ARx_Cmd.ag

COLLABORATORS

	TITLE : ARx_Cmd.ag	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		October 17, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

ARx_Cmd.ag

1.1 ARexxGuide | Command utilities

```
                  AN AMIGAGUIDE® TO ARexx                  Second edition (v2.0)
by Robin Evans

ARexx command reference:

               About this section
                 AmigaDOS command programs:

               RexxMast

               RXC

               RX

               HI

               RXLIB

               RXSET

               TCO

               TCC

               TS

               TE

               WaitForPort
                 Useful tools:

               WShell

               ExecIO
```


This guide is shareware . If you find it useful, please register.

1.2 ARexxGuide | Command utilities | ABOUT THIS SECTION

```
   ARexx command utilities
   ~~~~~~~~~~~~~~~~~~~~~~~
```
The distribution copies of ARexx include several utility programs. These are similar to other command utilities located in the C: directory. They are called in the same way, by entering the command name on the shell or using the 'Workbench/Execute command' menu item.

On a standard Workbench disk, the ARexx utility programs are located in a directory named 'rexxc'. There's nothing special about that directory name. The program files can be moved to whichever directory the user might choose, but wherever they are located, the directory should be included in the command path so that the command will be executed immediately.

The command path is controlled with the AmigaDOS command 'PATH'. Consult your AmigaDOS manual for more information if you are not familiar with its use.

The main difference between these commands and other command programs located in C: is that the ARexx command utilities will not supply help information when <command> ? is entered on the shell.

Next, Prev & Contents: Command utilities

1.3 ARexxGuide | Command utilities (1 of 11) | RexxMast

```
                RexxMast
```

Launches the ARexx resident process. The RexxMast program (which is located in the sys:System drawer of standard workbench disks) is the interpreter which must be running on the system before an ARexx script can be executed.

```
WAYS TO START REXXMAST
~~~~~~~~~~~~~~~~~~~~~~
```
To make ARexx available whenever needed, RexxMast should be started with a command in one of the files that are automatically run when the computer boots up. To do that, the icon for RexxMast can be placed in the WBStartup drawer of the Workbench or the following line can be added to the file s:user-startup.

```
   rexxmast >nil:
```

It may be necessary to include the full path ('sys:system/rexxmast', for example) if the drawer in which the program is located has not been added to the system's command search path.

If it is not started as part of the startup sequence, RexxMast can be started by double-clicking on its icon located in the sys:system directory.

A third alternative is to launch a script using the
RX
utility. That
command will start RexxMast if it is not available.

However it is started, RexxMast will continue to run in the background until it is specifically halted with the
RXC
command.

1.4 ARexxGuide | Command utilities (2 of 11) | RXC -- Rexx Close

RXC

Closes the ARexx resident process. The REXX port is withdrawn immediately upon execution of this command which means that no new ARexx programs may be started.

The command will not halt the interpretation of currently executing programs, however. They will continue as they normally would. The resident process (which is started with
RexxMast
) will exit as soon as all ARexx
tasks that were executing when the command was issued have finished.

Once the RXC
Next: RX | Prev: Command utilities | Contents: Command utilities

1.5 ARexxGuide | Command utilities (3 of 11) | RX

RX <program> [<arguments>]

Launches an ARexx <program> from the AmigaDOS shell, or can be used as the default tool for an icon that will launch the program.

Whether it is entered on the command line or used as the tool of an icon, the RX utility will start
RexxMast
, the interpreter, if it is not
already available.

ARexx search path:
~~~~~~~~~~~~~~~~~~
If <program> includes an explicit path specification, then that path alone is searched. If an extension (like .ed) is included, then that extension alone is searched for.

If a path specification is not included, ARexx will first search the
~~~~~~~~~~~~~~~~~~

current directory for <program> and then the REXX: directory. If an extension is not included, it will first look in each directory for the exact name as entered and then for a the specified name with the default extension added.

If the script is launched from a shell or from a workbench icon, the default extension will be .rexx. If it is launched as a macro from an application, the default extension will be something determined by the developer of that application. The extension used for macros in the Ed text editor, for instance, is .ed.

The PARSE SOURCE instruction can be used to determine the default extension for the current environment. The current default extension may be excluded from the name of script when it is invoked.

This search method is used both for commands -- scripts launched with the RX command -- and for external functions .

