# ARx_Func3.ag

| COLLABORATORS | | | |
|---|---|---|---|
| | *TITLE* :<br><br>ARx_Func3.ag | | |
| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
| WRITTEN BY | | April 17, 2022 | |

| REVISION HISTORY | | | |
|---|---|---|---|
| NUMBER | DATE | DESCRIPTION | NAME |
| | | | |

# Contents

# Chapter 1

# ARx_Func3.ag

## 1.1 "

```
AN AMIGAGUIDE® TO ARexx                  Second edition (v2.0)
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 console windows, printers, and other devices that 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
Compatibility issues:
ARexx file I/O function do not follow the REXX standard. The function names are different. They work differently than standard functions.

Standard REXX I/O
Next: File mgt. func. | Prev: Information func. | Contents: ↩
Function ref.

## 1.3 ARexxGuide | Functions reference | File I/O (1 of 5) | OVERVIEW

Overview of file I/O functions
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The most basic of the file I/O functions is
OPEN()
, which gives the
opened file a 'logical name' that the other functions like READLN(), WRITECH(), and SEEK() will then use when acting on that file. The logical name used with the OPEN() function can be any literal string or symbol . The name has significance only for the current script.

The input functions are
READLN()
, which reads characters from the
specified file until an ASCII 10 end-of-line character is encountered, and

READCH()
, which reads one character by default but can be made to read a
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

specified number of characters.

The complimentary output functions are
WRITELN()
, which adds a specified
string to a file and appends an end-of-line (EOL) character to the string,
and
WRITECH()
, which adds characters to the file without adding the EOL
character.

The
EOF()
function returns a Boolean flag of 1 (TRUE) when the end of a
file has been reached. The
SEEK()
function moves to a specified point
within the file.

As information is read from or written to a disk file, ARexx (through AmigaDOS) keeps track of the current position within the file with what is called a file pointer. When a file is first opened, the initial position of the pointer is determined by the <mode> argument in OPEN(<handle>, <file name>, <mode>).

The <mode> may be:

R for read (the default -- used when nothing else is specified),
Opens an existing file.
File pointer is at the beginning of the file.
A for append
Opens an existing file.
File pointer is at the end of the file.
W for write.
Creates a new file or truncates an existing file of the same name.
File pointer is at the beginning of the file.

The OPEN() function will fail and return a value of 0 if a 'R' or 'A' mode is specified for a file that does not yet exist. If the 'W' mode is specified, any existing file of the same name will be truncated (deleted) without warning.

The mode used to open a file does not affect the other I/O functions. It is possible to read from a file opened in 'W' or 'A' mode and it is possible to write to a file opened in 'R' mode. Unless
SEEK()
is used to
reposition the pointer, however, there will be nothing to read when the file pointer is located at the end of a file as it is in 'A' and 'W' modes. Writing to an existing file with the pointer located at its beginning will overwrite existing data.

The
SEEK()
function performs two tasks: it returns the current byte position within a file and may be used to move the file pointer to a new location. Because the AmigaDOS file system is byte-oriented rather than

line-oriented, there is no simple way to move to the beginning of a new line unless the lines are all of the same length.

AmigaDOS allows for different levels of access protection for opened files. ARexx uses two of those levels. Files opened in write mode are given an exclusive lock: until it is closed, the file cannot be accessed except through use of its ARexx handle by the script that opened the file. Files opened in the other modes are given a non-exclusive lock: not only may other processes have access to the file, but the same file can be the subject of multiple OPEN() statements.

Next: LOGICAL FILE NAMES | Prev: File I/O | Contents: File I/O

## 1.4 ARexxGuide | Functions reference | File I/O (2 of 5) | FILE NAMES

```
                    Naming logical files
   ~~~~~~~~~~~~~~~~~~~~
```

When a file or other device is opened using the
OPEN()
function, it is
given a logical name. In the original manual to ARexx, Bill Hawes uses a string for the logical name:

```
      say open('outfile', 'ram:temp', 'W')
```

Using a literal string makes it apparent that no assignment takes place in the function. 'outfile' is simply a name used to refer to the file. It isn't assigned an address or anything else.

The problem with this usage is that the name becomes case sensitive. The following will generate an error:

```
      call writeln('Outfile', String)
```

'Outfile' and 'outfile' are not the same name because of the difference in letter-case. Such a subtle difference might give rise to what Cowlishaw calls a "high astonishment factor." He notes, "If a feature, accidentally misused, gives apparently unpredictable results, then it has a high astonishment factor and is therefore undesirable."

