of the mainframe to which you are sending a message. When using AFR_SEND_3270 on an MVS session, monitor the Operator Information Area (OIA) for the X-clock to clear before sending another AFR_SEND_3270 command.

Example

RET = AFR_SEND_3270('userid@E')

The ASSIGN CONSOLE function on the 3090 hardware console could be done with the following:
RET = AFR_SEND_3270('@$H')

The 3090 hardware console could be started with the following:
RET = AFR_SEND_3270('@$P')

and stopped with the following:
RET = AFR_SEND_3270('@$W')

AFR_SEND_3270 can send multiple values with a single call.

See also

"AFR_SELECT" on page 87
"AFR_SEND" on page 89
"AFR_SEND_SCAN" on page 96

Table 5. AFR_SEND_3270 Modified HLLAPI codes

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@&lt;</td>
<td>BACKSPACE</td>
</tr>
<tr>
<td>@]</td>
<td>]</td>
</tr>
<tr>
<td>@B</td>
<td>TAB_BACK</td>
</tr>
<tr>
<td>@C</td>
<td>CLEAR</td>
</tr>
<tr>
<td>@D</td>
<td>DELETE</td>
</tr>
<tr>
<td>@E</td>
<td>ENTER</td>
</tr>
<tr>
<td>@F</td>
<td>ERASE.EOF</td>
</tr>
<tr>
<td>@I</td>
<td>INSERT</td>
</tr>
<tr>
<td>@L</td>
<td>LEFT</td>
</tr>
<tr>
<td>@N</td>
<td>NEWLINE</td>
</tr>
<tr>
<td>@0</td>
<td>HOME</td>
</tr>
<tr>
<td>@1</td>
<td>PF1</td>
</tr>
<tr>
<td>@2</td>
<td>PF2</td>
</tr>
<tr>
<td>@3</td>
<td>PF3</td>
</tr>
<tr>
<td>@4</td>
<td>PF4</td>
</tr>
<tr>
<td>@5</td>
<td>PF5</td>
</tr>
<tr>
<td>@6</td>
<td>PF6</td>
</tr>
<tr>
<td>@7</td>
<td>PF7</td>
</tr>
<tr>
<td>@8</td>
<td>PF8</td>
</tr>
<tr>
<td>@9</td>
<td>PF9</td>
</tr>
<tr>
<td>@@</td>
<td>=</td>
</tr>
</tbody>
</table>
AFR_SEND_3270

Table 5. AFR_SEND_3270 Modified HLLAPI codes (continued)

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@P</td>
<td>PRINT</td>
</tr>
<tr>
<td>@R</td>
<td>RESET</td>
</tr>
<tr>
<td>@T</td>
<td>TAB</td>
</tr>
<tr>
<td>@U</td>
<td>UP</td>
</tr>
<tr>
<td>@V</td>
<td>DOWN</td>
</tr>
<tr>
<td>@W</td>
<td>ESCAPE</td>
</tr>
<tr>
<td>@X</td>
<td>SYS_REQ</td>
</tr>
<tr>
<td>@Y</td>
<td>TEST</td>
</tr>
<tr>
<td>@Z</td>
<td>RIGHT</td>
</tr>
<tr>
<td>@a</td>
<td>PF10</td>
</tr>
<tr>
<td>@b</td>
<td>PF11</td>
</tr>
<tr>
<td>@c</td>
<td>PF12</td>
</tr>
<tr>
<td>@d</td>
<td>PF13</td>
</tr>
<tr>
<td>@e</td>
<td>PF14</td>
</tr>
<tr>
<td>@f</td>
<td>PF15</td>
</tr>
<tr>
<td>@g</td>
<td>PF16</td>
</tr>
<tr>
<td>@h</td>
<td>PF17</td>
</tr>
<tr>
<td>@s</td>
<td>ESCAPE</td>
</tr>
<tr>
<td>@A</td>
<td>CUR_SEL</td>
</tr>
<tr>
<td>@@F</td>
<td>ERASE_INP</td>
</tr>
<tr>
<td>@@H</td>
<td>SYS_REQ</td>
</tr>
<tr>
<td>@@J</td>
<td>CUR_SEL</td>
</tr>
<tr>
<td>@@Q</td>
<td>ATTN</td>
</tr>
<tr>
<td>@@x</td>
<td>DUP</td>
</tr>
<tr>
<td>@@y</td>
<td>FIELD_MARK</td>
</tr>
</tbody>
</table>

AFR_SEND_ASCII

Use this function to send an ASCII string to the active emulation session.

Format

AFR_SEND_ASCII(string,crfeed)

string This parameter specifies the characters to be written. Enclose the string in quotes.

crfeed This parameter specifies whether (1) or not (0) to append an ASCII carriage return character (CR) to the end of the string.

Returns

0 The string is successfully sent to an ASCII session.
1 There is an error or the active session is not an ASCII emulation session.

Remarks

The desired emulation session buffer must have been previously made active with the function described in "AFR_SELECT" on page 87.

Example

RET = AFR_SEND_ASCII('This is a teststring',0)

See also

"AFR_SELECT" on page 87
"AFR_SEND" on page 89

AFR_SEND_EMAIL

Use this function to send a message to one or more recipients using e-mail. Optional parameters for carbon copy (cc) and blind carbon copy (bcc) components are included.
AFR_SEND_EMAIL

Format

Format 1:
AFR_SEND_EMAIL(domain, from, to, [cc], [bcc], subject, text)

Format 2:
AFR_SEND_EMAIL(domain, from, to, cc, bcc, subject, text)

- **domain**: This is the name of the e-mail server.
- **from**: This is the e-mail address of the sender.
- **to**: This is the e-mail address of the recipient.
- **cc**: This is the e-mail address of the recipient receiving a carbon copy (cc).
- **bcc**: This is the e-mail address of the recipient receiving a blind carbon copy (bcc).
- **subject**: This parameter contains the subject line text.
- **text**: This parameter contains the text of the message.

Returns

- **0**: The e-mail was sent successfully.
- **1**: An initialization error occurred.
- **2**: A socket could not be created.
- **3**: The domain is unknown.
- **5**: Connection to the host was unsuccessful.
- **10**: The domain address is too long.

**Note:** All other return codes are codes returned by SMTP. For further information, refer to the RFC 0821 document which describes the widely implemented Simple Mail Transfer Protocol (SMTP).

Remarks

This function sends a message, using e-mail, and optionally carbon copies and optionally blind carbon copies one or more recipients. Optional parameters can be omitted, however, if specified they cannot be a null value (a space value is permitted).

If more than one recipient is specified for the to, the cc, or the bcc parameter; then the addresses must be separated by a semi-colon (:) as shown in example 2 below.

Example

**Example 1:**

In the first example Jim uses the domain companyb.com to send a test message to jtest.dev1.com.

```python
rc = AFR_SEND_EMAIL('companyb.com', 'jim@ibm.com', 'jtest.dev1.com', 'Test Message', 'This is a test')
```

**Example 2:**

In the next example Jim, using the domain server.com, sends message text to two recipients and a bcc to another recipient.
AFR_SEND_EMAIL

```c
rc = AFR_SEND_EMAIL('server.com','jim@ibm.com;Curt@ibm.com', 'hansW@dev.com',
'kurt@dev.com','Message with Bcc','message text')
```

See also

"AFR_SEND_MESSAGE"

AFR_SEND_MESSAGE

Use this function to send messages to a Message Collector session through the TCP/IP network, or to send an e-mail message over the Internet.

Format

You can use one of the following two formats. The first format:

```c
AFR_SEND_MESSAGE(domain:port, text)
```

*domain*  
The IP address of the TCP/IP domain to which SA IOM is connected.

*port*  
The Message Collector port number.

*text*  
Any text message. If the text uses more than one line, separate the lines with `<CR><LF>`.

The second format:

```c
AFR_SEND_MESSAGE(domain, from, to, subject, data)
```

*domain*  
The IP address or, if your TCP/IP network is configured for name resolution, the name of the TCP/IP domain to which SA IOM is connected. If the port number is not specified, number 25 (the standard for SMTP) is assumed.

*from*  
The user ID of the sender.

*to*  
The user ID of the destination.

*subject*  
The subject text for the message.

*data*  
Any text message. If the text uses more than one line, separate the lines with `<CR><LF>`.

The second format includes the sender's return path (IP address), and the sender's time and date stamps.

Returns

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The message is successfully sent.</td>
</tr>
<tr>
<td>1</td>
<td>An error occurred while initializing.</td>
</tr>
<tr>
<td>3</td>
<td>An error occurred while resolving the host name.</td>
</tr>
<tr>
<td>5</td>
<td>An error occurred while connecting to a host.</td>
</tr>
</tbody>
</table>

Remarks

Use this function to send messages from one SA IOM system to another. You can use AFR_LOAD or AFR_GET_LINE to get a message. You can use the trapping functions (AFR_SET_TRAP, AFR_CHECK_TRAP) to trap a message, take an appropriate action, and then respond to the message (using the return path) to report the results of the action.
AFR_SEND_MESSAGE

Example

Example 1:

/* Sending a message to the Message Collector */
/* of another SA IOM host using port 777 */

say AFR_SEND_MESSAGE("119.0.15.32:777", "This is a test message.")

Example 2:

;/* Sending e-Mail with a return-path and date/time stamp */

CrLf = d2c(13) || d2c(10) /* define <CR><LF> */
from = "user1@domain1.com" /* tell return path */
to = "user2@domain2.com"
Subject = "Testing e-mail"
data = "This is the first line." ||CrLf|| "And yet another line."

say AFR_SEND_MESSAGE("172.123.32.2", from, to, Subject, data)

See also

"AFR_CHECK_TRAP" on page 21
"AFR_GET_LINE" on page 42
"AFR_LOAD" on page 50
"AFR_SET_TRAP" on page 96

AFR_SEND_RPAGE

Use this function to send a message to a pager.

Format

AFR_SEND_RPAGE(snpp.domain, PID, message)

| snpp.domain | This parameter is the domain address obtained from a pager service provider. |
| PID | This is the pager ID. |
| message | This parameter contains the message text. |

Returns

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A message was sent to the pager successfully.</td>
</tr>
<tr>
<td>1</td>
<td>An initialization error occurred.</td>
</tr>
<tr>
<td>2</td>
<td>A socket could not be created.</td>
</tr>
<tr>
<td>3</td>
<td>The domain is unknown.</td>
</tr>
<tr>
<td>5</td>
<td>Connection to the host was unsuccessful.</td>
</tr>
<tr>
<td>10</td>
<td>The domain address is too long.</td>
</tr>
</tbody>
</table>

Note: All other return codes are codes returned by SMTP/SNPP.

Remarks

This function initiates radio (BEEPER) pages through TCP/IP using the SNPP (Simple Network Paging Protocol). Your paging service must support SNPP to use this function.
**AFR_SEND_RPAGE**

**Example**

In the following example a message is sent from the domain, snpp.airtouch.com:444, to the pager 3107274124.

\[rc = AFR\_SEND\_RPAGE('snpp.airtouch.com:444', '3107274124', msg)\]

**See also**

"AFR\_SELECT" on page 87
"AFR\_SEND" on page 89
"AFR\_SEND\_3270" on page 90

---

**AFR_SEND_SCAN**

Use this function to send a single value directly to a 3270 emulation adapter.

**Format**

\[AFR\_SEND\_SCAN(byte)\]

\[\text{byte}\]

This parameter specifies the scan code.

**Returns**

0 The scan code is successfully sent.

1 There is an error or the active session is not a 3270 emulation session.

**Remarks**

This rarely needed low-level function allows a single number (the scan code) to be sent to the previously selected 3270 session.

The desired emulation session buffer must have been previously made active with the function described in "AFR\_SELECT" on page 87.

**Example**

The value

\[RET = AFR\_SEND\_SCAN(210)\]

will perform the ASSIGN CONSOLE function on the 3090 hardware console.

