ARx_Func3.ag

COLLABORATORS

	TITLE : ARx_Func3.ag		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 8, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

ARx_Func3.ag

1.1 main

```
AN AMIGAGUIDE® TO ARexx                 Edition: 1.0a
by Robin Evans

   Note: This is a subsidiary file to ARexxGuide.guide. We recommend
   using that file as the entry point to this and other parts of the
   full guide.
```

1.2 ARexxGuide | Functions reference (7 of 12) | FILE INPUT/OUTPUT

```
    CLOSE      (<file>)
    EOF        (<file>)
    LINES      ([{STDIN | STDOUT | STDERR}])
    OPEN       (<file>, <filespec>, [<option>])
    READCH     (<file>, [<length>])
    READLN     (<file>)
    SEEK       (<file>, <offset>, [<anchor>])
    WRITECH    (<file>,<string>)
    WRITELN    (<file>,<string>)

Related function:
    EXISTS

                     Also see  File management functions
                               Informational functions
```

The functions in this list give to an ARexx script control over input and output, not just to disk files, but also to such devices as console windows and printers, which act much like standard files in the view of AmigaDOS.

The following nodes explain in more depth the use of I/O functions and instructions in ARexx.

```
    Overview of I/O functions
    Setting the logical file name
    Using I/O functions other devices
    Standard I/O files
```

Because the I/O functions of REXX, the language from which ARexx was born, were not fully implemented in the early versions of REXX on other systems, these functions are Amiga-specific extensions to the language.

Next: File mgt. func. | Prev: Information func. | Contents: Function ref.

1.3 ARexxGuide | Functions reference | File I/O (1 of 9) | CLOSE

```
CLOSE(<file>)
      returns a Boolean value
```

Closes the specified <file>. 0 will be returned if the file had not been opened previously.

<file> is the logical name assigned to the file with the OPEN() function. The name is case-sensitive, although an unassigned symbol may be used, in which case, it will be automatically translated to upper-case by ARexx and can therefore be entered in mixed case here.

```
      NOTE: ARexx automatically closes all opened files when a program
      ends -- even if it ends with some type of external interrupt -- so
      an error will not be generated if files are not explicitly closed
      with this function.

   Also see @{ " OPEN          " link OPEN()}
```

Next: EOF() | Prev: File I/O func. | Contents: File I/O func.

1.4 ARexxGuide | Functions reference | File I/O (2 of 9) | EOF

```
EOF(<file>)
      returns a Boolean value
```

The result is FALSE (0) until the end of the specified <file> has been reached.

```
   Also see @{ " READLN        " link READLN()}
            @{ " READCH        " link READCH()}
            @{ " SEEK          " link SEEK()}
```

Next: LINES() | Prev: CLOSE() | Contents: File I/O func.

1.5 ARexxGuide | Functions reference | File I/O (3 of 9) | LINES

```
LINES([{STDIN | STDOUT | STDERR}])
      returns a number

The result is the number of lines queued or typed ahead at the logical
device specified by the argument string, which must refer to an
interactive stream.

If the argument string is omitted, the result is the number of lines on
the program stack of  STDIN .

      NOTE: This function requires the 2.0+ AmigaShell,  WShell , or
      another shell managed by ConMan.

      Example:
         /**/
         push 'cd sys:'
         queue 'run program'
         say lines()                >>> 2

   Also see @{ " PUSH          " link ARx_Instr3.ag/PUSH} instruction
            @{ " QUEUE         " link ARx_Instr3.ag/QUEUE} instruction
            @{ " PULL          " link ARx_Instr3.ag/PULL} instruction

Next: OPEN() | Prev: EOF() | Contents: File I/O func.
```