The following list shows the programs that would be launched if the current directory is set to the value to the left of the '>' sign:

Current dir	Command	Scripts that would be launched
Work:>	rx Foobar	1. Work:Foobar
		2. Work:Foobar.rexx
		3. rexx:Foobar
		4 rexx:Foobar.rexx
Work:>	rx scp/Foobar	1. Work:scp/Foobar
		2. Work:scp/Foobar.rexx
Work:scp>	rx Foobar.rexx	1. Work:scp/Foobar.rexx
		2. rexx:Foobar.rexx
Work:>	rx Foobar.oxt	1. Work:Foobar.oxt
		2. rexx:Foobar.oxt

Next: HI | Prev: RXC | Contents: Command utilities

1.6 ARexxGuide | Command utilities (4 of 11) | HI -- Halt Interpretation

HI

Halts interpretation of all currently executing ARexx programs.

If a program has its halt trap enabled (with SIGNAL ON HALT), then the appropriate subroutine will be called for the program. Otherwise the program will terminate at the end of the currently executing clause. Because it waits until the end of the current clause to stop a program, the command will not immediately halt a program that is waiting for input with PULL , WAITPKT() , or similar functions, instructions, or commands; but will stop it only after the input request has been satisfied.

Next: RXLIB | Prev: RX | Contents: Command utilities

1.7 ARexxGuide | Command utilities (5 of 11) | RXLIB

```
RXLIB [<name> <priority> [offset, version]]

This command performs the same function as the built-in ARexx function
 ADDLIB() , with one extension.

The <name> argument is case sensitive, but -- unlike its use in the
ADDLIB() function -- <name> should not be quoted when used with this DOS
command since the parsing of commands is done according to AmigaDOS
conventions, which would cause the opening quote to become part of the name
of the library. Names entered on the command line are not translated to
uppercase.

If a library is specified, it should be located in the LIBS: directory. If
a function host is specified, then <name>, which is still case-sensitive,
should refer to the public message port opened by the host.

<priority> is an integer between -100 and 100. It sets the search
priority used by the resident process in case of duplicate function names
in the Library List. A library with a higher priority is searched before
others. If the priority is the same, then libraries are searched in the
order they were added to the list.

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

Used without arguments, RXLIB will list the currently loaded libraries and
function hosts. Names are ordered on the list by the search priority
assigned to them.

   Examples:
         RXLIB rexxsupport.library     0 -30 0
         RXLIB rexxreqtools.library    0 -30 37
         RXLIB RexxDosSupport.library  0 -30 0
         RXLIB rexxarplib.library      0 -30 0
         RXLIB rexxmathlib.library     0 -30 0
         RXLIB QuickSortPort
         RXLIB

   The last command might output:
         rexxsupport.library (library)
         rexxreqtools.library (library)
         RexxDosSupport.library (library)
         rexxarplib.library (library)
         rexxmathlib.library (library)
         QuickSortPort (host)
         REXX (host)
```

Next: RXSET | Prev: HI | Contents: Command utilities

1.8 ARexxGuide | Command utilities (6 of 11) | RXSET

```
RXSET [<name> [[=] <value>]]

This command performs the same task as the built-in ARexx function
 SETCLIP() , with one extension.

<name> may not include spaces and should not be enclosed in quotation
marks. The clip will be set using the exact case-sensitive characters used
as the first argument to the command.

When both <name> and <value> are specified, a clip of that case-sensitive
<name> will be created on the  clip-list  with <value> as its content.

If <name> is used without <value>, then any clip matching that <name> will
be removed from the list.

<value> can be any string that can be typed on the shell. The string can
be enclosed within double-quotation marks { " }, but does not have to be.
Since AmigaDOS parsing conventions are used with this command, single
quotation marks are not recognized as special characters and will be
included as part of the clip.

Entered without arguments, RXSET will list all currently defined clips
along with their values, but it will truncate the the values to 65
characters. The truncation happens on the display only. Strings on the
clip list can be up to 65535 characters in length.

Next: TCO | Prev: RXLIB | Contents: Command utilities
```