**See also**

"AFR\_SELECT" on page 87
"AFR\_SEND" on page 89
"AFR\_SEND\_3270" on page 90

---

**AFR_SET_TRAP**

Use this function to set a trap for a string on the screen. SA IOM will begin scanning for the specified target.

**Format**

\[AFR\_SET\_TRAP(target, begin, end, offset, length)\]
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>This parameter specifies a string to search for. It can be either a standard string enclosed in quotes or a variable that evaluates to a string. Attribute bytes are ignored. The string is case sensitive.</td>
</tr>
<tr>
<td>begin</td>
<td>This parameter specifies the beginning of the search in the active emulation buffer. The size of the buffer depends on the screen size of the selected emulator. If your emulation session screen size is model 2, the session buffer spans 2000 characters in the range from 0 to 1999 (80 characters per row, times 25 rows). If your emulation session screen size is model 4, the session buffer spans 3520 characters in the range from 0 to 3519 (80 characters per row, times 44 rows).</td>
</tr>
<tr>
<td>end</td>
<td>This parameter specifies the end point of the search in the active emulation buffer.</td>
</tr>
<tr>
<td>offset</td>
<td>This parameter specifies where the string (that is to be saved when the trap fires) starts in the buffer. It is a positive or negative value specified relative to where the first character of the target is found. (Thus an offset of zero means to save the string starting with the first character of the target.)</td>
</tr>
<tr>
<td>length</td>
<td>This parameter specifies the length in bytes of the string (that is to be saved when the trap fires).</td>
</tr>
</tbody>
</table>

### Returns

- **>0** Handle, identifying the trap.
- **-1** An error occurred.
- **-2** Invalid parameter.

### Remarks

Once the target is trapped, a string of the specified length, starting at the specified offset, is saved for later retrieval by the function described in "AFR_CHECK_TRAP" on page 21.

The trap is cleared when another session is selected with the AFR_SELECT function or when the program that set the trap is terminated.

### Example

The following is based on the AF/OPERATOR or OG/MVS message:

```
AOP3001 PEER INTERFACE INITIALIZATION STARTED
trap1 = AFR_SET_TRAP('AOP3001',0,1999,13,9)
```

When the target string 'AOP3001' is trapped, the function saves the string 'INTERFACE' and returns a unique handle identifying the trap.
AFR_SET_TRAP

See also

“AFR_CHECK_TRAP” on page 21
“AFR_CLEAR_TRAP” on page 23
“AFR_SELECT” on page 87
“AFR_SET_WTOR” on page 101

AFR_SET_TRAP2

Use this function to set a trap for a string on a session screen.

Format

AFR_SET_TRAP2(trapHistoryID, keyOffset, keyLength, targetStr, targetOffset [, retOffset [, retLength [, contStr [, contOffset [, beginLine [, endLine]]]]]])

trapHistoryID  This required parameter specifies a numeric identifier, unique to all REXX scripts running on this RpServer, that will be used by SA IOM to identify a trap history list. The trap history list is used by SA IOM to prevent duplicate trap firings. Best practices suggest using the results of AFR_WHOAMI(). The same trap history ID should be used by the REXX script for each distinct trap being set, that is, each AFR_SET_MSG_TRAP call with the same targetStr, beginLine, and endLine. Any integer value can be used, although simple counting numbers (1, 2, 3, and so on) represent the best programming practice.

keyOffset  This required parameter specifies where a string begins that is unique for each distinct message. It is defined relative to the beginning location of a console line and must occur on the first line of a multiple line message. It must be non-negative. Use a value of zero to specify the beginning of the line. For example, this offset might identify the start of a timestamp field in the message, in which case, the offset to specify might be 4.

keyLength  This required parameter specifies the length in characters of the unique key string. For example, this offset might identify the length of a timestamp field in the message.

targetStr  This required parameter specifies a string to search for. It can be either a standard string enclosed in quotes or a variable that evaluates to a string. Attribute bytes are ignored. This string is case sensitive. The target string must exist in the first line of a multiple line message.

targetOffset  This required parameter specifies the offset, in characters, from the beginning of each line in the search region that will be compared with the target string for equality. It must be non-negative. Use a value of zero to specify the beginning of the line.

retOffset  This optional parameter specifies where the string to be returned starts, relative to the beginning of the console line containing the found target string. It must be non-negative. If this parameter is omitted, then a default offset of zero is used.

retLength  This optional parameter specifies the length in characters of the string to be returned. If this parameter is omitted, then the remainder of the message to the end of the line is returned. If a value of zero is used, the remainder of the message, as defined by contStr and contOffset, will be returned. A maximum value of 480 can be used.

contStr  This optional parameter specifies a string that will be used to identify a continuation line of a message. For example, it might consist of eight blanks. If retLength is nonzero, this parameter is ignored. If this parameter is omitted or is given a null value, then a single line message is assumed.
AFR_SET_TRAP2

**contOffset**
This optional parameter specifies where the continuation string contStr begins, if the message has continuation lines. It is defined relative to the beginning location of a console line and must be non-negative. A value of zero means the first character of the console line. If retLength is nonzero, this parameter is ignored. If contStr is null, this parameter is ignored. If contStr is provided and this parameter is omitted, then a default offset value of zero is used.

**beginLine**
This optional parameter specifies the beginning console line of the search in the session buffer of the currently selected host session. The first (topmost) console line is line 1. If this parameter is omitted, a default value of 1 is used.

**endLine**
This optional parameter specifies the end console line of the search in the session buffer of the currently selected host session. If the host session uses 3270 emulation with a 3279 Model 2 screen size, and the search region is to include the entire 3279 screen but exclude the session status line appended by SA IOM, then a value of 24 would be used. If this parameter is omitted, a default value of 24 is used.

**Returns**

>0 Handle, identifying the trap

-1 Error occurred. Trap not set.

**Remarks**

AFR_SET_TRAP2 is used to set a trap for a string on the screen. It is useful for message oriented but non-WTOR trapping. It has the following behaviors.

- If a given trap has already fired, it will not fire again.
- Message text spanning an arbitrary number of console lines can be returned.
- A specific field on each line of the search region is inspected, rather than every character position in the search region.
- When the trap fires, the trap is automatically cleared and must be defined again.

This function returns a number that serves as a unique handle for subsequent AFR_CHECK_TRAP and AFR_CLEAR_TRAP functions.

If the value of the number is negative, there was an error and a valid trap was not set.

The desired emulation session buffer must have been previously made active with the function described in [AFR_SELECT](#) on page 87.

In its simplest form, this function can be used with only the first 5 parameters. A slightly different form of search and additional trapping control is realized with each additional parameter up to the maximum of 11. The following is a list of the various forms of search and how to implement each one.

**Single-Line Message**
Provide values for the first five parameters only (through targetOffset). Omit values for retOffset, retLength, contStr, contOffset, beginLine, and endLine. Use this form to search lines 1 through 24 and return the full line of message text.
### AFR_SET_TRAP2

<table>
<thead>
<tr>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-Line Partial Message</td>
<td>Provide values for the first six parameters only (through retOffset). Omit values for retLength, contStr, contOffset, beginLine, and endLine. Use this form to search lines 1 through 24 and return message text from retOffset to the end of the line.</td>
</tr>
<tr>
<td>Fixed-Length Message</td>
<td>Provide values for the first seven parameters only (through retLength). Omit values for contStr, contOffset, beginLine, and endLine. Use this form to search lines 1 through 24 and specify an exact length of message text to return. This length can span from the first line of the message to following lines.</td>
</tr>
<tr>
<td>Simple Continuation Message</td>
<td>Provide values for the first eight parameters only (through contStr). Omit values for contOffset, beginLine, and endLine. Use this form to search lines 1 through 24 and specify return of a message spanning a variable number of lines, where the string denoting a continuation line always begins in the first column.</td>
</tr>
<tr>
<td>Offset Continuation Message</td>
<td>Provide values for the first nine parameters only (through contOffset). Omit values for beginLine and endLine. Use this form to search lines 1 through 24 and specify return of a message spanning a variable number of lines, where the string denoting a continuation line does not begin in the first column.</td>
</tr>
<tr>
<td>Search Region Begins After Line 1</td>
<td>Provide values for the first ten parameters only (through beginLine). Omit a value for endLine. Use this form to search a region beginning after line 1 and continuing through line 24. You have full control over the return text and continuation line definition.</td>
</tr>
<tr>
<td>Search Region Ends Before Line 24</td>
<td>Provide values for all parameters. Use this form for full control of the search region, return text, and continuation line definition.</td>
</tr>
</tbody>
</table>

REXX programming hint: If your AFR_SET_TRAP2 call spans lines in your REXX source file, remember to include an extra comma at the end of the first line to indicate a continued statement.

### Results

Once the target string is trapped, the unique key string identified by keyOffset and keyLength is compared to the current list of trap entries in the trap history list. If an entry is found, then the trap is ignored, and the trap search continues until the end of the search region is encountered.

If an entry in the trap history list is not found, then a new entry in the trap history list is made, registering this trap as already fired. If retLength is nonzero, then retLength characters beginning at retOffset are returned.

If retLength is zero, then the message text beginning at retOffset and continuing until the end of the message, spanning one or more lines, will be returned. In this case, the end of the message will be determined by locating the first console line following the console line containing targetStr that does not contain the specified continuation string contStr.

### Examples

Set a trap on the word "OPSMAIN" and return the complete message following it.

The following illustration represents a few lines on a session screen. The session uses a Model 2 (24 line) console. The start of the timestamp field in first line of the illustration has at an offset of 4.
For the purpose of the following example, a trap history ID of 1 is used.

This example shows how the AFR_SET_TRAP2 function can be used with 9 parameters.

\[
\text{handle} = \text{AFR\_SET\_TRAP2}(1, 4, 8, '\text{OPSMAIN}', 29, 38, 0, ' ', 4)
\]

In this case, a successful AFR_CHECK_TRAP will return the string:

"Emer. OPSMAIN Shutdown. Contact A/S for restart"

See also

- "AFR\_CHECK\_TRAP" on page 21
- "AFR\_CLEAR\_TRAP" on page 23
- "AFR\_SELECT" on page 87
- "AFR\_WHOAMI" on page 131

AFR\_SET\_WTOR

Use this function to set a trap for a specified WTOR string. SA IOM will begin scanning for the specified target.

**Note:** This function can only be used with MVS 3270 sessions.

**Format**

\[
\text{AFR\_SET\_WTOR}(\text{target, begin, end, length})
\]

**target**

This parameter specifies a string to search for. Enclose the string in quotes.

**begin**

This parameter specifies the beginning of the search in the active emulation buffer. The size of the buffer depends on the screen size of the selected emulator. If your emulation session screen size is model 2, the session buffer spans 2000 characters in the range 0 to 1999 (80 characters per row, times 25 rows). If your emulation session screen size is model 4, the session buffer spans 3520 characters in the range 0 to 3519 (80 characters per row, times 44 rows).

**end**

This parameter specifies the end point of the search in the active emulation buffer.

**length**

This parameter specifies the length in bytes of the string that is saved for subsequent retrieval. The string starts at the beginning of the WTOR line.

**Returns**

- >0 Handle, identifying the trap.
- -1 An error occurred.
- -2 Invalid parameter.

**Remarks**

This function returns a number that serves as a unique handle for subsequent AFR\_CHECK\_WTOR and AFR\_CLEAR\_WTOR functions.
AFR_SET_WTOR

If the value of the number is negative, there was an error and a valid trap was not set.

The WTOR attribute byte is automatically checked by the function to prevent old, previously-answered WTORs from being trapped.

The desired emulation session buffer must have been previously made active with the function described in "AFR_SELECT" on page 87.

Results

Once the target is trapped, a string of the specified length, starting at the beginning of the WTOR line, is placed in an internal memory location for later retrieval by the function described in "AFR_CHECK_WTOR" on page 22.