That's a good test for each programmer of the best method to use when naming files. If a using a literal string often gives rise to errors, then it is probably better to avoid the usage.

Fortunately, REXX is a language designed to be adaptable to different styles, but most of all it is a language designed to use something as close as possible to a natural English-like style.

Any valid symbol can be used as the logical name. Entering the names without quotation marks -- as simple symbols -- means that the name will be treated as upper-case by ARexx no matter how it is written. The disadvantage of this construction is that the name could be used later in a variable assignment , which would change its value and make it no longer the same name for the purposes of the file I/O functions -- another

astonishing situation.

There is an interesting third alternative to using a literal (quoted) string or a variable symbol; an alternative which, like using a literal string, prevents the accidental assignment of a new value to <name>, but which also -- like the use a simple symbol -- preserves the general case insensitivity of REXX statements. The third alternative? Use a constant symbol for the name.

Unlike the symbols used for variables, constants cannot be assigned a value. There's no danger of accidentally using the symbol for something else. Constants are usually numbers (567.43 is a constant symbol, for instance), but they don't have to be. Any token beginning with a digit is considered a constant, so a symbol like '6Input' can be used as <name> in the OPEN() function. The name will be case insensitive since ARexx will translate it each time to uppercase. (A period can also be used as the first character in a constant symbol, but the ANSI REXX committee has recommended against using that feature since it might be made invalid by future extensions to the language.)

An assigned variable may also be used as the file <name>. In that case, the logical name of the file is the value of the variable and not the name of the variable. There are times (opening multiple files in a loop, for example) when it is far more elegant to use a variable.

This will write a line to the file 't:vartest':

```
/**/
LFname = 'TFile'
   /* the variable's name can be written in any mixture of U&lc */
if open(LFName, 't:vartest', 'W') then
      /* 'TFile' is now the logical name of the file        */
   call writeln('TFile', 'See, it works with a variable.')
call close LFName
```

Next: NON-FILE DEVICES | Prev: Overview | Contents: File I/O

## 1.5 ARexxGuide | Functions reference | File I/O (3 of 5) | OTHER DEVICES

### Using I/O functions with other devices

The Amiga operating system makes the file I/O functions even more useful because it extends the concept of 'file' to cover a range of devices including text windows and printers. Because the OS is able to treat a printer as a file-like device, ARexx can send output to a printer using a simple variation of the file I/O functions: The device 'PRT:' may be specified as the file name in the OPEN() function:

```
/**/
if open(Printer, 'prt:', 'W') then do
   call writeln(Printer, 'Hello world')
end
```

(The READLN() input function cannot be used when communicating with the

```
PRT: printer device.)

Using the operating system's  console device , a window can be opened and
treated in much the same way as a disk file:

   /**/
   if open(OutWin, "con:8/8/272/88/Output Window", W) then do
      call writeln(OutWin, 'Hello there, you big bad world.')
      call delay 500
      call close OutWin
   end

Even the input functions READLN() and READCH() can be used with the
console device and will act much like the instruction  PARSE PULL  does on
the standard input window.

Next: STANDARD I/O FILES | Prev: Logical file name | Contents: File I/O
```

## 1.6 ARexxGuide | Functions reference | File I/O (4 of 5) | STANDARD I/O

```
                    Standard input/output files: STDOUT, STDIN, STDERR
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
The function  SHOW('F')  will return the names of all currently open
logical files. The logical name of any file added with OPEN() will appear
on the list. In virtually all cases, the returned list will also contain
the names of at least two files that were not explicitly opened in the
script: STDIN and STDOUT are logical files that are available by default
to all scripts. The names refer to the standard input and output devices.

The instructions  SAY  and  PARSE PULL  are closely related to the
functions
                  WRITELN()
                  and
                  READLN()
                 . SAY and PULL output and retrieve
items from a defined logical file, except that the file used by the
instructions need not be opened.

SAY outputs a specified string to STDOUT, making it a simpler variation of
the clause 'call writeln(STDOUT, <string>)'. In the same way, PULL
retrieves its input from the STDIN device much like 'Input =
readln(STDIN)'. The instruction  PARSE EXTERNAL  also retrieves output
from a logical file, one named  STDERR , that is normally available only
when the  trace console  is open.

The STDIN and STDOUT files can be redirected to other devices using a
standard AmigaDOS facility: When a command is followed by the character
'<', STDIN -- the standard input device -- is redirected to the device
specified after that character. Similarly, the '>' redirects standard
output or STDOUT to a specified device.

The interactive example uses the following simple script to demonstrate
the effect of redirection.

   /**/
```

```
   options prompt "0a"x||"Enter any text then press <Enter>: "
   pull T$
   say T$
```

```
              Interactive example:  Standard I/O demonstration        *
```