1.6 ARexxGuide | Functions reference | File I/O (4 of 9) | OPEN

```
OPEN(<file>, <filespec>, [<option>])
      returns a Boolean value

Opens a file with the name specified by <filespec>.

<file> is a logical name that will be used by other functions that
communicate with the channel. It may be any expression -- most often a
literal string, unassigned symbol, or variable name. The result of the
expression is used as the logical name, which is case-sensitive.
                                  NOTE: Naming logical files

<filespec> may be any valid device or filename. 'PRT:' may be used as
<filespec> to allow output to a printer.

The <option> (which is READ by default) determines the mode in which the
file is opened. Only the first character { A|R|W } need be used to specify
the <option>.

   'APPEND'  -- An existing file will be opened for input with the pointer
             located at the end. Although it is usually used to add more
             information to an existing file, the read functions are still
             available when a file is opened in this manner. This option
             establishes a non-exclusive lock on the file.
   'READ'    -- An existing file will be opened with the pointer located
             at the beginning of the file. Although it is usually used to
             read information from an existing file, the write functions
             are still available when a file is opened with this option.
             This option establishes a non-exclusive lock on the file.
   'WRITE'   -- A new file will be opened for input. If a file of the same
             name exists, it will be replaced by the new file. Although it
             is usually used to add information to a new file, the read
             functions are still available when a file is opened with this
             option. This option establishes a exclusive lock on the file.

Because OPEN() returns a  Boolean value , it is often used in an  IF
instruction which allows for handling error conditions arising from
failure to open the specified file.

   Examples:
         /* open a channel to the printer                               */
       if open('PRINTER', 'PRT:', 'W') then ...
          /* if [WinSpec} contains valid  CON:  specs, this will open    **
          ** a console window                                           */
       if open(.Win, WinSpec, 'W') then ...
         /* open an existing file for more data                         */
       if open(OldFile, FileName, 'A') then ...
         /* create a new file                                           */
       if open('NEWFILE', 't:Information.data', 'W') then ...
         /* open an existing file for reading data                      */
       if open(.IFile, FileName, 'R') then ...

   Also see @{ " CLOSE           " link CLOSE()}
            @{ " READLN          " link READLN()}
            @{ " READCH          " link READCH()}
            @{ " WRITELN         " link WRITELN()}
            @{ " WRITECH         " link WRITECH()}
            @{ " SIGNAL ON IOERR " link ARx_Instr3.ag/IOERR}

Next: READCH() | Prev: LINES() | Contents: File I/O func.
```

1.7 ARexxGuide | Functions reference | File I/O (5 of 9) | READCH

```
READCH(<file>, [<length>])
      returns a string

Returns the number of characters specified by <length> (the default is 1)
from the logical <file>, which must have been opened with a prior call to
 OPEN() .

<file> is the  logical name  assigned to the file with the OPEN() function.

   Also see @{ " READLN        " link READLN()}
            @{ " WRITECH       " link WRITECH()}

Next: READLN() | Prev: OPEN() | Contents: File I/O func.
```

1.8 ARexxGuide | Functions reference | File I/O (6 of 9) | READLN

```
READLN(<file>)
      returns a string

Returns a string of characters from the logical <file> which must have
been opened with a prior call to  OPEN()  The function will
read characters from <file> until it encounters a line-feed character,
which will not be included in the returned value.

<file> is the  logical name  assigned to the file with the OPEN() function.

   Example:
         /* This is a simple word-counting program. It reads each line **
         ** in a file and counts the words. Because the contents of a  **
         ** line are not important, READLN() is  nested  within the    **
         **  WORDS()  function.                                        */
      arg FileName
      if FileName = '' | FileName = '?' then do
         say 'WordCount <FileName>'
         say '  Specify the name of the file to be counted.'
         exit 0
      end
      WdTotal = 0
      if open(.IFile, FileName, 'R') then do
         say 'Counting words in' FileName'.'
         do until eof(.IFile)
            WdTotal = WdTotal + words(READLN(.IFile))
         end
         say 'There are' WdTotal 'words in' FileName'.'
      end
      else do
         say 'WordCount failed. File not found'
         exit 20
      end

   Also see @{ " READCH         " link READCH()}
            @{ " WRITELN        " link WRITELN()}

Next: SEEK() | Prev: READCH() | Contents: File I/O func.
```

1.9 ARexxGuide | Functions reference | File I/O (7 of 9) | SEEK

```
SEEK(<file>, <offset>, [<anchor>])
      returns a number

Moves the  pointer  <offset> number of bytes from the <anchor> to a new
position in the logical <file>. The <anchor> may be 'BEGIN', 'CURRENT', or
'END'. (Only the first character need be used.) The default <anchor> of
'C' will be used if nothing else is specified.

If 'E' is the anchor, then <offset> should be a negative number to move
the pointer backwards by <offset> bytes.

The result is the new byte position relative to the beginning of the file.
```

```
<file> is the  logical name  assigned to the file with the OPEN() function.

   Example:
         /* This example depends on a previous assignment to variables **
         ** [NamesDB] which would be the name of a file on disk,       **
         ** [RecSize] which would be the length in bytes of one record **
         ** in the file, and [RecNum] which would be the sequential    **
         ** number of the record to be retrieved.                      */
      if open(DBFile, NamesDB, 'R') then do
         CurRecPos = SEEK(DBFile, RecNum * RecSize, 'B')
         Rec.RecNum = readch(DBFile, RecSize)
      end

   Also see @{ " OPEN          " link OPEN()}
            @{ " READCH        " link READCH()}
            @{ " READLN        " link READLN()}
            @{ " EOF           " link EOF()}

Next: WRITECH() | Prev: READLN() | Contents: File I/O func.
```