The trap is cleared when another session is selected with the AFR_SELECT function or when the program that set the trap is terminated.

Example

Here is an example based on the WTOR sent by the AF/OPERATOR or OMEGACENTER Gateway for z/OS BEEP command:

```
trap = AFR_SET_WTOR('!AOBEEP',0,1999,60)
```

When the WTOR trigger '!AOBEEP' is trapped, the function saves the entire WTOR, `nnnn !AOBEEP T9,1-818-555-1212,,, 69345 00013' (plus trailing blanks out to column 60) and returns a unique handle identifying the trap.

See also

"AFR_CHECK_WTOR" on page 22
"AFR_CLEAR_WTOR" on page 23
"AFR_SELECT" on page 87

AFR_SETENV

Use this function to assign a string to the specified configuration data item.

Format

AFR_SETENV(variable,string)

variable One of the following SA_IOM configuration data items (described in "AFR_GETENV" on page 43).

- TA_CALL
- SYS_TIMEZONE

string This parameter specifies the characters to be written. Enclose the string in quotes.

Returns

0 The variable is successfully assigned.
1 There is an error.
Example

RET = AFR_SETENV('TA_CALL','05 00:00')

See also

"AFR_GETENV" on page 43

AFR_SHRACCESS

Use this function to gain access to a pre-allocated shared memory file if it was created outside of SA IOM by an independent application program.

Format

AFR_SHRACCESS(filename)

filename  This parameter specifies the name of a pre-allocated shared memory area using the DOS filename format (name.ext).

Returns

0  The memory file is successfully accessed (first time accessed).
1  The memory file is successfully accessed (subsequent accesses).
2  Memory file not found
3  Memory file access denied
5  Resources exhausted

Remarks

An AFR_SHRACCESS call gives all REXX scripts running under SA IOM access to the file that was created by a program external to SA IOM. You must have access to the shared memory file before you can use the AFR_SHRREAD, AFR_SHRWRITE, and AFR_SHRFREE functions.

If a shared memory file was created by a call to AFR_SHRALLOC, a call to AFR_SHRACCESS is not needed; all REXX scripts running under SA IOM already have access to the shared memory file. See "AFR_SHRALLOC" on page 104 for more on using the SA IOM shared memory functions.

Call AFR_SHRFREE to de-allocate the shared memory file when it is no longer needed.

Example

RET = AFR_SHRACCESS("ShrRam.0")

See also

"AFR_SHRALLOC" on page 104
"AFR_SHRREAD" on page 105
"AFR_SHRWRITE" on page 106
"AFR_SHRFREE" on page 105
AFR_SHRALLOC

Use this function to allocate a shared memory file that is accessible to all REXX scripts running under SA IOM and to external programs.

Format

AFR_SHRALLOC(filename,size)

filename  This parameter names a shared memory file, using the DOS filename format (name.ext). The AFR_SHRREAD, AFR_SHRWRITE, and AFR_SHRFREE functions all use this name to address the shared memory file.

size  This parameter specifies the size of the shared memory file in bytes.

Returns

0  The memory file is successfully allocated.
4  Memory file already exists
5  Resources exhausted

Remarks

An AFR_SHRALLOC call creates a globally accessible named shared memory file, and gives all REXX scripts running under SA IOM access to it. This shared memory file can be addressed by any other REXX script or any external program that knows the file's name.

If a shared memory file already exists, because it was created by an external program that communicates to the REXX script, use AFR_SHRACCESS to gain access to the memory file. A call to AFR_SHRACCESS is not needed if AFR_SHRALLOC is used to create the named shared memory file.

AFR_SHRALLOC gives you access to the memory file.

In general, a REXX script or a related group of REXX scripts uses either an AFR_SHRALLOC call or an AFR_SHRACCESS call, not both. The call is made once, sometime during initialization. Several REXX scripts can share the same named memory file. The memory file remains active until it is de-allocated by calling the AFR_SHRFREE function or until SA IOM shuts down.

Since the shared memory file persists in memory until it is freed, REXX scripts that run during different time periods can pass information to each other using a named memory file.

Call AFR_SHRFREE to de-allocate the shared memory file when it is no longer needed.

Example

SCRIPT1

RET = AFR_SHRALLOC("ShrRam.1",10)
RET = AFR_SHRWRITE("ShrRam.1",0,'0123456789')

pull  /* wait for SCRIPT2 to read */

RET = AFR_SHRFREE("ShrRam.1")
SCRIPT2

ShrRam = AFR_SHRREAD("ShrRam.1",0,10)
say ShrRam

SCRIPT1 calls AFR_SHRALLOCC to create a shared memory file named ShrRam.1 and writes into it. SCRIPT2, at a later time, reads out of ShrRam.1. The last line in SCRIPT2:
say ShrRam

will display the 0123456789 written by SCRIPT1 into the shared memory.

See also

"AFR_SHRACCESS" on page 103
"AFR_SHRREAD" on page 106
"AFR_SHRFREE"

AFR_SHRFREE

Use this function to deallocate a shared memory file.

Format

AFR_SHRFREE(filename)

filename This parameter specifies the name of a pre-allocated shared memory file using the DOS filename format (name.ext).

Returns

0 The memory file is successfully de-allocated.
1 There is an error.

Remarks

The shared memory file must exist and you must have access to it before using this function. If a shared memory file is not de-allocated with an AFR_SHRFREE call, the file remains until SA IOM shuts down. Depending on how you accessed the file, more than one call to AFR_SHRFREE may be necessary to deallocate it. See "AFR_SHRALLOCC" on page 104 for more information on using the SA IOM shared memory functions.

Example

RET = AFR_SHRFREE("ShrRam.1")

See also

"AFR_SHRALLOCC" on page 104

AFR_SHRREAD

Use this function to read a specified number of bytes from a shared memory file.
AFR_SHRREAD

Format

AFR_SHRREAD(filename,offset,bytes)

filename  This parameter specifies the name of a pre-allocated shared memory file using the DOS filename format (name.ext).
offset    This parameter specifies where the string starts in the shared memory file. It is expressed in bytes from the beginning of the file.
bytes     This parameter specifies the number of bytes to read.

Returns

This function returns a null string if there is an error.

Remarks

The shared memory file must exist and you must have access to it before using this function. See "AFR_SHRALLOC" on page 104 for more information on using the SA IOM shared memory functions.

Example

RET = AFR_SHRREAD("ShrRam.1",0,13)

See also

"AFR_SHRALLOC" on page 104

AFR_SHRWRITE

Use this function to write a specified number of bytes to a shared memory file.

Format

AFR_SHRWRITE(filename,offset,string)

filename  This parameter specifies the name of a pre-allocated shared memory file using the DOS filename format (name.ext).
offset    This parameter specifies where the string starts in the shared memory file. It is expressed in bytes from the beginning of the file.
string    This parameter specifies the characters to be written. Enclose the characters in quotes.

Returns

0         The string is successfully written.
1         There is an error.

Remarks

The shared memory file must exist and you must have access to it before using this function. See "AFR_SHRALLOC" on page 104 for more information on using the SA IOM shared memory functions.

Example

RET = AFR_SHRWRITE("ShrRam.1",500,'test string')
**AFR_START_PROGRAM**

Use this function to start a REXX script.

**Format**

```plaintext
AFR_START_PROGRAM(startup)
```

*startup*  This parameter specifies as a string the script to start. This string consists of the script filename and startup parameters. If the script is located in the `\scripts` subdirectory, the filename must be the full filename and if the extension is not .rex then the extension must also be specified. For example, in the `\scripts` directory, to start program `myscript.rex`, specify `("myscript")` to start program `myscript.fil` specify `("myscript.fil")`. If the script resides in some other directory, then you must specify the full path name of the script `("c:\rexx\afr\myscript.rex")`. The entire string must be enclosed in quotes.

**Returns**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The script is successfully started.</td>
</tr>
<tr>
<td>non-0</td>
<td>There is an error.</td>
</tr>
</tbody>
</table>

**Example**

```plaintext
RET = AFR_START_PROGRAM('TEST.REX123 ' )
```

---

**AFR_STOP_PROGRAM**

Use this function to stop the specified REXX script. The REXX program is specified by its SA IOM script ID number.

**Format**

```plaintext
AFR_STOP_PROGRAM(script_id)
```

*script_id*  This parameter is the SA IOM script ID number assigned to the REXX program. The script ID is displayed after the REXX script name on the Scripts control panel and also on the title bar of an opened script session window.

**Returns**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Stopped, already stopped, or never started</td>
</tr>
<tr>
<td>1</td>
<td>Not stopped, security violation, see Usage below</td>
</tr>
</tbody>
</table>

This function may also return the standard REXX error numbers 26 and 40.

**Remarks**

Use the **AFR_STOP_PROGRAM** function to stop a REXX script from within a REXX script.
AFR_STOP_PROGRAM

The AFR_STOP_PROGRAM function has built-in security to prevent an unauthorized user from stopping a REXX script. The following situation causes the function to return a 1, indicating a security violation, and the target script is not stopped:

- The REXX script calling AFR_STOP_PROGRAM was neither started by a user with script management authority nor started by the same owner as the target script.

To stop a program AFR_STOP_PROGRAM must be called by the owner of the script to be stopped or by a user with script management authority. See (“REXX Script Owner” on page 26).

Example

\[ RET = \text{AFR\_STOP\_PROGRAM}(1) \]

See also

“AFR\_WHOAMI” on page 131

AFR_SUSPEND

Use this function to pause script execution until the specified timeout value expires.

Format

\[ \text{AFR\_SUSPEND}(\text{delay}) \]

\text{delay} \quad \text{This parameter specifies, in milliseconds, the time to wait before continuing.}

Returns

0 \quad \text{The delay is successful.}
1 \quad \text{The delay is not successful.}

Example

\[ RET = \text{AFR\_SUSPEND}(2000) \]

AFR_TELNET_CLOSE

Use this function to terminate a connection with a Telnet host.

Format

\[ \text{AFR\_TELNET\_CLOSE}() \]

Returns

0 \quad \text{The connection was successfully terminated.}
1 \quad \text{An invalid parameter was specified on the AFR\_TELNET\_CLOSE.}
2 \quad \text{An invalid REXX environment exists.}
3 \quad \text{An AFR\_SELECT was not previously done for a Telnet session.}
4 \quad \text{The AFR\_TELNET\_CLOSE failed to complete.}
AFR_TELNET_CLOSE

100xx The TCP/IP error occurred while closing the connection to the specified host.

Remarks

AFR_TELNET_CLOSE terminates whatever host connection is active. Therefore, it is not necessary to specify the name of the Telnet host from which to disconnect. It is not considered an error condition to call AFR_TELNET_CLOSE if there is no active connection; a return code of 0 will still be returned.

Example

/*REXX*/
   RET = AFR_TELNET_CLOSE()
   If RET <> 0 then
     say "AFR_TELNET_CLOSE failed with RC:" RET
   End
   Exit

See also

"AFR_SELECT" on page 87

AFR_TELNET_GETLASTERROR

Use this function to obtain the return code from the last executed Telnet REXX function. The Telnet session must have been previously made available through an AFR_SELECT.

Format

AFR_TELNET_GETLASTERROR()

Returns

This function returns the error code from the last Telnet REXX function call. If the last Telnet REXX function completed successfully, this value can be zero.

See also

"AFR_SELECT" on page 87

AFR_TELNET_OPEN

Use this function to start a connection to a Telnet host. The Telnet session must have been previously made available through an AFR_SELECT.