1.9 ARexxGuide | Command utilities (7 of 11) | TCO -- Trace Console Open

```
                TCO

Opens the global tracing console -- a standard console window that can be
moved and resized by the user and will have all of the standard input
controls of the user-shell specified for the system.

Trace output generated by currently active ARexx scripts will be diverted
immediately to the tracing console.

The tracing output itself is controlled by the  TRACE  instruction, the
 TRACE()  function, or by the
                 TS
                 and
                 TE
                 commands.

If a trace request is issued when the trace console is not open, ARexx
will attempt to output the trace to a shell. Without the trace console,
however, ARexx will often have no place to send trace output from a macro
that was started within an application program.

The trace console is opened as the logical device  STDERR . An ARexx
```

script can send output to the window with this instruction:

```
   CALL  WRITELN  STDERR, <output>
```

The same instruction can be used at any interactive trace prompt to view the results of an expression on the trace console.

The instruction 'PARSE EXTERNAL <template>' will retrieve user input from the trace console (or anything else defined as STDERR) rather than the shell.

Also see Error codes

Tutorial Debugging a script

The output of a program trace is far more useful if a shell like WShell is used since the window's scroll bars and scrolling keys make it easy to review the entire history of the trace.

Next: TCC | Prev: RXSET | Contents: Command utilities

1.10 ARexxGuide | Command utilities (8 of 11) | TCC -- Trace Console Close

TCC

Issues a close request to the global tracing console which is opened with the TCO command. The window will actually close only after all currently active tracing requests have been satisfied.

Next: TS | Prev: TCO | Contents: Command utilities

1.11 ARexxGuide | Command utilities (9 of 11) | TS -- Trace Start

TS

Starts an interactive trace of all currently executing ARexx programs. The tracing mode is the same as that set when the instruction 'TRACE ?R ' is entered in a program.

If the trace console has been opened with the TCO command, the output of the trace(s) will be diverted to that window.

This command is one way to gain control of a program that is performing in an unexpected way -- one caught in an endless loop, for instance. Program

statments can be entered at any pause point to change the values of
variables, or even to halt just one of several programs by entering
Control-C at a pause point of the program to be halted. Other programs
will continue.

The instruction 'TRACE OFF' entered at any pause point will stop the trace
for the script or subroutine being traced at that point. The TRACE()
function (with the 'Background' or 'Off' option) can be used within the
program code to suppress tracing of a script even when the global tracing
flag is set.

The command utility
TE
will remove the tracing flag set by TS.

Also see Error codes

Tutorial: Debugging a script

Next: TE | Prev: TCC | Contents: Command utilities

1.12 ARexxGuide | Command utilities (10 of 11) | TE -- Trace End

TE

Withdraws the tracing flag set by the
TS
command. A trace called by the
TS command will end after the current pause point in the program is
completed.

The TE command will not affect tracing that is called by the TRACE
instruction or the TRACE() function entered as part of the program code.

The command can be entered in the
trace console
at any '>+>' prompt. If
the clause 'address command TE' is entered, the command will work in any
environment. One additional prompt is usually presented before the TE
command takes effect.

Next: WAITFORPORT | Prev: TS | Contents: Command utilities

1.13 ARexxGuide | Command utilities (11 of 11) | WaitForPort

WaitForPort <port>

Pauses 10 seconds or until the named <port> is available on the system. If
the port is not available after 10 seconds, the command will set a return
code (RC) of 5.

Applications that support ARexx open a port with a name specified by the application's developer. The port name should be detailed in the documentation. Port names are case sensitive, so that 'TurboText1' is not the same name as 'TURBOTEXT1'. WaitForPort lets a script pause while a program is loaded by the system since the port will not be added until the application is ready to receive input.

The following fragment shows how the command is used to pause while the terminal program VLT is loaded:

```
/**/
   /*  SHOW()  can be used to check for the existence of a port    */
if ~show('P', 'VLT') then do
   address command    /* both RUN and WAITFORPORT are DOS commands.*/
   'run >nil: vlt:VLT'
      /*
         The loop below assures that the program will wait at least 50
         seconds for the port to open, but will exit as soon as the port
         is available.
      */
   do 5 while ~show('P', 'VLT')
      'waitforport VLT'
   end
   address           /*  toggle  back to previous address           */
      /* The return code RC is set to 5 when WaitForPort times out  **
      ** before the port is available.                               */
   if rc = 5 then do
      say 'Unable to load VLT'
      exit
   end
end
```

WaitForPort is not an ARexx command or function. It is a DOS command much like 'list' or 'cd' and should be treated the same way in scripts. A common error is to issue the command to the wrong environment. The instruction 'ADDRESS COMMAND ' may have to be issued to assure that the command is sent to AmigaDOS.