1.10 ARexxGuide | Functions reference | File I/O (8 of 9) | WRITECH

```
WRITECH(<file>,<string>)
      returns a Boolean value

Writes the character(s) in <string> to the logical <file>, which must have
been opened with a prior call to  OPEN() .

This function will not append a newline character to <string>.

<file> is the  logical name  assigned to the file with the OPEN() function.

   Also see @{ " WRITELN       " link WRITELN()}
            @{ " READLN        " link READLN()}
            @{ " SEEK          " link SEEK()}

Next: WRITELN() | Prev: SEEK() | Contents: File I/O func.
```

1.11 ARexxGuide | Functions reference | File I/O (9 of 9) | WRITELN

```
WRITELN(<file>,<string>)
      returns a Boolean value

Writes <string> to the logical <file>, which must have been opened with a
prior call to  OPEN()  The function appends a line-feed
character to the string.

<file> is the  logical name  assigned to the file with the OPEN() function.

   Also see @{ " WRITECH       " link WRITECH()}
            @{ " READLN        " link READLN()}
            @{ " SEEK          " link SEEK()}
```

Next: File I/O func. | Prev: WRITECH() | Contents: File I/O func.

1.12 ARexxGuide | Functions reference (8 of 12) | FILE MANAGEMENT

```
    DELETE    (<filespec>)
    EXISTS    (<filespec>)
    MAKEDIR   (<dirname>)
    RENAME    (<oldfile>, <newfile>)
    STATEF    (<filespec>)

                     Also see  File input/output functions
```

Although each of these functions could be replaced by calls to AmigaDOS commands such as {address command 'delete' <file> }, the functions here are significantly quicker than such constructions and more informative. Since they return a value within variable space of the calling script, it is far easier to handle conditions that cause one of the functions to fail.

Next: ARexx control func. | Prev: File I/O func. | Contents: Function ref.

1.13 ARexxGuide | Functions reference | File Mgt. (1 of 5) | DELETE

```
                                     a rexxsupport.library function
DELETE(<filespec>)
      returns a Boolean value
```

Deletes the file specified by <filespec>. Returns 1 if the file was found and successfully deleted.

```
   Example:
         say delete('t:tempfile');    >>> 1 /* if the file was found */
```

Next: EXISTS() | Prev: File mgt. func. | Contents: File mgt. func.

1.14 ARexxGuide | Functions reference | File Mgt. (2 of 5) | EXISTS

```
                                     a rexxsupport.library function
EXISTS(<filespec>)
      returns a Boolean value
```

Checks the Amiga file system for the presence of a file named <filespec>, which may include full path specifications. If only a partial path specification is included, the search is made relative to the current directory.

```
   Example:
         say exists('sys:system/rexxmast');      >>> 1

   Also see @{ " SHOWLIST      " link ARx_Func2.ag/SHOWLIST()}
```

```
            @{ " PRAGMA        " link PRAGMA()}
            @{ " MAKEDIR       " link MAKEDIR()}
```

Note: SHOWLIST('A') returns a list (in upper case and without the ':') of all currently assigned directories. SHOWLIST('V') returns a similar list of currently mounted volumes. The lists can be used to check for the presence of a file device specification.

When EXISTS() is used to check for the existence of a file on a device that might not be available, the system requester that asks "Please insert volume..." can be suppressed through use of PRAGMA('W', 'N') .

PRAGMA('D', <dir>) will change the default directory examined by EXISTS() to that specified by <dir>.

Next: MAKEDIR() | Prev: DELETE() | Contents: File mgt. func.

1.15 ARexxGuide | Functions reference | File Mgt. (3 of 5) | MAKEDIR

```
                                     a rexxsupport.library function
MAKEDIR(<dirname>)
      returns a Boolean value
```

Creates a new directory, like the AmigaDOS command of the same name.