Format

AFR_TELNET_OPEN(hostname, portnumber)

hostname The IP address of the Telnet host with which you want to establish a session. This can be either a dotted decimal IP address or a hostname defined to your network's domain name server. If it is a dotted decimal IP address, none of the four numeric levels of the address can exceed 255.

portnumber The port number of the Telnet host. If omitted, the default Telnet port number of 23 is assumed.
AFR_TELNET_OPEN

Returns

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The connection was successfully started.</td>
</tr>
<tr>
<td>1</td>
<td>An invalid parameter was specified on the AFR_TELNET_OPEN.</td>
</tr>
<tr>
<td>2</td>
<td>An invalid REXX environment exists.</td>
</tr>
<tr>
<td>3</td>
<td>An AFR_SELECT was not previously done for a Telnet session.</td>
</tr>
<tr>
<td>4</td>
<td>The AFR_TELNET_OPEN failed to complete.</td>
</tr>
<tr>
<td>5</td>
<td>An active connection already exists for the current Telnet session.</td>
</tr>
<tr>
<td>100xx</td>
<td>A TCP/IP error occurred while connecting to the specified host.</td>
</tr>
</tbody>
</table>

Remarks

The TCP/IP error code value will usually be a very important piece of diagnostic data in any failed AFR_TELNET_OPEN call. TCP/IP error codes, per the Windows socket specification, are 5-digit numbers in the range of 10000 to 11000. For example, an attempt to connect to a Telnet host that is not running will fail with a 10061. For a list of TCP/IP error codes, see the appendix “TCP/IP Error Codes” in the System Automation for Integrated Operations Management User's Guide.

If you want to automate the process of logging on to a Telnet host and issuing commands on that host, combine AFR_TELNET_OPEN with AFR_SEND. In a typical scenario, after a successful AFR_TELNET_OPEN, a series of AFR_SEND calls could be implemented to send a user ID and password followed by one or more commands, and then if appropriate, an AFR_TELNET_CLOSE.

Example

```rexx
/* REXX */
parse arg hostname portnumber
ret = AFR_SELECT(Telnet1)
if ret <> 0 then
   do
      say "AFR_SELECT error:" || ret
      exit
   end
ret = AFR_TELNET_OPEN(hostname, portnumber)
if ret <> 0 then
   do
      say "AFR_TELNET_OPEN error:" || ret
      exit
   end
AFR_SUSPEND(2000)  /* Allow time for connection to complete */
AFR_SEND(Telnet1, "myUserid@E")
AFR_SUSPEND(2000)
AFR_SEND(Telnet1, "myPswd@E")
AFR_SUSPEND(2000)  /* Allow time for logon to complete */
```

See also

“AFR_SELECT” on page 87
“AFR_SEND” on page 89
“AFR_TELNET_CLOSE” on page 108

AFR_TELNET_QUERY

Use this function to obtain detailed information, including the current connect/disconnect status, of any configured Telnet session.
AFR_TELNET_QUERY

Format

AFR_TELNET_QUERY(keyword)

keyword The type of Telnet session data to return.

Returns

This function returns either a string of the type requested in the keyword parameter or a null string. Following is a list of supported keyword parameters and their possible return strings:

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS</td>
<td>&quot;Connected to xxxxx since yyyyyy&quot;, where xxxxx is the currently connected Telnet hostname and yyyyyy is the date and time when the connection occurred.</td>
</tr>
<tr>
<td>PROTOCOL</td>
<td>VT100, VT220, or VT420</td>
</tr>
<tr>
<td>SCREENSIZE</td>
<td>24 or 43</td>
</tr>
<tr>
<td>ECHO</td>
<td>ON or OFF</td>
</tr>
</tbody>
</table>

If a null string is returned, you can use AFR_TELNET_GETLASTERROR to find the exact reason for the AFR_TELNET_QUERY error. Following are the possible error codes:

1. An Invalid parameter was specified on the AFR_TELNET_QUERY.
2. An invalid REXX environment exists.
3. An AFR_SELECT was not previously done for a Telnet session.
4. The AFR_TELNET_QUERY failed to complete.

Remarks

AFR_TELNET_QUERY offers several useful applications to SA IOM REXX scripts:

- A REXX script can periodically check to see if a Telnet host connection is still active. If the session has been lost for some reason, as signified by a "Not connected" status returned from AFR_TELNET_QUERY("STATUS"), then you can issue an AFR_TELNET_OPEN to that host to re-establish the connection.
- A REXX script can determine the VT protocol of a Telnet session and, if necessary, change the protocol (using AFR_TELNET_SET) prior to issuing an AFR_TELNET_OPEN.

Example

In the REXX code fragment below, an AFR_TELNET_QUERY("STATUS") is issued for the currently selected Telnet session called Telnet2. If the query returns "Not connected", then an AFR_TELNET_OPEN is issued to a host called "SystemA", which uses the default Telnet port of 23. Another query is then done to obtain the new status of the Telnet2 session.

```rexx
/* REXX */
if AFR_TELNET_QUERY("STATUS") = "Not connected" then do
    ret = AFR_TELNET_OPEN("SystemA", 23)
    if ret <> 0 then do
        say "AFR_TELNET_OPEN error to SystemA, rc:" || ret
```
AFR_TELNET_QUERY

```r
exit
end
say "current status of Telnet2:" AFR_TELNET_QUERY("STATUS")
end
```

See also

- "AFR_SELECT" on page 87
- "AFR_TELNET_GETLASTERROR" on page 109
- "AFR_TELNET_OPEN" on page 109
- "AFR_TELNET_SET"

**AFR_TELNET_SET**

Use this function to change the protocol and local echo setting of a Telnet session. The session must have been previously made available through an AFR_SELECT.

**Format**

```
AFR_TELNET_SET(keyword, value)
```

- **keyword**
  - PROTOCOL or ECHO
- **value**
  - If the keyword is PROTOCOL, the value parameter can be VT100, VT220, or VT420.
  - If the keyword is ECHO, the value parameter can be ON or OFF.

**Returns**

- **0** The requested session value was successfully implemented.
- **1** An invalid parameter was specified on the AFR_TELNET_SET.
- **2** An invalid REXX environment exists.
- **3** An AFR_SELECT was not previously done for a Telnet session.
- **4** The AFR_TELNET_SET failed to complete.
- **5** The selected Telnet session currently has an active connection.
- **6** An invalid keyword/value combination was specified.

**Remarks**

You cannot use AFR_TELNET_SET to change the protocol of a currently active Telnet session. Protocol changes must be performed before a session connection has been established or an RC = 5 error code will be returned. However, you can change the local echo setting either before or after a connection has occurred.

**Example**

In the following example, a Telnet hostname and port number are passed as parameters to a REXX script. The script selects a configured Telnet host session called Telnet1. Because the configured protocol of Telnet1 is VT220, the script uses AFR_TELNET_SET to change the protocol to VT100 prior to issuing AFR_TELNET_OPEN. The Telnet open is then attempted using the supplied hostname and port number. If the open succeeds, another AFR_TELNET_SET is issued to make sure that echoing is on for the session.
/* REXX */
parse arg hostname portnumber
ret = AFR_SELECT(Telnet1)
if ret <> 0 then
  do
    say "AFR_SELECT error:" || ret
    exit
  end

/*Change the session protocol to VT100 prior to opening the connection*/
ret = AFR_TELNET_SET("PROTOCOL", "VT100")
if ret = 0 then
  do
    ret = AFR_TELNET_OPEN(hostname, portnumber)
    if ret <> 0 then
      do
        say "AFR_TELNET_OPEN error:" || ret
        exit
      end
  end

/* Make sure that local echoing is turned on for this session */
ret = AFR_TELNET_SET("ECHO", "ON")
say "Current local echo value of Telnet1:" AFR_TELNET_QUERY("ECHO")

See also
"AFR_SELECT" on page 87
"AFR_TELNET_OPEN" on page 109

AFR_TTS_FILE
Use this function to speak the text contained in a file to a user dialed into a Dialogic voice card.

Format

AFR_TTS_FILE(channel, file[, modifier] )

channel The voice channel to use. You can enable a card for two or four channels.

file The name of the file containing text to be spoken. If the file name includes
      embedded spaces or a fully qualified path, it should be enclosed in single or
double quotes.

modifier An optional literal string that modifies certain spoken qualities such as volume,
speech rate, speech pitch, and voice. The string takes the form '5,150,200,9'.

The first parameter is volume, ranging from 0 to 16, default 5.

The second parameter is speech rate, ranging from 50 to 250, default 150.

The third parameter is speech pitch, ranging from 50 to 400, default 85.

The fourth parameter is voice, ranging from 0 to 9, default 0. Voice is a combination
of the second and third parameters, which means the voice value overrides any values specified for the speech rate and pitch.

If you set only the first and fourth parameters, the parameter string would appear as '9,,9'.

If you set only the second and third parameters, the string would appear as ',100,100'.

Functions 113
AFR_TTS_FILE

Returns

This function returns the following codes:

- **0**  Text file spoken successfully.
- **-5**  Error on TTS open. Text-to-speech not installed or enabled.
- **-6**  Error speaking a text-to-speech file.
- **-7**  Error speaking a text-to-speech string.
- **-14**  Error allocating voice control block.
- **-15**  Invalid line requested.
- **-16**  Error allocating the channel.
- **-17**  Error setting the hook state.
- **-19**  Error clearing the digit buffer.
- **-20**  Playback was ended when a key was pressed on the phone keypad.
- **-21**  Noise on playback, caller may have hung up.
- **-22**  Line active on another thread.
- **-23**  Phone was hung up during playback.
- **-77**  The file *TextTalk.DLL* could not be found.
- **-99**  Error loading the voice DLL.

Remarks

**Note:** This function can be used only if a supported voice adapter is installed in the SA IOM server. For information about specific voice adapters supported by this product, see the *System Automation for Integrated Operations Management User's Guide.*

The AFR_TTS_FILE function allows a file containing text to be spoken to a user dialed into a supported Dialogic voice card.

Example

In this example, the file *test.txt* is spoken using default values.

```
AFR_TTS_FILE(1, test.txt)
```

In this example, the file *voice.txt* is spoken at a faster rate and higher pitch.

```
AFR_TTS_FILE(2, voice.txt, ',100,200')
```

See also

- "AFR_VOICEINIT" on page 126
- "AFR_VOICEHANGUP" on page 125

AFR_TTS_STRING

Use this function to speak the text contained in a string to a user dialed into a Dialogic voice card.

Format

```
AFR_TTS_STRING(channel, string[, modifier] )
```
AFR_TTS_STRING

channel The voice channel to use. You can enable a card for two or four channels.

string The string containing text to be spoken. It should be enclosed in single or double quotes.

modifier An optional literal string that modifies certain spoken qualities such as volume, speech rate, speech pitch, and voice. The string takes the form '5,150,200,9'.

The first parameter is volume, ranging from 0 to 16, default 5.

The second parameter is speech rate, ranging from 50 to 250, default 150.

The third parameter is speech pitch, ranging from 50 to 400, default 85.

The fourth parameter is voice, ranging from 0 to 9, default 0. Voice is a combination of the second and third parameters, which means the voice value overrides any values specified for the speech rate and pitch.

If you set only the first and fourth parameters, the parameter string would appear as '9,,,9'.

If you set only the second and third parameters, the parameter string would appear as '100,100'.

Returns

This function returns the following codes:

0 Text string spoken successfully.
-5 Error on TTS open. Text-to-speech not installed or enabled.
-7 Error speaking a text-to-speech string.
-14 Error allocating voice control block.
-15 Invalid line requested.
-16 Error allocating the channel.
-17 Error setting the hook state.
-19 Error clearing the digit buffer.
-20 Playback ended when a key was pressed on the phone keypad.
-21 Noise on playback, caller may have hung up.
-22 Line active on another thread.
-23 Phone was hung up during playback.
-77 The file TextTalk.DLL could not be found.
-99 Error loading the voice DLL.

Remarks

Note: This function can be used only if a supported voice adapter is installed in the SA IOM server. For information about specific voice adapters supported by this product, see the System Automation for Integrated Operations Management User’s Guide.