Redirection is often used on the Amiga to suppress output by setting up a dummy device called 'nil:' as the destination and source for a command. When the output of an ARexx program is redirected with the '>NIL:' option, the instruction SAY will have no effect. Its output will disappear. Similarly, the instruction PULL will return with an empty string when input is redirected to nil: with '<NIL:'.

```
   Also see:  SIGNAL ON SYNTAX

               Technique note:  Open custom console windows
                                Format() user function
```

Next: File I/O | Prev: Non-file devices | Contents: File I/O

## 1.7 ARexxGuide | Functions reference | File I/O (5 of 5) | REXX I/O

```
                Standard REXX I/O functions
~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Because of the unique characteristics of file I/O methods on different systems, implementations of REXX exhibit significant differences in file I/O methods. Despite the differences, most implementations other than ARexx use the following functions that are defined in  TRL2 :

```
 CHARIN(<stream>  [,<position> <,count>])  - read characters
 CHAROUT(<stream> [,<string> <,position> ] - write characters
 LINEIN(<stream> [,<line> <,count>])       - read a line
 LINEOUT(<stream> [,<string> <,line>])     - write a line
 STREAM(<stream> [,<option> <,command>])   - misc. stream operations
```

There is no OPEN() function in the standard because <stream> is taken to be actual name of the character stream (a filename, for instance). The STREAM() function is defined to allow for implementation-specific commands that can retrieve information about a stream or invoke system-specific commands for the stream. It is sometimes used to assign a logical name to a stream.

The standard functions listed above could be duplicated in ARexx within a subroutines that issued used the

```
                OPEN()
                function behind the scenes to
set up an ARexx
                file handle
               .
```

## 1.8 Using a `.' as the first character in a symbol

```
 Compatibility issues:
   The ANSI committee that is working on a standardized definition of REXX
   has recommended against use of "." as the first character in a non-
   numeric  symbol . Although ARexx may not be changed to meet the ANSI
   definitions, users might want to avoid use of this kind of construction
   to maintain greater compatibility with other versions of REXX.
```

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

```
                rv = CLOSE(<file>)
     rv is 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
                Compatibility issues:
   All file I/O function in ARexx are system-specific.

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

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

```
               rv = EOF(<file>)
     rv is a  Boolean value

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

   Also see
                READLN

                READCH

                SEEK
                  Example:
      do until EOF('AFile')
```

```
        ...
      end
                  Technique note:  Read one file, write to another
                                   Getting output from a command

 Compatibility issues:
   All file I/O function in ARexx are system-specific.

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

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

```
rv = LINES([{STDIN | STDOUT | STDERR}])
     rv is 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           instruction
             QUEUE          instruction
             PULL           instruction

                  Technique note:  Data scratchpad with PUSH & QUEUE

 Compatibility issues:
   As defined in  TRL2 , this function should return the number of lines
   remaining in any character input stream -- files as well as consoles. It
   has more limited utility in ARexx.

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

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

```
                rv = OPEN(<file>, <filespec>, [<option>])
     rv is 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.

More information:
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:

```
      /* create a new file                                          */
    if open('AFile', 't:Information.data', 'W') then ...
      /* 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 ...
      /* open an existing file for reading data                     */
    if open(.IFile, FileName, 'R') then ...
```

Also see

CLOSE

READLN

READCH

WRITELN

```
                WRITECH
                            SIGNAL ON IOERR

                Technique note:  Open custom console windows
                                 CountWords() user function
                                 Read single record in data file
                                 Output text to printer
                                 Read one file, write to another
                                 Using the clip list
                                 Data scratchpad with PUSH & QUEUE
                                 Get/set environmental variables
                                 Getting output from a command

 Compatibility issues:
   All file I/O function in ARexx are system-specific.