This is one of the rare cases where an ARexx function works differently with different versions of the Amiga operating system. Under AmigaDOS 1.3, the function returns 1 (TRUE) even if the directory already exists, so the call can be made to ensure that a directory exists. Under Release 2.04 and higher, however, the return value is 0 (FALSE) if the directory already exists.

A return of FALSE might also occur under any version of the OS if the specified volume is not available or is full.

```
   Example:
         say makedir('env:ARexxGuide')    >>> 1
```

Next: RENAME() | Prev: EXISTS() | Contents: File mgt. func.

1.16 ARexxGuide | Functions reference | File Mgt. (4 of 5) | RENAME

```
                                     a rexxsupport.library function
RENAME(<oldfile>, <newfile>)
      returns a Boolean value
```

Renames <oldfile> to <newfile>.

Next: STATEF() | Prev: MAKEDIR() | Contents: File mgt. func.

1.17 ARexxGuide | Functions reference | File Mgt. (5 of 5) | STATEF

```
                                     a rexxsupport.library function
STATEF(<filespec>)
      returns a string

Returns information about the file named <filespec>. The status string for
a file is formatted as

   FILE|DIR <bytes> <blocks> <protect-flags> <days> <min> <ticks> <comment>

   <protect-flags> are reported in the order HSPARWED with a dash "-" if
      the attribute isn't present.
   <days> is the number of days since January 1, 1978
   <min> is the number of minutes since midnight
   <ticks> is the number of tick intervals (1/50 second) in the minute.

   Examples:
         say statef('sys:rexxc');      >>> DIR 0 0 ----RWED 5362 727 2702
         say statef('sys:rexxc/tco'); >>> FILE 364 1 --P-RWED 5362 727 2688

   Also see @{ " SHOWDIR       " link ARx_Func2.ag/SHOWDIR()}
            @{ " PRAGMA        " link PRAGMA()}

Next: File mgt. func. | Prev: RENAME() | Contents: File mgt. func.
```

1.18 ARexxGuide | Functions reference (9 of 12) | ARexx CONTROL

```
    ADDRESS   ()
    ADDLIB    (<name>, <priority>, [offset, version])
    ARG       ([<argnumber>], ['EXISTS' | 'OMITTED'])
    DATATYPE  (<string>, [<type>])
    DELAY     (<number>)
    DIGITS    ()
    ERRORTEXT (<number>)
    FORM      ()
    FUZZ      ()
    GETCLIP   (<name>)
    PRAGMA    (<option> [,<value>])
    REMLIB    (<libname>)
    SETCLIP   (<clipname>, [<value>])
    SOURCELINE([<line number>])
    SYMBOL    (<name>)
    TRACE     ([<option>])
    VALUE     (<name>)

                     Also see  Message port functions

This list includes a variety of functions that give the programmer control
over the script itself. Some of the functions, like TRACE(), SOURCELINE(),
and ERRORTEXT() will be useful mainly for debugging a program under
development. The two clip functions let one ARexx script set up variables
that can be read by any other script. VALUE() extends the naming and
referencing power of variable symbols while SYMBOL() and DATATYPE() allow
```

for greater control over the typeless variables in ARexx.

ADDLIB() is an Amiga extensions to the standard language definition that
give ARexx access to the power of external libraries .

ADDRESS() returns information about the effect of the instruction with the
same name just as DIGITS(), FUZZ(), and FORM() reveal the settings of the
instruction NUMERIC .

Finally, the ARG() function can replace, in some instances, use of the
 ARG instruction.

Next: Port mgt. func. | Prev: File mgt. func. | Contents: Function ref.

1.19 ARexxGuide | Functions reference | ARexx control (1 of 17) | ADDRESS

```
ADDRESS()
      returns a string
```

The result is the name of the ARexx port to which commands are currently
being submitted.

```
   Examples:
         say address();               >>> WSH_4
         say address();               >>> TURBOTEXT2

   Also see @{ " ADDRESS        " link ARx_Instr.ag/ADDRESS} instruction
            @{ " PARSE SOURCE   " link ARx_Instr2.ag/PARSE} instruction
            @{ " Current host   " link ARx_Elements3.ag/HOST} Basic elements  ↩
               explanation
```

Next: ADDLIB() | Prev: ARexx control func. | Contents: ARexx control func.