The text string (AFR_TTS_STRING) can be spoken to a user dialed into a supported Dialogic voice card.
AFR_TTS_STRING

Example

In this example, the string is spoken using default values.
ret = AFR_TTS_STRING(1,'there is no time like the present')

In this example, the string is spoken at a faster rate and higher pitch.
ret = AFR_TTS_STRING(2,'Are we there yet?',',100,200')

See also

“AFR_VOICEINIT” on page 126
“AFR_VOICEHANGUP” on page 125

AFR_UA_WRITE

Use this function to write a record to saiom2ua.txt, which is monitored by the IBM Tivoli Universal Agent.

Format

AFR_UA_WRITE([type, severity, ]text)

If you specify text only, this implies a type of "01" and severity of "N"

| type   | The message type. It is user-defined. |
| severity | The severity of the message. It is user-defined. |
| text    | The free-form content of the record that is to be trapped by Universal Agent. |

Returns

0 The record was successfully written.

Remarks

See the sample meta file, config\saiom2ua.mdl, for the Universal Agent configuration.

Example

rc = AFR_UA_WRITE("Message to be trapped by UA")

AFR_UNLOCK

Use this function to free access to a previously specified emulation session or serial port configured for script use.

Format

AFR_UNLOCK(resource|ALL)

| resource | The name of the SA IOM emulation session or resource to be freed. It must match the resource name specified in the preceding AFR_LOCK function call. The name is case sensitive and may include imbedded blanks. Enclose this parameter in quotes. |
| ALL      | This parameter specifies that all resources held by the program are freed. |
**AFR_UNLOCK**

**Returns**

- **0**: The resource has been freed.
- **1**: The resource has not been freed (the resource is owned by another program).
- **4**: The resource is not defined.

**Remarks**

Use the ALL parameter if the program that you are using has serialized a number of resources.

Remember to free any resource held exclusively after a program has executed. Failure to free a session can prevent other programs that need exclusive access to that session from executing successfully.

**Example**

```plaintext
AFR_UNLOCK("ALL")
```

**See also**

"AFR_LOCK" on page 52

---

**AFR_USER_CHANGEPSW**

Use this function to change the specified userid's password when the old password is known.

**Format**

```plaintext
AFR_USER_CHANGEPSW(userid, oldpsw, newpsw, type)
```

- **userid**: The user-ID as defined in the SA IOM configuration.
  - If changing a Windows password, the ID must also be defined in the Windows user list.
- **oldpsw**: The current password that is associated with this user ID.
- **newpsw**: The new password that will be associated with this user ID.
- **type**: Specify one of the following password change types.
  - **DEFAULT**: Use SA IOM's current method of password authentication.
  - **AFR**: Make the password change in the SA IOM configuration.
  - **WINDOWS**: Make the password change in the Windows user list.

**Returns**

- **0**: OK
- **1**: User not found
- **2**: Incorrect password
- **3**: Invalid new password
- **4**: Invalid type
AFR_USER_CHANGEPSW

Cannot change the password for "Administrator"

Cannot save rpuser.dat

Rpuser.dat locked by other user

RpUser file busy. Locked by other user

The AFR_USER REXX functions are not enabled. See the ALLOW_AFR_USER_REXX keyword in the server profile file.

Character convert error

WSA startup error

Get hostname error

Wide character conversion error

WSA cleanup error

Windows access error

Incorrect user or password

Invalid computer name

Cannot change password: Not primary

Invalid password format

The new password does not meet the format requirements

Cannot read password server's domain data

Not allowed on last administrator

Not allowed on groups

Other Windows error

Remarks

The keyword ALLOW_AFR_USER_REXX = YES must be defined in the server's RpSvrPrf.txt file.

Example

RET = AFR_USER_CHANGEPSW("Jones","Chuck","Davey","DEFAULT")

See also

AFR_LOG on page 53
AFR_OP_ERROR on page 66

AFR_USER_LOGONTYPE

Use this function to retrieve or to set the password authentication method SA IOM uses to validate logons to the server. Using this function requires Administrators authority.

Format

AFR_USER_LOGONTYPE(type)
AFR_USER_LOGONTYPE

Specify one of the following password authentication settings:

**DEFAULT**
Return SA IOM's current password authentication setting.

**AFR**
Set passwords to be authenticated by SA IOM only.

**WINDOWS**
Set passwords to be authenticated by the Windows user list.

**Returns**
The AFR_USER_LOGONTYPE function returns "AFR" or "WINDOWS" if the *type* is "DEFAULT" and there are no errors. Otherwise it returns one of the following:

0  The type is "AFR" or "WINDOWS" and the default is changed, and there are no errors.
4  Invalid type.
7  Script owner not authorized to perform the function.
8  Script owner not found in the SA IOM configuration.
10 The AFR_USER REXX functions are not enabled. See the ALLOW_AFR_USER_REXX keyword in the server profile file.

**Remarks**
The keyword ALLOW_AFR_USER_REXX = YES must be defined in the server's RpSvrPrf.txt file. The REXX owner must have Administrators authority.

**Example**

RET = AFR_USER_LOGONTYPE("AFR")

**See also**

|AFR_USER_CHANGEPSW| on page 117
|AFR_USER_SETPSW|

AFR_USER_SETPSW

Use this function to set the specified userid's password. Using this function requires Administrators authority.

**Format**

AFR_USER_SETPSW(userid, newpsw, type)

**userid**
The user-ID as defined in the SA IOM configuration.

If changing a Windows password, the ID must also be defined in the Windows user list.

**newpsw**
Specify the new password.
AFR_USER_SETPSW

**type**

Specify one of the following password authentication types.

- **DEFAULT**
  Use SA IOM's current authentication setting.

- **AFR**
  Make the password change in the SA IOM configuration.

- **WINDOWS**
  Make the password change in Windows user list.

**Returns**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK</td>
</tr>
<tr>
<td>1</td>
<td>User not found</td>
</tr>
<tr>
<td>3</td>
<td>Invalid new password</td>
</tr>
<tr>
<td>4</td>
<td>Invalid type</td>
</tr>
<tr>
<td>5</td>
<td>Cannot change the password for &quot;Administrator&quot;</td>
</tr>
<tr>
<td>6</td>
<td>Cannot save rpuser.dat</td>
</tr>
<tr>
<td>7</td>
<td>Script owner not authorized to perform the function</td>
</tr>
<tr>
<td>8</td>
<td>Owner not found in the SA IOM configuration</td>
</tr>
<tr>
<td>9</td>
<td>RpUser file busy. Locked by other user</td>
</tr>
<tr>
<td>10</td>
<td>The AFR_USER REXX functions are not enabled. See the ALLOW_AFR_USER_REXX keyword in the server profile file.</td>
</tr>
<tr>
<td>11</td>
<td>Character convert error</td>
</tr>
<tr>
<td>12</td>
<td>WSA startup error</td>
</tr>
<tr>
<td>13</td>
<td>Get hostname error</td>
</tr>
<tr>
<td>14</td>
<td>Wide character conversion error</td>
</tr>
<tr>
<td>15</td>
<td>WSA cleanup error</td>
</tr>
<tr>
<td>16</td>
<td>Windows access error</td>
</tr>
<tr>
<td>17</td>
<td>Incorrect user or password</td>
</tr>
<tr>
<td>18</td>
<td>Invalid computer name</td>
</tr>
<tr>
<td>19</td>
<td>Cannot change password: Not primary</td>
</tr>
<tr>
<td>20</td>
<td>Invalid password format</td>
</tr>
<tr>
<td>21</td>
<td>The new password does not meet the format requirements</td>
</tr>
<tr>
<td>22</td>
<td>Cannot read password server's domain data</td>
</tr>
<tr>
<td>30</td>
<td>Other Windows error</td>
</tr>
</tbody>
</table>

**Remarks**

The keyword ALLOW_AFR_USER_REXX = YES must be defined in the server's RpSvrPrf.txt file. The REXX owner must have Administrators authority.

**Example**

```
RET = Afr_User_SetPsw("Jones","secret","DEFAULT")
```
AFR_VOICEANSWER

Use this function to wait for an incoming call.

Format

AFR_VOICEANSWER(line,timeout,nrings)

line This parameter specifies the voice adapter line number. Up to 32 lines can be made available depending on the number of voice adapters installed.

timeout This parameter specifies the time (in seconds) to wait for an incoming call. The maximum value for this parameter is $2^{31}-1$ millisecond.

A value of -1 causes an indefinite wait.
nrings This parameter specifies which ring to answer on.

Returns

This function returns the following codes:

0 Successful answer.
-1 Unspecified error. This usually is an internal error to Dialogic. The only way to recover is to hang up the line by using AFR_VOICEHANGUP and then begin again with AFR_VOICEINIT.
-2 Error releasing previously requested line.
-3 Error requesting ringing line.
-4 Time out with no phone ringing.
-5 Error answering a call.
-13 Error opening the device driver.
-14 Error allocating the voice control block.
-15 Invalid line requested.
-16 Error allocating the channel.
-17 Error setting the hook state.
-22 Line active on another thread. The line used in the AFR_VOICE function call appears to be active inside another thread. Make sure the line number requested on the AFR_VOICE function call is correct and is not being used by another REXX script.
-99 Error loading the voice DLL.

Remarks

Note: This function can be used only if a supported voice adapter is installed in the SA IOM server. For information about specific voice adapters supported by this product, see the System Automation for Integrated Operations Management User's Guide.

If the call is completed, this function answers the phone.
**AFR_VOICEANSWER**

**Example**

```plaintext
RET = AFR_VOICEANSWER(1, 60, 2)
```

**AFR_VOICEBEEP**

Use this function to generate a number of beeps.

**Format**

```plaintext
AFR_VOICEBEEP(line, nbeeps, tone)
```

- **line**: This parameter specifies the voice adapter line number. Up to 32 lines can be made available depending on the number of voice adapters installed.
- **nbeeps**: This parameter specifies the number of beeps which can be used to prompt the user.
- **tone**: This parameter specifies the tone of the beep:

<table>
<thead>
<tr>
<th>Tone</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>250 HZ</td>
</tr>
<tr>
<td>1</td>
<td>500 HZ</td>
</tr>
<tr>
<td>2</td>
<td>750 HZ</td>
</tr>
<tr>
<td>3</td>
<td>1000 HZ</td>
</tr>
<tr>
<td>4</td>
<td>1250 HZ</td>
</tr>
<tr>
<td>5</td>
<td>1500 HZ</td>
</tr>
<tr>
<td>6</td>
<td>1750 HZ</td>
</tr>
<tr>
<td>7</td>
<td>2000 HZ</td>
</tr>
</tbody>
</table>

**Returns**

This function returns the following codes:

- **0**: The beep was successfully generated.
- **-1**: Error setting the tone of the beep.
- **-2**: Error generating the beep.
- **-13**: Error opening the device driver.
- **-14**: Error allocating the voice control block.
- **-15**: Invalid line requested.
- **-16**: Error allocating the channel.
- **-17**: Error setting the hook state.
- **-22**: Line active on another thread. The line used in the AFR_VOICE function call appears to be active inside another thread. Make sure the line number requested on the AFR_VOICE function call is correct and is not being used by another REXX script.
- **-99**: Error loading the voice DLL.
Remarks

**Note:** This function can be used only if a supported voice adapter is installed in the SA IOM server. For information about specific voice adapters supported by this product, see the *System Automation for Integrated Operations Management User's Guide.*

Example

```
RET = AFR_VOICEBEEP(1,2,3)
```

**AFR_VOICEDIAL**

Use this function to dial a number.

**Format**

```
AFR_VOICEDIAL(line,dialstring,rings)
```

- **line**  This parameter specifies the voice adapter line number. Up to 32 lines can be made available depending on the number of voice adapters installed.
- **dialstring**  This parameter specifies the number to be dialed. Enclose the number in quotes. It must evaluate to a literal string of 64 characters or less. Table 6 on page 124 lists the characters you can use in an AFR_VOICEDIAL `dialstring`.
- **rings**  This parameter specifies the number of rings to wait before returning a code.