Next: Command utilities | Prev: TE | Contents: Command utilities

1.14 ARexxGuide | Command utilities | Tools (1 of 2) | WShell

```
                WShell replacement shell
~~~~~~~~~~~~~~~~~~~~~~~~
```

WShell is a commercial program. It is not included with either the standard OS or the commercial version of ARexx. It has a place here, however, because it is a program specifically designed to make use of ARexx on the Amiga easier for both programmer and user.

WShell was written and is supported by Bill Hawes (Wishful Thinking Development Co.) who also wrote ARexx. The close relationship shows. The

```
RX
```

command utility is unneeded on WShell since the shell will launch

ARexx programs transparently, by just entering the program name (without the .rexx extension) on the shell. There is no need to set the script bit or anything else -- just type in the name and the script will run.

Scripts launched with WShell inherit an address that is significantly more useful than the REXX address handed to scripts launched by
RX
. That
address is named WSH_# where # is the task number of the shell. The WSH addresses act like a combination of the COMMAND port and the REXX port. Within a script launched from WShell, it is rarely necessary to issue the 'ADDRESS COMMAND' instruction since the default address handles commands.

WShell windows include a scroll-bar and definable scrolling keys that make reviewing the history of a traced program far easier.

The shell includes a number of other enhancements, including shell menus, an extraordinarily powerful and configurable filename completion feature, icon support, and more. It worthwhile for any shell user, but those who use ARexx extensively should give it special consideration.

Next: EXECIO | Prev: Command utilities | Contents: Command utilities

1.15 ARexxGuide | Command utilities | Tools (2 of 2) | EXECIO

ExecIO
~~~~~~
A utility that, among other things, will redirect the output from OS commands like 'list' into the variable environment of an ARexx script. It is not included with the OS releases or with the commercial version of ARexx, but is included with
WShell
, which is required for its use.

The following simple example will transfer to the stem variable [ArxFN.] the names of all files in the current directory that match the variable [FilePat]. When the command returns, the variable [ArxFN.0] will hold the number of files <n>; the variables [ArxFN.1] to [ArxFN.<n>] will hold the actual file names.

```
'list quick nohead' FilePat ' | ExecIO stem ArxFN.'
```

That's often useful but could easily be duplicated by other means. (The function library rexxarplib, for instance, includes a function that will do the same thing.) Some of the more arcane abilities of ExecIO are more difficult to duplicate. The following command will read lines from the file [FileName] and will assign to numbered branches of the stem [Nodes.] only those lines that contain, in columns 1 to 5, the (non case-sensitive) characters '@node'.

```
'ExecIO read' FileName 'stem Nodes. locate "@node" colend 5'
```

In the following complex command (the one-line command is spread over two lines), ExecIO is used twice, first (at the end of the command) to
~~~~~~

transfer to the variables [FileCmd.] a list of ExecIO read commands for each of the files matching [FilePat]. The 'lformat' option to 'list' is used to prepare a command matching that in the example above for each of the matching files:

```
'list quick nohead' FilePat 'lformat "ExecIO read %s stem Nodes.
                     locate *"@node*" colend 5" | ExecIO stem FileCmd.'
```

ExecIO history

ExecIO is also useful for those who wish to investigate the variety of REXX scripts written for other systems: A command of that name is used widely in REXX programs written for the system on which the language was born -- the CMS environment for IBM'S VM mainframe computers.

ExecIO is an OS command in CMS. It was (and is) used in REXX scripts on that system because the original releases of the language did not include other facilities for file I/O. (The file I/O functions used in ARexx are system-specific extensions to the language.) Later versions of REXX patched that hole in the language, (with instructions and functions different than those used in ARexx, unfortunately), but not soon enough to prevent ExecIO from gaining an established place in the language.

Most of the other computer systems to which REXX has been ported include some version of ExecIO to maintain greater compatibility with the thousands of scripts written for CMS. Thanks to Bill Hawes and WShell, the Amiga is not an exception.

Next: Command utilities | Prev: WShell | Contents: Command utilities