1.20 ARexxGuide | Functions reference | ARexx control (2 of 17) | ADDLIB

```
ADDLIB(<name>, <priority>, [offset, version])
      returns a Boolean value
```

Adds a function library or function host to the Library List maintained
by the resident process.

The <name> argument is case sensitive. If a library is specified, it
should be located in the LIBS: directory. If a function host is specified,
then <name> refers to the public message port associated with the host.

<priority> is an integer between -100 and 100 and refers to the search
priority to be used by the resident process in case of duplicate function
names in the Library List.

<offset> and <version> are used only for function libraries. The numbers
to be used should be specified by the library’s developer.

```
   Examples:
         call addlib('rexxsupport.library',0,-30,0)
         call addlib('rexxmathlib.library',0,-30,0)
         call addlib('rexxarplib.library',0,-30,0)
            /* the following adds  function host  program           */
         if ~show('p','QuickSortPort') then
         address command
         do
            'run >nil: quicksort'
            do for 5 while ~show('p','QuickSortPort')
               '  WaitForPort  "QuickSortPort" '
            end
            if show('p','QuickSortPort') then
               call addlib('QuickSortPort',-30)
         end

   Also see @{ " REMLIB            " link REMLIB()}
            @{ " RXLIB             " link ARx_Cmd.ag/RXLIB} command
            @{ " Library functions " link ARx_Elements3.ag/LIBFUNC} Basic Elements ↩
                explanation
```

The library named as an argument to this function is not actually loaded. ARexx doesn't even check to see if the library exists. The library is actually loaded only when ARexx needs it to find an unmatched function call. Specifying a non-existent library with this function may cause a syntax error much later:

```
   +++ Error 14 in line <#>: Requested library not found
```

Line <#> will indicate a line containing a function call. Using an invalid library name with ADDLIB() can cause valid function names to be unrecognized because ARexx might check for the function first within the invalid library.

Next: ARG() | Prev: ADDRESS() | Contents: ARexx control func.

1.21 ARexxGuide | Functions reference | ARexx control (3 of 17) | ARG

```
ARG([<argnumber>], ['EXISTS' | 'OMITTED'])
   returns a number
           or a string
           or a Boolean value
```

Without arguments ARG() returns the number of arguments supplied when the current program or function was executed.

If only <argnumber> is specified, then the argument string in that position is returned or a null string if nothing is specified in that position.

The 'EXISTS' and 'OMITTED' options (for which only the first letter need be supplied) test whether the specified <argnumber> was used and returns a Boolean value .

      Note: Any arguments supplied on the shell are considered part of one

```
      string, even if the string contains commas.

   Examples:
            assume the program was started from a shell with:
            prg Foo, Widget
         say arg();            >>> 1
         say arg(1);           >>> Foo, Widget
         say arg(2,E);         >>> 0

            assume this call to an internal or external routine:
            call prg 'Foo',, 'Widget'
         say arg();           >>> 3
         say arg(1);          >>> Foo
         arg(2,E);            >>> 0
         say arg(3);          >>> Widget

   Also see @{ " PARSE ARG     " link ARx_Instr2.ag/PARSE} instruction

Next: DATATYPE() | Prev: ADDLIB() | Contents: ARexx control func.
```

1.22 ARexxGuide | Functions reference | ARexx control (4 of 17) | DATATYPE

```
DATATYPE(<string>, [<type>])
returns either 'NUM' or 'CHAR'
        or a Boolean value

If only <string> is specified, 'NUM' will be returned if <string> is a
valid REXX number in any format or 'CHAR' for any other input.

When a  <type>  (A|B|L|M|N|S|U|W|X) is specified, the result
is a Boolean value indicating whether the supplied <string> is a valid
value of that type.

   Examples:
         say datatype(A)                 >>> CHAR
         A = 1; say datatype(A)          >>> NUM
         A = 'Molloy'; say datatype(A)   >>> CHAR
         A = 'Molloy';say datatype(A, M) >>> 1

   Also see @{ " VERIFY        " link ARx_Func.ag/VERIFY()}
            @{ " ABS           " link ARx_Func2.ag/ABS()}
            @{ " SIGN          " link ARx_Func2.ag/SIGN()}
            @{ " SYMBOL        " link SYMBOL()}

                                  NOTE: Checking unique datatypes

Next: DELAY() | Prev: ARG() | Contents: ARexx control func.
```

1.23 ARexxGuide | Functions reference | ARexx control | DATATYPE (1 of 1) | OPTIONS

Only the first letter of the following option keywords need be used with the DATATYPE() function.