**Returns**

This function returns the following codes:

- 0  Voice detected.
- -1  Unspecified error. Thus usually is an internal error to Dialogic. The only way to recover is to hang up the line by using AFR_VOICEHANGUP and then begin again with AFR_VOICEINIT.
- -2  Busy signal detected.
- -3  Fast-busy signal detected.
- -4  Special-information tone detected (operator intercept).
- -5  Time out with no answer.
- -6  Error dialing.
- -7  No ringback.
- -9  Call progress analysis error.
- -10  Dialing was interrupted.
- -11  Fax tone detected.
- -12  No dialtone.
- -13  Error opening the device driver.
- -14  Error allocating the voice control block.
- -15  Invalid line requested.
- -16  Error allocating the channel.
- -17  Error setting the hook state.
AFR_VOICEDIAL

-22 Line active on another thread. The line used in the AFR_VOICE function call appears to be active inside another thread. Make sure the line number requested on the AFR_VOICE function call is correct and is not being used by another REXX script.

-24 An answering machine answered the phone (Extended return code). Use AFR_VOICESPEAK to leave a message, then AFR_VOICEHANGUP to hangup the phone. If you want to force voice-only answer mode when a call is dialed, place a "V" in the phone number. This will cause REXX_Voice_Version 2 to function like REXX_Voice_Version 1 and disable the answering machine detection.

-25 An answering machine answered the phone, but the end of the greeting could not be found in 30 seconds (Extended return code). You can wait longer and then try to leave a message, but will probably be unable to do so. Try hanging up the phone, re-initializing the line, and dialing again.

-99 Error loading the voice DLL.

Remarks

Note: This function can be used only if a supported voice adapter is installed in the SA IOM server. For information about specific voice adapters supported by this product, see the System Automation for Integrated Operations Management User's Guide.

You can also use this function to emulate touch tones by calling this function with the rings parameter specified as zero (0) when the line has successfully connected.

Example

RET = AFR_VOICEDIAL(1,'9,1-800-555-1212',20)

Table 6. AFR_VOICEDIAL Dialstring characters

<table>
<thead>
<tr>
<th>Char</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-9</td>
<td>Touch-tone digits to be generated</td>
</tr>
<tr>
<td>#</td>
<td>Touch-tone pound to be generated</td>
</tr>
<tr>
<td>*</td>
<td>Touch-tone star to be generated</td>
</tr>
<tr>
<td>()</td>
<td>No effect; used only to make dialstring more readable</td>
</tr>
<tr>
<td>,</td>
<td>2-second delay</td>
</tr>
<tr>
<td>!</td>
<td>Flash hook</td>
</tr>
<tr>
<td>V</td>
<td>Voice-only answer mode. V should be the first character of the dialstring. The call is assumed to be answered by a real person. Answering machine detect is disabled if V is coded in the dialstring.</td>
</tr>
</tbody>
</table>

AFR_VOICEGETTONES

Use this function to retrieve the touch-tones generated by a caller.

Format

AFR_VOICEGETTONES(line,maxtones,endtones,timeout)

line This parameter specifies the voice adapter line number. Up to 32 lines can be made available depending on the number of voice adapters installed.
AFR_VOICEGETTONES

maxtones  This parameter specifies the maximum number of tones to receive before returning the string.
endtones  This parameter specifies a list of tones that indicate the end of the string. The digits 0-9, the pound sign (#), and the star sign (*) are all valid endtones. Enclose the characters in quotes.
timeout   This parameter specifies the number of seconds to wait for a touch-tone before timing out and returning the string. The maximum value for this parameter is 2^{31} - 1 milliseconds.

Returns

Upon successful completion, this function returns a literal string consisting of touch-tones generated by the person to whom the voice adapter is speaking.

A null string is returned when a valid endtone is received, or when a time out occurs, or when the number of tones received equals the maxtones parameter. See more about null strings under Remarks.

Upon error, possible returns are:

-13        Error opening the device driver.
-14        Error allocating the voice control block.
-15        Invalid line requested.
-16        Error allocating the channel.
-22        Line active on another thread. The line used in the AFR_VOICE function call appears to be active inside another thread. Make sure the line number requested on the AFR_VOICE function call is correct and is not being used by another REXX script.
-99        Error loading the voice DLL.

Remarks

Note: This function can be used only if a supported voice adapter is installed in the SA IOM server. For information about specific voice adapters supported by this product, see the System Automation for Integrated Operations Management User’s Guide.

If this function returns a null string, then either an error occurred, or the function timed out before the person called by the adapter could respond. To find out the reason for the failure, refer to the topic “Voice Diagnostic and Debugging Facilities” in the chapter on voice control in the System Automation for Integrated Operations Management User’s Guide.

Example

STR = AFR_VOICEGETTONES(1,15,'#',10)

AFR_VOICEHANGUP

Use this function to hang up the phone and deallocate the voice adapter.

Format

AFR_VOICEHANGUP(line)
AFR_VOICEHANGUP

**line**
This parameter specifies the voice adapter line number. Up to 32 lines can be made available depending on the number of voice adapters installed.

**Returns**

This function returns the following codes:

- **0**  
  Successful hangup.
- **-2**  
  Error in hanging up.
- **-3**  
  Error releasing line.
- **-4**  
  Error releasing channel.
- **-13**  
  Error opening the device driver.
- **-14**  
  Error allocating the voice control block.
- **-15**  
  Invalid line requested.
- **-16**  
  Error allocating the channel.
- **-17**  
  Error setting the hook state.
- **-22**  
  Line active on another thread. The line used in the AFR_VOICE function call appears to be active inside another thread. Make sure the line number requested on the AFR_VOICE function call is correct and is not being used by another REXX script.
- **-99**  
  Error loading the voice DLL.

**Remarks**

**Note:** This function can be used only if a supported voice adapter is installed in the SA IOM server. For information about specific voice adapters supported by this product, see the *System Automation for Integrated Operations Management User's Guide*.

Calling this function hangs up a previously-connected line.

If you are using Dialogic adapter(s), you must include this function in a REXXX program to release the line. However, you can call any other voice function without hanging up the previously-used line.

**Example**

```
RET = AFR_VOICEHANGUP()
```

AFR_VOICEINIT

Use this function to test if the server’s voice adapter is accessible by this application. It verifies that the voice adapter is operational by checking that the device drivers are installed and running. If successful, you can then safely call other voice functions.

**Warning**

It is recommended that you call this function before calling other voice functions if you are unsure that your voice adapter device drivers are installed and running. Any return code other than zero indicates that the adapter is not operational. Under such a condition, use of other REXX voice functions may cause system failure.
AFR_VOICEINIT

Format

AFR_VOICEINIT(line)

line This parameter specifies the voice adapter line number. Up to 32 lines can be made available depending on the number of voice adapters installed.

Returns

This function returns the following codes:

0 Successful initialization.
-13 Error opening the device driver.
-14 Error allocating the voice control block.
-15 Invalid line requested.
-16 Error allocating the channel.
-17 Error setting the hook state.
-22 Line active on another thread. The line used in the AFR_VOICE function call appears to be active inside another thread. Make sure the line number requested on the AFR_VOICE function call is correct and is not being used by another REXX script.
-99 Error loading the voice DLL.

Remarks

Note: This function can be used only if a supported voice adapter is installed in the SA IOM server. For information about specific voice adapters supported by this product, see the System Automation for Integrated Operations Management User’s Guide.

Example

RET = AFR_VOICEINIT(1)

See also

AFR_VOICEHANGUP on page 125

AFR_VOICERECORD

Use this function to record a voice message into a server PC file. At the start of this function, a beep prompts the user to speak. The function stops recording if the user stops talking or presses any telephone key.

Format

AFR_VOICERECORD(line, filename, length, mute[, speed])

line This parameter specifies the voice adapter line number. Up to 32 lines can be made available depending on the number of voice adapters installed. This parameter is optional unless you provide the speed parameter.
filename This parameter specifies the server PC file that will contain this message.
length This parameter specifies the length of the message in seconds. The minimum specifiable length is 5 seconds.
**AFR_VOICERECORD**

**mute**
This parameter specifies whether to mute (1) or not mute (0) the outside line when recording a message. This is necessary because when you pick up the local receiver, you are connected to the outside line and hear a dial tone. Muting the outside line prevents the dial tone from being recorded with your message. If you are using Dialogic adapters, specify mute (1).

**speed**
This parameter specifies one of the six supported speeds for record. This parameter is optional. The default speed is ADPCM24. If speed is specified, the line parameter is required. Record and playback speeds must match or results will be unpredictable. Record speeds are:

- **ADPCM24**
  Records at 24 kbps using the default mode ADPCM. The sample rate is 6000 samples per second using 4 bits.

- **ADPCM32**
  Records at 32 kbps using the default mode ADPCM. The sample rate is 8000 samples per second using 4 bits.

- **PCMM48**
  Records at 48 kbps using MuLaw encoding. The sample rate is 6000 samples per second using 8 bits. This type of encoding is only available in North America and Japan. The Dialogic D/21D and D/41D support this type of encoding.

- **PCMM64**
  Records at 64 kbps using MuLaw encoding. The sample rate is 8000 samples per second using 8 bits. This type of encoding is only available in North America and Japan. The Dialogic D/21D and D/41D support this type of encoding.

- **PCMA48**
  Records at 48 kbps using ALaw encoding. The sample rate is 6000 samples per second using 8 bits. This type of encoding is used everywhere except North America and Japan. The Dialogic D/41ESC supports this type of encoding.

- **PCMA64**
  Records at 64 kbps using ALaw encoding. The sample rate is 8000 samples per second using 8 bits. This type of encoding is used everywhere except North America and Japan. The Dialogic D/41ESC supports this type of encoding.

**Returns**

This function returns the following codes:

- **0**  
  Successful record.

- **-2**  
  Unable to open file.

- **-3**  
  Error recording message.

- **-13**  
  Error opening the device driver.

- **-14**  
  Error allocating the voice control block.

- **-15**  
  Invalid line requested.

- **-16**  
  Error allocating the channel.

- **-17**  
  Error setting the hook state.

- **-18**  
  Record time expired (Extended return code). After AFR_VOICERECORD, the time limit expired to record your voice file. This may indicate the line was hung up sometime during the record or before the record began.

- **-19**  
  Error clearing Digit buffer. While attempting to clear the digit buffer in the firmware, the Dialogic API returned an error. This usually is an internal error to Dialogic. The only way to recover is to hang up the line by using AFR_VOICEHANGUP and then begin again with AFR_VOICEINIT.
AFR_VOICERECORD

-22 Line active on another thread. The line used in the AFR_VOICE function call appears to be active inside another thread. Make sure the line number requested on the AFR_VOICE function call is correct and is not being used by another REXX script.

-99 Error loading the voice DLL.

Note: When recording messages, be aware that every second requires 3K of disk space in your voice file. Forty 10-second messages would require 1.2 megabytes of disk space.

Remarks

Note: This function can be used only if a supported voice adapter is installed in the SA IOM server. For information about specific voice adapters supported by this product, see the System Automation for Integrated Operations Management User’s Guide.

To record a message of less than 5 seconds, press any touch tone at the end of the recording to halt it immediately.

Example

RET = AFR_VOICERECORD(1,'voxfile1.vox',60,1,ADPCM24)

AFR_VOICESPEAK

Use this function to play back a pre-recorded voice message.