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

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

```
               rv = READCH(<file>, [<length>])
     rv is 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.

The function will read a maximum of 65535 characters from the file.
Specifying a longer <length> will not cause an error, but also will not
return more than the 65535 characters.

   Example:
      Chars = readch('AFile', 6)     /* will read the next 6 characters  */
      File  = readch('AFile', 65535) /* Will read entire file _or_ the   **
                                     ** first 64k bytes of it.          */

   Also see
                READLN

                WRITECH
                                Technique note:  Read single record from data  ↩
                                   file
                                 Open custom console windows
 Compatibility issues:
   All file I/O function in ARexx are system-specific.

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

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

```
                rv = READLN(<file>)
     rv is 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.

The maximum length of the value returned by READLN() is 1000 bytes. If
line-feed characters are not used in a file, then multiple calls to
READLN() would return the contents of the file in 1000-character chunks.

   Example:
      ThisLine = readln('MyFile')

   Also see
                READCH

                WRITELN
                                Technique note:  CountWords() user function
                                 Read one file, write to another
                                 Get/set environmental variables
                                 Getting output from a command

 Compatibility issues:
   All file I/O function in ARexx are system-specific.

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

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

```
                rv = SEEK(<file>, <offset>, [<anchor>])
     rv is 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:
      PartOfLine = readch('AFile', 6)

   Also see
                OPEN

                READCH

                READLN

                EOF
                                Technique note:  Read single record from data  ↩
                                   file

 Compatibility issues:
   All file I/O function in ARexx are system-specific.

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

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

```
               rv = WRITECH(<file>,<string>)
     rv is a  number

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.

The return value from the function is a count of the characters written to
the file. If the function was successful, the number returned will be
equal to the length of <string>. Any other return indicates failure.


   Also see
                WRITELN

                READLN

                SEEK
                                Technique note:  Output text to printer
```

Get/set environmental variables

Compatibility issues:
All file I/O function in ARexx are system-specific.

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

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

```
                rv = WRITELN(<file>,<string>)
     rv is a  number
```

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.

The return value from the function is a count of the characters written to the file. If the function was successful, the number will be one more than the length of <string> since the function counts the new-line character that it adds. Any other return indicates failure.

Example:
```
call writeln('AFile', 'This will be sent to file opened as AFile')
```

Also see
WRITECH

READLN

SEEK

EOF

Technique note: Read one file, write to another
Using the clip list

Compatibility issues:
All file I/O function in ARexx are system-specific.

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

```
rv = ADDRESS()
     rv is 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         instruction
             PARSE SOURCE    instruction
             Current 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

```
                rv = ADDLIB(<name>, <priority>, [<offset>], [<version>])
     rv is 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: "REXXSupport.Library" is not the same thing as "rexxsupport.library". Library names are usually written in lowercase, but there are some exceptions. Be careful to use the correct case; otherwise the desired functions will not be made available to ARexx. Be careful also to enclose the library name in quotation marks. ARexx converts symbols to uppercase if they are not quoted. Such a shift would cause the wrong name to be sent to the function.

Unless an explicit path is specified in the argument string, ARexx will look for the library when needed in the system libs: directory.

<priority> is an integer between -100 and 100. It may be chosen by the user and controls the order in which ARexx will search for a function-name match within the libraries. A library with a higher priority number will be searched before other libraries.

When a function is called and several entries are included on the Library List, ARexx passes the function name to each of the libraries in turn. A library will send a code back to ARexx indicating whether a match was made. This takes some time, so it may be desirable to assure that a frequently-used library is searched early.

If several libraries are added to the list with the same priority, ARexx will search them in the order they were added.

The <offset> number for a library must be specified by the library's developer. For rexxsupport.library and most of the other packages released so far, the number is -30.

The <version> number is often specified as 0, which tells ARexx to load any library with the specified name. If a minimum version number is required, then the integer part only of the verion may be specified in this argument. ARexx will not add the name to its list if the available library has a version number less than that specified here.

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, RXLIB, command, Library functions, 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.

 Compatibility issues:
   This function is system-specific to ARexx. Other implementations offer
   similar but differently-named functions to load external libraries.
    TRL2  does not define a standard function for the task.

   In OS/2 REXX the function RxFuncAdd() performs a similar task.

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

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

```
rv = ARG([<argnumber>], ['EXISTS' | 'OMITTED'])
     rv is 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 the argument was not supplied.

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

If the script was started as a command from the shell (usually with the
 RX  command), then all arguments are treated as a single string, even if
the string contains commas. Multiple argument strings are available only
for  subroutines  called as  internal functions  or scripts called as
 external functions .

