ARx_UcrTutor2.ag

COLLABORATORS			
	TITLE : ARx_UcrTutor2.ag		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 8, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

ARx_UcrTutor2.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 SetDest:

Return to program listing

SetDest is called an internal function because it is defined within the program from which it is called. The colon after 'SetDest:' identifies it as a label . It tells ARexx that the following program lines are part of a function.

In this case, all of the variables that were declared in the main program section will be available within the function definition. They could even be changed in the function. That behavior can be avoided by using the keyword PROCEDURE immediately following the function label.

The keyword RETURN at the end of the listing indicates the end of the function and returns control to the place in the main code of the program from which the function was called.

1.3 parse arg AProg

Return to program listing

The unabbreviated form of the ARG instruction which was used at the beginning of the program is used here. Adding PARSE prevents the received value from being translated to upper case. That may be important here because the case of the letters used in an option string is significant to some of archive programs.

The instruction picks up the argument that was included within the parentheses when the function was called.

1.4 cdir = pragma('D') /* Store the current directory */

Return to program listing

PRAGMA() returns information about the system environment in which an ARexx script is executing and allows the script to change some aspects of that environment. Here it is used for purely informational purposes; it retrieves the name of the current directory.

1.5 /* Change archive filename...

Return to program listing

This comment partly explains the purpose of the code within the SELECT block that follows.

The code is not absolutely necessary, but is included because it makes the script friendlier to the user by allowing any valid file specification to be used, even something like { :dl/flooey.lzh } or { //flooey.lzh }.

It is useful because we might be changing directories later in the script to a volume different than the one where [FileName] is located. In order to properly identify [FileName] from the changed directory, we'll need a complete path specification for the file.

1.6 prg63

Return to program listing

The variable [CDir] contains the name of the current directory. That value will be altered later in the program, but we'll have to remember its original value to restore the system to its original state. [FDir] can now be changed without affecting [CDir].

1.7 when pos(':', FileName) > 1 then

Return to program listing

This is the first of four conditionals (three WHENs and the OTHERWISE instruction). Using the built-in function POS() , it checks for a [FileName] that contains ':' somewhere other than the first character: { foo:dl/flooey.lha }. That positioning of ':' means that a volume designation is included in [FileName], so it need not be modified.

1.8 when left(FileName, 1) == '/' then

Return to program listing

This checks for a relative file specification like { //flooey.lzh } in which [FileName] is specified relative to the current directory without including the name of the directory.

Notice that the comparative operator '==' is used here rather than '='. Either one would work since a single character is returned by the function, but the sign for exact equality makes it clear that the comparison must be exact -- without extra spaces.

1.9 do while left(FileName, 1) == '/' /* is 1st char '/'? */

Return to program listing

We've used a DO block several times to group a block of instructions together under an IF clause. This one does the same thing but it also introduces one of several iteration specifiers that can be used to make a program loop through a block of instructions repeatedly.

Like IF and WHEN , the WHILE subkeyword must be followed by an expression that results in a Boolean value . The expression is evaluated before any clauses in the block are executed. As long as the expression returns a TRUE value, the following block will be executed.

In this case, the LEFT() function pulls off the first character in [FileName], which is then compared to '/'.

If the value of the expression is FALSE, (which will happen when there is no longer a '/' character at the beginning of [FileName]) the block between this line and the matched END will be skipped and program execution will continue with the next line.

1.10 FileName = substr(FileName, 2) /* strip it off */

Return to program listing

As it's used here, with only the first and second arguments specified, the SUBSTR() function returns the portion of the string [FileName] beginning at the second character; in other words, the first character of the value is stripped from [FileName]. { //flooey.lha } would become { /flooey.lha }.

1.11 DivPos = max(lastpos('/', FDir,Pln),,

Return to program listing

This is the first part of one clause divided over two lines. The extra comma at the end of this line tells ARexx to combine this line with the next one and treat them both as a single line.

The technique used here for locating a filename and path is explained in a note to the LASTPOS() function. The nested functions tell us where the final directory or volume divider character is located in [FDir].

The variable [Pln] was defined on the previous line. It is a number one less than the length of [FDir] and is used to start the search for the '/' or ':' one character from the end of the string, resulting in a path that includes either ':' or '/' as the final character. That path will be used when we combine [FDir] with [FileName] to create a fully qualified file name.

1.12 FDir = left(FDir, DivPos)

Return to program listing

The value of [DivPos] tells us the length of the string that contains the file path to the current definition of [FileName]. The LEFT() function will truncate [FDir] to just that length.