Format

AFR_VOICESPEAK(line,filename,mute[,speed])

line This parameter specifies the voice adapter line number. Up to 32 lines can be made available depending on the number of voice adapters installed. This parameter is optional unless you provide the speed parameter.

filename This parameter specifies the server PC voice file and must contain both the filename and extension (for example, voxfile1.vox), enclosed in quotes.

mute This parameter specifies whether to mute (1) or not mute (0) the outside line. If the line is muted, no dial tone is heard. If it is not muted, the speech can be heard over the telephone line. If you are using Dialogic adapters, specify mute (1).
speed

This parameter specifies one of the six supported speeds for record. This parameter is optional. The default speed is ADPCM24. If speed is specified, the line parameter is required. Record and playback speeds must match or results will be unpredictable. Record speeds are:

- **ADPCM24** Records at 24 kbps using the default mode ADPCM. The sample rate is 6000 samples per second using 4 bits.
- **ADPCM32** Records at 32 kbps using the default mode ADPCM. The sample rate is 8000 samples per second using 4 bits.
- **PCMM48** Records at 48 kbps using MuLaw encoding. The sample rate is 6000 samples per second using 8 bits. This type of encoding is only available in North America and Japan. The Dialogic D/21D and D/41D support this type of encoding.
- **PCMM64** Records at 64 kbps using MuLaw encoding. The sample rate is 8000 samples per second using 8 bits. This type of encoding is only available in North America and Japan. The Dialogic D/21D and D/41D support this type of encoding.
- **PCMA48** Records at 48 kbps using ALaw encoding. The sample rate is 6000 samples per second using 8 bits. This type of encoding is used everywhere except North America and Japan. The Dialogic D/41ESC supports this type of encoding.
- **PCMA64** Records at 64 kbps using ALaw encoding. The sample rate is 8000 samples per second using 8 bits. This type of encoding is used everywhere except North America and Japan. The Dialogic D/41ESC supports this type of encoding.

**Returns**

This function returns the following codes:

- **0** Successful play back.
- **-2** Unable to access file.
- **-3** Error playing message.
- **-13** Error opening the device driver.
- **-14** Error allocating the voice control block.
- **-15** Invalid line requested.
- **-16** Error allocating the channel.
- **-17** Error setting the hook state.
- **-19** Error clearing Digit buffer. While attempting to clear the digit buffer in the firmware, the Dialogic API returned an error. This usually is an internal error to Dialogic. The only way to recover is to hang up the line by using AFR_VOICEHANGUP and then begin again with AFR_VOICEINIT.
- **-20** Playback ended with keypress (Extended return code). During playback of a message, a key was pressed on the phones key pad. You can use AFR_VOICEGETTONES to retrieve the key that was pressed.
- **-21** Noise on playback (Extended return code). Noise was detected during file playback (AFR_VOICESPEAK). This usually indicates that the phone was hung up during the playback, but can also indicate a very noisy line.
AFR_VOICESPEAK

-22 Line active on another thread. The line used in the AFR_VOICE function call appears to be active inside another thread. Make sure the line number requested on the AFR_VOICE function call is correct and is not being used by another REXX script.

-23 Phone was hung up during playback (Extended return code). The call hung up the phone during the playback of a voice message. Call AFR_VOICEHANGUP and then begin again with AFR_VOICEINIT.

-99 Error loading the voice DLL.

Remarks

Note: This function can be used only if a supported voice adapter is installed in the SA IOM server. For information about specific voice adapters supported by this product, see the System Automation for Integrated Operations Management User’s Guide.

Example

RET = AFR_VOICESPEAK(1,'voxfile1.vox',1,ADPCM24)

AFR_WHOAMI

Use this function to determine the SA IOM script ID.

Format

AFR_WHOAMI()

Returns

This function returns the SA IOM script ID number assigned to this REXX program.

Example

RET = AFR_WHOAMI()

See also

“AFR_STOP_PROGRAM” on page 107

AFR_WHOIS

Use this function to return the line containing the script filename, parameters, time, date, and status of the specified REXX program. The REXX program is specified by SA IOM script ID number.

Format

AFR_WHOIS(script_id)

script_id This parameter is the SA IOM script ID number assigned to the REXX program. The script ID is displayed after the REXX script name on the Scripts control panel and also on the title bar of an opened script session window.

Returns

This function returns the status line for the target script ID.
AFR_WHOIS

This function returns a null string if the specified SA IOM script ID number has not yet run a program. This function may also return the standard REXX error numbers 5, 26, and 40.

Example

```rexx
Do i=1 to 96
   d=afr_whois(i)
   if d<>'' then
      say i d
   end
```
Appendix. Notify return codes

One of the following return codes can be expected from the notify REXX functions if there is an error.

Table 7. Notify return codes

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NRC_OK&lt;br&gt;Successful completion.</td>
</tr>
<tr>
<td>-1</td>
<td>NRC_ODBC_ERROR&lt;br&gt;There is an ODBC issued error. Refer to the documentation provided by your ODBC application.</td>
</tr>
<tr>
<td>-2</td>
<td>NRC_NOTIFY_NOT_ENABLED&lt;br&gt;The alert escalation feature is not enabled.</td>
</tr>
<tr>
<td>-3</td>
<td>NRC_NOTIFY_UNINITIALIZED&lt;br&gt;Afr_Notify_Init has not successfully competed for the current script.</td>
</tr>
<tr>
<td>-4</td>
<td>NRC_ALREADY_INITIALIZED&lt;br&gt;Afr_Notify_Init can only be executed once.</td>
</tr>
<tr>
<td>-5</td>
<td>NRC_INVALID_USER&lt;br&gt;Incorrect user specified.</td>
</tr>
<tr>
<td>-6</td>
<td>NRC_INVALID_PSW&lt;br&gt;Incorrect password specified.</td>
</tr>
<tr>
<td>-7</td>
<td>NRC_ESCALATION_NOT_ACTIVE&lt;br&gt;The escalation is not active.</td>
</tr>
<tr>
<td>-8</td>
<td>NRC_DB_CONNECT_ERROR&lt;br&gt;Error connecting to the database server.</td>
</tr>
<tr>
<td>-9</td>
<td>NRC_DB_OPEN_ERROR&lt;br&gt;Open error accessing the database.</td>
</tr>
<tr>
<td>-10</td>
<td>NRC_DB_GETNEXT_ERROR&lt;br&gt;Database GetNext function failed.</td>
</tr>
<tr>
<td>-11</td>
<td>NRC_UPDATE_ERROR&lt;br&gt;Database Update function failed.</td>
</tr>
<tr>
<td>-12</td>
<td>NRC_REQUERY_ERROR&lt;br&gt;Database Requery function failed.</td>
</tr>
<tr>
<td>-13</td>
<td>NRC_MOVE_LAST_ERROR&lt;br&gt;Database MoveLast function failed.</td>
</tr>
<tr>
<td>-16</td>
<td>NRC_DB_TOO_OLD&lt;br&gt;The database version is too old.</td>
</tr>
<tr>
<td>Return</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>-17 NRC_DB_TOO_NEW</td>
<td>The database version is too new.</td>
</tr>
<tr>
<td>-18 NRC_INSERT_ERROR</td>
<td>Database Insert function error.</td>
</tr>
<tr>
<td>-19 NRC_DB_CANNOT_APPEND</td>
<td>Database Append function failed.</td>
</tr>
<tr>
<td>-20 NRC_FUNCTION_NOT_INIT</td>
<td>Internal database function not initialized.</td>
</tr>
<tr>
<td>-31 NRC_EOF</td>
<td>The database encountered an end of file.</td>
</tr>
<tr>
<td>-32 NRC_CLOSED</td>
<td>The database is closed.</td>
</tr>
<tr>
<td>-33 NRC_EMPTY_EMAIL</td>
<td>Encountered an empty email address.</td>
</tr>
<tr>
<td>-34 NRC_INVALID_EMAILTYPE</td>
<td>Encountered an invalid email type.</td>
</tr>
<tr>
<td>-35 NRC_CANT_START_ESCALATION</td>
<td>The escalation process could not be started.</td>
</tr>
<tr>
<td>-36 NRC_INVALID_PARAMETER</td>
<td>Encountered an invalid parameter.</td>
</tr>
<tr>
<td>-37 NRC_INVALID_ESCID</td>
<td>Encountered an invalid escalation ID. Be aware that the escalation ID is case-sensitive.</td>
</tr>
<tr>
<td>-38 NRC_INVALID_STATE_FOR_ACK</td>
<td>The alert is no longer in the state where an “accept” acknowledgment is valid. The Alert does not accept acknowledgement.</td>
</tr>
<tr>
<td>-40 NRC_QUEUE_SIZE_EXCEEDED</td>
<td>An internal queue is full.</td>
</tr>
<tr>
<td>-41 NRC_MAX_ACTIVE_ALERTS_EXCEEDED</td>
<td>The number of active alerts exceeds the maximum as set by the server profile.</td>
</tr>
<tr>
<td>-50 NRC_CANT_FIND_USER</td>
<td>Cannot find the user in the database.</td>
</tr>
<tr>
<td>-51 NRC_CANT_FIND_ADMIN</td>
<td>Cannot find the administrator in the database.</td>
</tr>
</tbody>
</table>
Table 7. Notify return codes (continued)

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
</table>
| -52    | NRC_CANNOT_FIND_ESCID  
|        | Cannot find the escalation ID in the database. |
| -53    | NRC_CANNOT_FIND_ESCLVL  
|        | Cannot find the escalation level in the database. |
| -54    | NRC_CANNOT_FIND_ESCMEM  
|        | Cannot find the escmem key in the database. |
| -55    | NRC_CANNOT_FIND_PERSON  
|        | Cannot find the person in the database. |
| -56    | NRC_CANNOT_FIND_NOTIFICATION  
|        | Cannot find the notification in the database. |
| -57    | NRC_CANNOT_FIND_ALERT  
|        | Cannot find the alert in the database. |
| -58    | NRC_CANNOT_FIND_EVENT  
|        | Cannot find the event in the database. |
| -59    | NRC_CANNOT_FIND_RESULT  
|        | Cannot find the result in the database. |
| -62    | NRC_CANNOT_FIND_GROUPNAME  
|        | Cannot find the group name in the database. |
| -63    | NRC_CANNOT_FIND_GROUPMEMBER  
|        | Cannot find the group member in the database. |
| -64    | NRC_CANNOT_FIND_CONFIG_TYPE  
|        | Cannot find the configuration type in the database. |
| -70    | NRC_CANNOT_CREATE_ALERT  
|        | Cannot create an alert entry. |
| -71    | NRC_CANNOT_CREATE_EVENT  
|        | Cannot create an event entry. |
| -72    | NRC_SCHEDULE_INACTIVE  
|        | Current schedule inactive. |
| -73    | NRC_SCHEDULE_ONDUTY  
|        | Current schedule active and on duty. |
| -74    | NRC_SCHEDULE_OFFDUTY  
|        | Current schedule active and off duty. |
| -75    | NRC_UNKNOWN_SCHEDULE_TYPE  
<p>|        | The specified schedule type does not exist. |</p>
<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-76</td>
<td>NRC_SCHEDULE_REG_ERROR</td>
</tr>
<tr>
<td></td>
<td>Registry error for timezone conversion.</td>
</tr>
</tbody>
</table>
This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe on any IBM intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, N.Y. 10504-1785
U.S.A.

For additional information, visit the web at:
http://www.ibm.com/ibm/licensing/contact/

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM web sites are provided for convenience only and do not in any manner serve as an endorsement of those web sites. The materials at those web sites are not part of the materials for this IBM product and use of those web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been
estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

If you are viewing this information in softcopy, the photographs and color illustrations may not appear.

Copyrights

References in this documentation to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program or service is not intended to state or imply that only IBM’s product, program or service may be used. Any functionally equivalent product, program or service that does not infringe any of IBM’s intellectual property rights may be used instead of the IBM product, program or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the user’s responsibility.

No part of this document covered by copyright may be reproduced in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system—without prior written permission of the copyright owner.

Software distributed under the License is distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.277-7103 (October 1988) and FAR 52-227-19 (June 1987).