Keywords Accepted	Values which yield TRUE result
Numeric	Valid number
Whole	Integer
X	Hex digits/alpha string
Binary	Binary digits string
Alphanumeric	A-Z,a-z, or digits 0-9
Upper	Uppercase alphabetic A-Z
Lowercase	Lowercase alphabetic a-z
Mixed	Mixed alphabetic A-Z,a-z
Symbol	Valid REXX symbol

Samples:

Function	Result	Comment
datatype(45.78, 'n')	1	
datatype(3.32e9, 'n')	1	Exponential notation is recognized.
datatype(45.78, 'w')	0	
datatype(1011,'b')	1	
datatype('A43BD', 'x')	1	
datatype('A43BD', 'a')	1	
datatype('Amiga','a')	1	
datatype(333,'a')	1	
datatype(33.1,'a')	0	The '.' is not alphanumeric.
datatype('molloy', 'u')	0	
datatype('Amiga', 'l')	0	
datatype('unnamable', 'l')	1	
datatype('Amiga', 'm')	1	
datatype('Yeltzin', 's')	1	
datatype('Ram:', 's')	0	':' is not valid in symbols

Next: DATATYPE() | Prev: DATATYPE() | Contents: DATATYPE()

1.24 ARexxGuide | Functions reference | ARexx control (5 of 17) | DELAY

```
                                     a rexxsupport.library function
DELAY(<number>)
      return value is insignificant
```

Waits for the specified <number> of ticks (1/50 second) and then returns.

This function should be used rather than a busy-loop when an ARexx program must be suspended for a set period. DELAY() frees the computer to execute other tasks while the program is waiting.

```
   Example:
         call delay(100)         >>> (2 seconds)

   Also see @{ " TIME          " link ARx_Func2.ag/TIME()}
```

Next: ERRORTEXT() | Prev: DATATYPE() | Contents: ARexx control func.

1.25 ARexxGuide | Functions reference | ARexx control (6 of 17) | ERRORTEXT

```
ERRORTEXT(<number>)
      returns a string

The result is the error text associated with ARexx error <number>, or a
null string if nothing is defined for that number.

   Example:
         say errortext(5);                      >>> Unmatched quote

   Also see @{ " SOURCELINE    " link SOURCELINE()}
```

Next: DIGITS() | Prev: DELAY() | Contents: ARexx control func.

1.26 ARexxGuide | Functions reference | ARexx control (7 of 17) | DIGITS

```
DIGITS()
      returns a number

The result is the current  NUMERIC  DIGITS setting.

   Example:
         numeric digits 6
         say digits()             ==> 6

   Also see @{ " FORM           " link FORM()}
            @{ " FUZZ           " link FUZZ()}
            @{ " PARSE NUMERIC " link ARx_Instr2.ag/PARSE}
```

Next: FORM() | Prev: ERRORTEXT() | Contents: ARexx control func.

1.27 ARexxGuide | Functions reference | ARexx control (8 of 17) | FORM

```
FORM()
      returns a string

The result is the current setting of the  NUMERIC  FORM instruction.

   Also see @{ " DIGITS         " link DIGITS()}
            @{ " FUZZ           " link FUZZ()}
            @{ " PARSE NUMERIC " link ARx_Instr2.ag/PARSE}
```

Next: FUZZ() | Prev: DIGITS() | Contents: ARexx control func.

1.28 ARexxGuide | Functions reference | ARexx control (9 of 17) | FUZZ

```
FUZZ()
      returns a number

The result is the current  NUMERIC  FUZZ setting.

   Example:
         numeric fuzz 3
         say fuzz()                    >>> 3

   Also see @{ " DIGITS        " link DIGITS()}
            @{ " FORM          " link FORM()}
            @{ " PARSE FUZZ    " link ARx_Instr2.ag/PARSE}

Next: GETCLIP() | Prev: FORM() | Contents: ARexx control func.
```