This loop is performing much the same task as entering { cd / } on the CLI: it steps down one level in the file path each time this clause is called within the loop.

1.13 when left(FileName,1) = ':' then

Return to program listing

This conditional checks for a relative file name that refers to the root directory in its specification -- something like { :flooey.lha }

1.14 parse var CDir FDir ':' .

Return to program listing

PARSE is a powerful variation of the assignment clause , which we've used several times already -- powerful because it is able to assign values to several variables at once and to perform, in one step, the tasks of functions like LEFT() , RIGHT() , and SUBSTR() as well as the location functions like POS() .

We've used variations of the instruction at three other spots in this script. Twice it was used to retrieve an argument, and once to pull a value from an interactive prompt presented to the user.

This time, it is used to break down a variable. The subkeyword VAR indicates that the symbol [CDir] is a variable that will supply the source string for the instruction. The following symbols -- (FDir ':' .) -- form a template that guides PARSE in splitting the input value into its parts.

The ':' in the template used here is enclosed in quotation marks, which identify it as a pattern marker . PARSE will look for that character within the string [CDir] and assign all characters to the left of it to the variable [FDir]. All characters to the right of the first ':' will be assigned to [.]. That dot is a placeholder token . It works like a variable would in that position, except that the value that would have been assigned to the variable is thrown away. We use it here because we don't care about characters to the right of the ':'.

This same operation could have been performed using the following functions:

```
FDir = left(CDir, pos(':', CDir) - 1)
```

1.15 if verify(right(FDir, 1), '/:', 'M') ~= 0 then

Return to program listing

What's left? Only a filename entered without a path -- one for a file located in the current directory.

The VERIFY() function will return 0 if neither of the characters '/' or ':' is used at the end of the path specification. If the value is not 0 (the comparison operator meaning 'not equal' is used here), then a '/' character is added in the next line.

1.16 FileName = FDir||FileName

Return to program listing

The final step in the process is to combine the path determined by the routine above with [FileName]. This gives us a fully qualified file name

that will be understood by AmigaDOS no matter what the current directory may be.

1.17 /* Perform the uncrunching */

Return to program listing

Now that we're sure we have a good file name, we'll perform the work that we really came here for: uncrunch a file. The following three commands will be sent to AmigaDOS.

1.18 'cd \

Return to program listing

Since this function is used for programs like ARC and ZOO that output unarchived files to the current directory, the first task is to move to the uncrunch directory defined in the main program. The { CD } statements are surrounded by quotation marks. That tells ARexx to treat them as commands to be run by the host -- AmigaDOS in our case. These commands will have the same effect as they would have if they were typed directly on the CLI.

Notice the odd collection of quotation marks. The single quotation (') marks are there for ARexx and will be removed before the command is sent to the host, but the double marks (") are there for AmigaDOS (in case one of the directory specifications used here contains a space or '?' mark) and will be sent on by ARexx because they are enclosed within the single-marks.

Both { CD } statements include ARexx variables outside the quotation marks. The variables will be expanded by ARexx before the command is sent to AmigaDOS.

1.19 "AProg '\

Return to program listing

Here, finally, is the heart of the matter -- another command that will be sent to AmigaDOS. Notice though, that the command is a variable this time. Both variables in this clause will be expanded by ARexx before they are sent on to the host . The resulting command will be something like

```
Arc x "work:foo/flooey.arc"
```

It will be executed by AmigaDOS just as would be if it had been typed on the CLI.

The doubled quote marks that begin the line { ''AProg ...} are a useful way to indicate that the clause is a command. The quotes form a null string which tells ARexx that the clause is neither an instruction nor an assignment .

Note: the '\' characters that may appear in the titlebar when viewing this listing are an odd effect of AmigaGuide and not a part of the program line.

1.20 return

Return to program listing

Program control is RETURNed to the line after the place where SetDest: was called.

1.21 if arg(1, 'E') then

Return to program listing

The ARG() function may be used instead of the instructions ARG or PARSE ARG . It offers some options not available with its instruction cousin. In this line, the 'Exists' option is used to check for the existence of a string in the first argument slot.

1.22 say arg(1)

Return to program listing

The SAY command prints out a message on the currently active shell. It is used in this subroutine to print a more complete explanation of how the UnCrunch program is used.

In this line, the instruction prints out the string that was sent as an argument to the function. That string is retrieved using the ARG() function.

Once the information is displayed, the program exits with an (arbitrary) error code of 20. (You could use any number you wish in this spot. 20, though, is the error number supplied by AmigaDOS in similar situations.)