Trademarks

- IBM, AF/OPERATOR, AF/REMOTE, AIX, DB2, HACMP, NetView, OMEGAMON, Tivoli, Tivoli Enterprise, Tivoli Enterprise Console, WebSphere, and z/OS are trademarks of International Business Machines Corporation in the United States, other countries, or both.
- Microsoft, Windows, Windows Vista, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
- Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
- Linux is a trademark of Linus Torvalds in the United States, other countries, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
• Other company, product, and service names may be trademarks or service marks of others.
Index

A

about this document ix
AFR_ANSI 18
AFR_ASYNC_DELETE 18
AFR_ASYNC_WTOR 19
AFR_AUDIT_LOGSIZE 17
AFR_AUDIT_REOPEN_LOG 17
AFR_BROADCAST 20
AFR_CHECK_TRAP 21
AFR_CHECK_WTOR 22
AFR_CLEAR_TRAP 23
AFR_CLEAR_WTOR 23
AFR_CLS 24
AFR_CNTL_TAPI_START 24
AFR_CNTL_TAPI_STOP 25
AFR_COMBREAK 25
AFR_COMCLOSE 26
AFR_COMCTRL 26
AFR_COMGETBAUD 27
AFR_COMGETBITS 27
AFR_COMGETPARITY 28
AFR_COMGETSTOPBITS 28
AFR_COMGETSTR 29
AFR_COMINPUTBYTES 29
AFR_COMJOURNALSTART 30
AFR_COMJOURNALSTOP 32
AFR_COMOPEN 33
AFR_COMOUTPUTBYTES 34
AFR_COMPUTSTR 34
AFR_COMSETBAUD 35
AFR_COMSETLINEC 36
AFR_COMSTATUS 36
AFR_CONSOLE_TYPE 37
AFR_CURSOR 38
AFR_EMAIL_LOGSIZE 40
AFR_EMAIL_REOPEN_LOG 40
AFR_EMIT_UA_SNMP_TRAP 38
AFR_ERROR 41
AFR_GET_LINE 42
AFR_GET_STATUS 42
AFR_GETENV 43
AFR_GETOWNER 46
AFR_GMT 47
AFR_KBHIT 48
AFR_KEY_ET 49
AFR_LOAD 50
AFR_LOAD_COLOR 50
AFR_LOCK 52
AFR_LOG 53
AFR_LOGSIZE 54
AFR_MC_LASTMSGNUM 54
AFR_MC_LASTMSGTEXT 55
AFR_MC_LASTMSGTIME 55
AFR_MC_LOGSIZE 56
AFR_MC_REOPEN_LOG 56
AFR_MC_SEND 57
AFR_MODEMANWER 58
AFR_MODEDCALL 58
AFR_MODEMDISCONNECT 60
AFR_MODEMSTATUS 60
AFR_NOTIFY 61
AFR_NOTIFY_ACK 62
AFR_NOTIFY_EXIT 64
AFR_NOTIFY_INIT 64
AFR_NOTIFY_LASTERROR 64
AFR_NOTIFY_SET_RESULT 65
AFR_OP_ERROR 66
AFR_PEER_CLOSE 67
AFR_PEER_GETLASTERROR 68
AFR_PEER_OPEN 69
AFR_PEER_QUERY 70
AFR_PEER_RCV 73
AFR_PEER_SEND 75
AFR_PROMPT 79
AFR_PROMPT_USER 80
AFR_REOPEN_LOG 81
AFR_RXQUEUE_CREATE 82
AFR_RXQUEUE_DELETE 83
AFR_RXQUEUE_EXISTS 83
AFR_RXQUEUE_EXTERNAL 84
AFR_RXQUEUE_GET 84
AFR_RXQUEUE_Purge 85
AFR_RXQUEUE_SET 85
AFR_SAY 86
AFR_SCREEN_WAIT 86
AFR_SELECT 87
AFR_SELECTED 88
AFR_SEND 89
AFR_SEND_3270 90
AFR_SEND_ASCII 92
AFR_SEND_EMAIL 92
AFR_SEND_MESSAGE 94
AFR_SEND_RPAGE 95
AFR_SEND_SCAN 96
AFR_SET_TRAP 96
AFR_SET_TRAP2 98
AFR_SET_WTOR 101
AFR_SETENV 102
AFR_SHRACCESS 103
AFR_SHRALLOC 104
AFR_SHRFREE 105
AFR_SHREAD 105
AFR_SHWRITE 106
AFR_START_PROGRAM 107
AFR_STOP_PROGRAM 107
AFR_SUSPEND 108
AFR_TELNET_CLOSE 108
AFR_TELNET_GETLASTERROR 109
AFR_TELNET_OPEN 109
AFR_TELNET_QUERY 110
AFR_TTELNET_SET 112
AFR_TTS_FILE 113
AFR_TTS_STRING 114
AFR_UA_WRITE 116

© Copyright IBM Corp. 2007
AFR_UNLOCK 116
AFR_USER_CHANGEPSW 117
AFR_USER_LOGONTYPE 118
AFR_USER_SETPSW 119
AFR_VOICEANSWER 121
AFR_VOICEBEEP 122
AFR_VOICEDIAL 123
AFR_VOICEGETTONES 124
AFR_VOICEHANGUP 125
AFR_VOICEINIT 126
AFR_VOICERECORD 127
AFR_VOICESPEAK 129
AFR_WHOAMI 131
AFR_WHOIS 131
ASSIGN CONSOLE function 91, 96
AUTOEXEC.REX 2
automatically executed scripts 3
automating tasks using REXX functions 2
AUTouser.REX 3

C
color control
  broadcast messages 20, 41, 67
  retrieve color of a string 50
  REXX script text 3
constants, REXX 1
customizing REXX scripts 2

D
diagnostic aid when working with serial ports 32
Dialogic voice card 115

E
e-mail
  AFR_SEND_EMAIL function 92
  AFR_SEND_MESSAGE function 94
escape sequences 3

I
intellectual property 137

L
license, patents 137
licensing
  address 137
  Web address 137
lowercase letters, REXX 1

M
Message Collector
  AFR_GET_LINE function 42
  AFR_SEND_MESSAGE function 94

N
notices
  used in this document ix
NRC notify return codes 133
null string
  in REXX 1

P
patents 137
PDF files, adding annotations x
peer REXX functions 12
printing problems x

Q
queue operation REXX functions 13
quotes
  in REXX 1

R
REXX
  and SA IOM 2
  AUTOEXEC.REX 2
  AUTouser.REX 3
  clauses 1
  constants 1
  external functions 1
  literal strings 1
  owner 3
  quotes, using 1
  shared memory usage 3
  sharing with other languages 3
  startup script 2
  terminology 1
  user-written scripts at startup 3
  variables 1
REXX functions
  AFR_ANSI 18
  AFR_ASYNC_DELETE 18
  AFR_ASYNC_WTOR 19
  AFR_AUDIT_LOGSIZE 17
  AFR_AUDIT_REOPEN_LOG 17
  AFR_BROADCAST 20
  AFR_CHECK_TRAP 21
  AFR_CHECK_WTOR 22
  AFR_CLEAR_TRAP 23
  AFR_CLEAR_WTOR 23
  AFR_CLS 24
  AFR_CNTL_TAPI_START 24
  AFR_CNTL_TAPI_STOP 25
  AFR_COMBREAK 25
  AFR_COMCLOSE 26
  AFR_COMCTRL 26
  AFR_COMGETBAUD 27
  AFR_COMGETBITS 27
  AFR_COMGETPARITY 28
  AFR_COMGETSTOPBITS 28
  AFR_COMGETSTR 29
REXX functions (continued)
AFR_COMINPUTBYTES 29
AFR_COMJOURNALSTART 30
AFR_COMJOURNALSTOP 32
AFR_COMOPEN 33
AFR_COMOUTPUTBYTES 34
AFR_COMPUTSTR 34
AFR_COMSETBAUD 35
AFR_COMSETLINEC 36
AFR_COMSTATUS 36
AFR_CONSOLE_TYPE 37
AFR_CURSOR 38
AFR_EMAIL_LOGSIZE 40
AFR_EMAIL_REOPEN_LOG 40
AFR_EMIT_UA_SNMP_TRAP 38
AFR_ERROR 41
AFR_GET_LINE 42
AFR_GET_STATUS 42
AFR_GETENV 43
AFR_GETOWNER 46
AFR_GMT 47
AFR_KBHIT 48
AFR_KEY_ET 49
AFR_LOAD 50
AFR_LOAD_COLOR 50
AFR_LOCK 52
AFR_LOG 53
AFR_LOGSIZE 54
AFR_MC_LASTMSGNUM 54
AFR_MC_LASTMSGTEXT 55
AFR_MC_LASTMSGTIME 55
AFR_MC_LOGSIZE 56
AFR_MC_REOPEN_LOG 56
AFR_MC_SEND 57
AFR_MODEMANSWER 58
AFR_MODEDCALL 58
AFR_MODEMDISCONNECT 60
AFR_MODEMSTATUS 60
AFR_NOTIFY 61
AFR_NOTIFY_ACK 62
AFR_NOTIFY_EXIT 64
AFR_NOTIFY_INIT 64
AFR_NOTIFY_LASTERROR 64
AFR_NOTIFY_SET_RESULT 65
AFR_OP_ERROR 66
AFR_PEER_CLOSE 67
AFR_PEER_GETLASTERROR 68
AFR_PEER_OPEN 69
AFR_PEER_QUERY 70
AFR_PEER_RCV 73
AFR_PEER_SEND 75
AFR_PEER_SENDRCV 77
AFR_PROMPT 79
AFR_PROMPT_USER 80
AFR_REOPEN_LOG 81
AFR_RXQUEUE_CREATE 82
AFR_RXQUEUE_DELETE 83
AFR_RXQUEUE_EXISTS 83
AFR_RXQUEUE_EXTERNAL 84
AFR_RXQUEUE_GET 84
AFR_RXQUEUE_PURGE 85

REXX functions (continued)
AFR_RXQUEUE_SET 85
AFR_SAY 86
AFR_SCREEN_WAIT 86
AFR_SELECT 87
AFR_SELECTED 88
AFR_SEND 89
AFR_SEND_3270 90
AFR_SEND.ASCII 92
AFR_SEND_EMAIL 92
AFR_SEND_MESSAGE 94
AFR_SEND_RPAGE 95
AFR_SEND_SCAN 96
AFR_SET_TRAP 96
AFR_SET_TRAP2 98
AFR_SET_WTOR 101
AFR_SETENV 102
AFR_SHRACCESS 103
AFR_SHRALLOC 104
AFR_SHRFREE 105
AFR_SHRREAD 105
AFR_SHRWRITE 106
AFR_START_PROGRAM 107
AFR_STOP_PROGRAM 107
AFR_SUSPEND 108
AFR_TELNET_CLOSE 108
AFR_TELNET_GETLASTERROR 109
AFR_TELNET_OPEN 109
AFR_TELNET_QUERY 110
AFR_TELNET_SET 112
AFR_TTS_FILE 113
AFR_TTS_STRING 114
AFR_UA_WRITE 116
AFR_UNLOCK 116
AFR_USER_CHANGEPSW 117
AFR_USER_LOGONTYPE 118
AFR_USER_SETPSW 119
AFR_VOICEANSWER 121
AFR_VOICEBEEP 122
AFR_VOICEDIAL 123
AFR_VOICEGETTONES 124
AFR_VOICEHANGUP 125
AFR_VOICEINIT 126
AFR_VOICERECORD 127
AFR_VOICESPEAK 129
AFR_WHOAMI 131
AFR_WHOIS 131
REXX functions, according to their use 7
REXX session screen
background color 3
text color 3
text format 3

S
security
REXX script owner 3
shared memory file 3
speaking text 114
starting
scripts automatically at server startup 3
T
Telnet REXX functions 12
TTS (text-to-speech) 11, 113, 114

V
variables
REXX 1
Voice
AFR_TTS_FILE 113
AFR_TTS_STRING 114
AFR_VOICEANSWER 121
AFR_VOICEBEEP 122
AFR_VOICEDIAL 123
AFR_VOICEGETTONES 124
AFR_VOICEHANGUP 125
AFR_VOICEINIT 126
AFR_VOICERECORD 127
AFR_VOICESPEAK 129
Voice functions 11