1.29 ARexxGuide | Functions reference | ARexx control (10 of 17) | GETCLIP

```
GETCLIP(<name>)
      returns a string

Returns the value associated with clip <name>. The search for <name>
in the clip list is case sensitive. A null string is returned if a clip of
the specified name is not found.

   Example:
         say setclip('Molloy','Samuel Beckett');  >>> 1
         say getclip('Molloy');                   >>> Samuel Beckett
            /*  The following has no result because the clip name is   **
            **  case-sensitive. Leaving out the quotes converts the     **
            **  name to uppercase                                       */
         say getclip(Molloy);                     >>>

                                 NOTE: Using the clip list

   Also see @{ " SETCLIP       " link SETCLIP()}

Next: PRAGMA() | Prev: FUZZ() | Contents: ARexx control func.
```

1.30 ARexxGuide | Functions reference | ARexx control (11 of 17) | PRAGMA

```
PRAGMA(<option> [,<value>])
      returns a string
           or a Boolean value

Allows an ARexx program to change some attributes of the system
environment. The  <option>  argument specifies the environmental
attribute. A specific <value> is expected for each type of <option>.

   Also see @{ " SHOWLIST      " link ARx_Func2.ag/SHOWLIST()}
```

Next: REMLIB() | Prev: GETCLIP() | Contents: ARexx control func.

1.31 ARexxGuide | Functions reference | ARexx control | PRAGMA (1 of 1) | OPTIONS

```
These are the options that are available with PRAGMA(). Note that only the
first letter of the option name need be provided.

   Option      Value       Explanation
   ---------   ----------  -----------------------------------------
   Directory   [<dir>]     If <dir> is specified, the 'current'
                           directory for the running ARexx program is
                           changed. (This does not affect the current
                           directory of the host.)

                           PRAGMA(D) without a <value> returns the
                           name of the current directory.

   ID                      Returns a hexadecimal string which is the task
                           ID for the currently executing script. If
                           several copies of the same exec are running at
                           once, this number can be used to distinguish
                           them. It might be useful when setting the name
                           of a port to be used with the  OPENPORT()
                           function.


   Priority    [<number>]  Controls the system priority of the currently
                           executing script, much like the AmigaDOS
                           command SETPRI.

                           If <number> is omitted, the function returns the
                           current priority setting.

                           If <number> is included, the priority will be
                           changed to that value. The number of the
                           previous priority will be returned.

                           <number> may be between -127 and 127, but should
                           be restricted to a far more limited range and
                           should never be greater than the priority of
                           the resident process (which usually runs at 4).

   Stack       [<number>]  sets the stack size for a program launched by
                           the current exec and returns the stack size
                           previously set.

                           If <number> is omitted, the function will
                           return the size of the current stack.

   *           [<name>]    defines the specified logical name as the
                           current ("*") console handler, thereby
                           allowing the user to open two streams on
                           one window. This option appears to be unneeded
                           on most current shells.

   Window      [{'N'| 'W'}]   Controls the display of system requesters
                                 (like 'Please insert volume...'). If the 'N'
                           or 'Null' option is used, such requesters won't
                           appear at all. The 'W' or 'Workbench' option is
                           the default. It causes the requesters to be
                           displayed on the Workbench screen and can also
                           be called by using PRAGMA('W') without a second
                           option.
```

Next, Prev & Contents: PRAGMA()

1.32 ARexxGuide | Functions reference | ARexx control (12 of 17) | REMLIB

```
REMLIB(<libname>)
      returns a Boolean value
```

Removes the name of a library or function host the the list maintained by the resident process. The library is not actually removed from memory, but may be purged by the system when needed.

The function is most useful when an the name of a non-existent library was used with the ADDLIB() function. Keeping such a name on the library list may cause ARexx to search for the library each time a function is called and, in some circumstances, will prevent a function which is present from being found. This function will remove the name from the list.

```
   Also see @{ " ADDLIB       " link ADDLIB()}
```

Next: SETCLIP() | Prev: PRAGMA() | Contents: ARexx control func.

1.33 ARexxGuide | Functions reference | ARexx control (13 of 17) | SETCLIP

```
SETCLIP(<clipname>, [<value>])()
      returns a Boolean value
```

Sets the <value> associated with clip <name> or deletes the <named> clip if <value> is not specified. The search for <name> within the clip list is case sensitive.

```
   Example:
         say setclip('Molloy','Samuel Beckett');  >>> 1
         say getclip('Molloy');                   >>> Samuel Beckett

                                 NOTE: Using the clip list

   Also see @{ " GETCLIP       " link GETCLIP()}
            @{ " RXSET         " link ARx_Cmd.ag/RXSET} command
```

Next: SOURCELINE() | Prev: REMLIB() | Contents: ARexx control func.