   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      instruction

                Technique note:  CountChar() user function
                                 CountWords() user function
                                 Extract file name from full spec
                                 Get/set environmental variables
```

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

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

```
                rv = DATATYPE(<string>, [<type>])
     rv is 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
             ABS
             SIGN

                 SYMBOL
                                Technique note:  Checking for unique datatypes
                                 Format() user function
```

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
rv = DELAY(<number>)
     rv 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

 Compatibility issues:
   All support functions are system specific.
```

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

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

```
                rv = ERRORTEXT(<number>)
     rv is 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
                Next: DIGITS() | Prev: DELAY() | Contents: ARexx control func.
```

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

```
                rv = DIGITS()
     rv is a  number

The result is the current  NUMERIC  DIGITS setting.

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

   Also see
                 FORM

                 FUZZ
                             PARSE NUMERIC

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

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

```
                rv = FORM()
     rv is a  string

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

   Also see
                 DIGITS

                 FUZZ
                             PARSE NUMERIC

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

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

```
               rv = FUZZ()
     rv is a  number

The result is the current  NUMERIC  FUZZ setting.

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

   Also see
                DIGITS

                FORM
                            PARSE FUZZ

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

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

```
               rv = GETCLIP(<clipname>)
     rv is a  string

Returns the value associated with clip <clipname>. The search for the
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);                     >>>

   Also see
                SETCLIP
                               Technique note:  Using the clip list

 Compatibility issues:
   This function is an ARexx extension that is not supported and not
   duplicated in the standard language definition.

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

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

```
               rv = PRAGMA(<option> [,<value>])
     rv is a  string
```

```
        or a  Boolean value

Changes, or returns information about aspects of the system environment.
The
                 <option>
                 argument specifies the environmental attribute. A specific
<value> is expected for each type of <option>.

   Also see  SHOWLIST

 Compatibility issues:
   This function is an ARexx extension that is not supported and not
   duplicated in the standard language definition.

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(). Only the first letter of the option is needed.

| 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.)<br><br>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 script 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.<br><br>If <number> is omitted, the function returns the current priority setting.<br><br>If <number> is included, the priority will be changed to that value. The number of the previous priority will be returned.<br><br><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 script 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, the 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

```
                rv = REMLIB(<libname>)
     rv is a  Boolean value

Removes <libname> -- the name of a library or function host -- from the
list maintained by the resident process. The library is not actually
removed from memory, but will no longer be available to ARexx scripts and
may be purged by the system when it needs the memory.

The function is useful when 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 that is present from being
found. This function will remove the name from the list.

   Also see
                ADDLIB
                Compatibility issues:
   This function is an ARexx extension that is not supported and not
   duplicated in the standard language definition.

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

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

```
                rv = SETCLIP(<clipname>, [<value>])()
     rv is a  Boolean value

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

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


   Also see
                GETCLIP
                            RXSET          command

               Technique note:  Using the clip list

 Compatibility issues:
   This function is an ARexx extension that is not supported and not
   duplicated in the standard language definition.

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

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

```
                rv = SOURCELINE([<line number>])
     rv is 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 */

                  Technique note: Using in-line data

   Also see
                ERRORTEXT
                            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

```
               rv = SYMBOL(<name>)
     rv is '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
                            ABS
```

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

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

```
rv = TRACE([<option>])
     rv is a  string
```

Returns a string that indicates the tracing option in effect when the function was called. If <option> is used to set the tracing mode in the same way as the TRACE instruction.

The <option> argument can 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. 'N' is returned if the default tracing mode was used.

Unlike the trace instruction, this function will alter the trace mode from within a program even if interactive tracing was started with the TS command.

The '?' and '!' characters can 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

               Interactive example:  Experiment with trace options     *

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

```
         say trace(off)          >>> N

   Also see  Error codes

   Tutorial  Debugging a script

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

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

```
rv = VALUE(<name>)
     rv is 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      Instruction

                 Technique note:  Interpreted variable names
                                  Get/set environmental variables

 Compatibility issues:
   Two additional arguments not supported in ARexx are defined in  TRL2 .
   The syntax of the standard function is:

               VALUE(<name>, [<newvalue>], [<selector>] )

   If <newvalue> is specified, then that value is assigned to the variable
   represented by the <name> expression. The third argument allows a
   REXX program to access a variable from the environment specified by
   <selector>.

   In OS/2, for instance, <selector> can be used to set or retrieve the
   value of an environmental variable:
```

```
                PathVar = value("MyPath",,"OS2ENVIRONMENT")
```

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