1.34 ARexxGuide | Functions reference | ARexx control (14 of 17) | SOURCELINE

```
SOURCELINE([<line number>])
      returns a string
           or a number

The result is the text of the specified <line number> in the currently
executing ARexx program.  If the line argument is omitted, the function
returns the total number of lines in the file.

This function is often used to embed "help" information in a program.

   Examples:
         /* A simple test program */
         say sourceline()            >>> 3
         say sourceline(1)           >>> /* A simple test program */

                                 NOTE: Using in-line data

   Also see @{ " ERRORTEXT      " link ERRORTEXT()}
            @{ " SIGL           " link ARx_Elements2.ag/SIGL} Special variable:  ↩
                Basic elements explanation

Next: SYMBOL() | Prev: SETCLIP() | Contents: ARexx control func.
```

1.35 ARexxGuide | Functions reference | ARexx control (15 of 17) | SYMBOL

```
SYMBOL(<name>)
      returns 'BAD', 'VAR', or 'LIT'

'BAD' is returned if <name> is not a valid ARexx symbol. 'VAR' indicates
that the <name> is an ARexx variable with an assigned value. 'LIT'
indicates that <name> is either a  variable symbol  that has not been
assigned a value or a  constant .

   Examples:
         say symbol('A');       >>> LIT
         A = 'foo';
         say symbol('A');       >>> VAR
         say symbol('A%')       >>> BAD

   Also see @{ " DATATYPE       " link DATATYPE()}
            @{ " ABS            " link ARx_Func2.ag/ABS()}

Next: TRACE() | Prev: SOURCELINE() | Contents: ARexx control func.
```

1.36 ARexxGuide | Functions reference | ARexx control (16 of 17) | TRACE

```
TRACE([<option>])
      returns a character

returns information about the current tracing mode, or sets the tracing
```

```
mode in the same way as the  TRACE  instruction.

If  <option>  isn't specified, then a character indicating the current
trace option is returned or 'N' if the default normal tracing is in effect.

The <option> may be any expression that yields one of the characters
associated with the TRACE instruction. When an option is specified, the
result is the trace condition previously in effect, which may be used to
reset the tracing mode later in the program.

Unlike the trace instruction, this function will alter the trace mode from
within a program even if interactive tracing is active.

The { ? } and { ! } characters may be used alone { TRACE('?') } or with
any of the letter options { TRACE('?R') }. They act as toggles: Used once,
they turn the option on; used a second time, they turn it off

   ?  is the toggle for  interactive_tracing
   !  is the toggle for  command_inhibition

   Experiment with trace options:
    Run interactive example    *

   Examples:
         say trace()             >>> N
         trace ?I; say trace()   >>> ?I
         say trace(off)          >>> N
```

Next: VALUE() | Prev: SYMBOL() | Contents: ARexx control func.

1.37 ARexxGuide | Functions reference | ARexx control (17 of 17) | VALUE

```
VALUE(<name>)
     returns a string
          or a number

The result is the value of the ARexx symbol <name>. <name> can be any
expression that returns a valid  symbol token .

   Examples:
            /* the same thing as SAY A */
         A = 'foo'; say value(A)                        >>> foo
            /* outputs value of VarMix */
         VarMix = 4; Foo= 'Mix'; say value('Var'Foo)     >>> 4
            /* outputs assignment to Sub since the value of Foo **
            ** is substituted, 'Sub' and passed to SAY          */
         Sub = 8; Foo = 'Sub'; say value(Foo)           >>> 8
            /* A. is a different var than A so there's no assignment */
         foo.1 = 67; a = foo; say a.1                   >>> A.1
            /* the value of A is substituted. Output value of FOO.1  */
         foo.1 = 67; A = 'foo'; say value(A'.1')           >>> 67

      /**/
         Name = 'Bob'; Bob='Mary'; Mary='Sarah'
         say Name 'is married to' value(name)
         say 'His mother-in-law is' value(value(name))
                                        >>> Bob is married to Mary
                                        >>> His mother-in-law is Sarah

   Also see @{ " INTERPRET     " link ARx_Instr.ag/INTERPRET} Instruction

                                  NOTE: Finding VALUE()

Next: ARexx control func. | Prev: TRACE() | Contents: ARexx control func.
```