11/30/94

Quick Remedial REXX

The REXX language manages strings of characters or words. REXX can compute numbers, but that is not really its strong point. In normal use, it constructs commands to be executed by the OS/2 system prompt or by the editor. Under SLIP, REXX is used to construct strings containing modem commands and responses needed by the communications controller, and to analyze the response that comes back from those strings.

A string in REXX is a set of characters between quotes. They can be double or single quote marks, the difference is a matter of taste. Anything outside the quoted string is either a variable name or part of the REXX language. Unlike C, REXX ignores the difference between upper and lower case except within quoted strings.

```
say 'Configuring local address =' os2_address ', Annex =' annex_address
```

SAY is the REXX language statement that writes to the screen (comparable to PRINT in BASIC, WRITELN in PASCAL, ECHO in BAT commands). The rest of the statement generates the text of the message. There are two character strings 'Configuring local address =' and ',Annex='. There are two variable names OS2_ADDRESS and ANNEX_ADDRESS. The meaning of this expression must now be explained.

Each REXX variable holds a character, string of characters, word, or fragment of a command. This value is stored in the variable explicitly by assignment, or by extracting one of the arguments passed to the program or some text read in from a file or, in this case, from the modem.

```
DIALCMD="ATDT"
PREFIX="9"
NUMBER="4329900"
```

These are assignment statements that put particular text strings into three named variables. DIALCMD contains the four character string "ATDT" which, of course, is the beginning of the string sent to a modem to dial the phone. There are two ways to combine variables to create complete commands. To combine the strings of two variables directly, write an expression where variable names are separated by two consecutive vertical bar characters ("||"), the REXX operation for concatenation.

```
DIALYALE=DIALCMD||PREFIX||NUMBER
```

This expression puts the string "ATDT94329900" in the variable DIALYALE. When building OS/2 or editor commands, however, it is often convenient to separate elements by a blank. REXX supports this when an expression includes variable names simply separated by a blank.

```
say 'Configuring local address =' os2_address ', Annex =' annex_address
```

Getting back to this statement, the value of the variable OS2_ADDRESS is inserted between the two strings. A blank is added after the '=' that ends the first string and before the ',' that begins the second string. After the '=' that ends the second string, a blank is inserted and then the value of the variable ANNEX_ADDRESS.

People who program in other languages would find this approach normal in expressions that format output, but rather strange in an assignment statement. The casual response is that REXX is mostly designed to format output. However, to make everyone happy, it is possible to restate this concept in

formal terms. *When variables or strings appear one after another, separated only by blanks, then there is an implied operation between them called BLANK. The BLANK operation concatenates the value of the first variable to a single blank character and then to the value of the second variable.*

REXX is equally casual in its handling of input. The PARSE statement breaks strings of characters up into smaller sections separated by special strings or by blanks. The full syntax of PARSE is horribly complicated and cannot be handled here. Fortunately, you only need a small subset of it.

```
parse arg interface , dialcmd username password
```

Unlike other languages, REXX doesn't automatically assign its parameters to variables. You have to call the PARSE statement to decode them. This statement appears at the beginning (except for comments) of the ANNEX script. It decodes the two arguments passed to the ANNEX script and stores them in variables. The comma indicates the break between the two arguments. The first argument will be stored in variable INTERFACE. The second argument, however, is a string that contains three words. PARSE breaks the string down so that the first word goes in DIALCMD, the second in USERNAME, and the last word (and, incidentally, any trailing trash left over) goes in PASSWORD.

The PARSE statement takes whatever characters are in the source and puts them into the supplied variables. Unfortunately, it has to account for all of the original characters. If some extra characters appear at the end of the source, they end up getting added to the end of the last variable. To get rid of this, there is a special trash collector where a period appears where a variable name might be:

```
parse arg interface , dialcmd username password .
```

This version of the statement adds an isolated period after the variable PASSWORD. Since a blank also appears between them, this causes PARSE to put the first word of the source in DIALCMD, the second in USERNAME, the third in PASSWORD, and now everything after the third word matches the dummy period and is thrown away. Without the period, the third word, forth word, and in fact anything else would have been assigned to the poor variable PASSWORD because PARSE would have no place else to dump them.

In a real connection script used by Warp, the first argument is generated internally by SLIP and contains the internal TCP/IP identifier for the SLIP link, the name "SL0". The three words, however, were configured by you when you filled in the Login Script field of the first Dial Other Internet Provider panel in the example above. If the field contained the value:

```
ANNEX.CMD ATDT4329900 GILBERT SOMETHING
```

then the DIALCMD variable gets "ATDT4329900", the USERNAME variable gets "GILBERT" and the PASSWORD variable gets "SOMETHING". Of course, since you configure the string you can choose not to type any stray chracters at the end and avoid the need for a period to collect the trash. However, later on when the script parses output from the communications controller we may need that extra period.

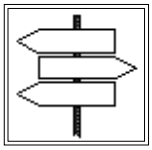
Normally, the REXX program has one statement on each line. In this case there is no need for special punctuation. However, a few lines of the ANNEX script have two separate REXX program statements, which requires that they be separated by a semicolon (';').

The CALL statement is used in REXX to call a subroutine. The word CALL is followed by the name of the subroutine and then by any arguments. The ANNEX and similar sample scripts have a set of subroutines to handle interaction with the COM port. The WAITFOR subroutine reads characters from the COM port until a certain trigger string has been read. When the trigger string arrives, the WAITFOR ends. All the characters that came in before the trigger are stored in a variable named WAITFOR_BUFFER. The SEND subroutine sends a character string out to the modem. The FLUSH_RECEIVE subroutine takes the contents of the WAITFOR_BUFFER and writes it out so the user can see the dialog between the communications controller and the script.

```
parse var waitfor_buffer  a '.' b '.' c '.' d '.' .
```

This is probably the worst statement in the entire ANNEX script to have to explain to a casual user. The WAITFOR statement has hit a trigger. All the output from the communications control up to the trigger is in the variable WAITFOR_BUFFER. This script expects that the buffer contains an IP address in the usual "130.132.57.21" notation. It breaks the address up into its four numbered components. When a string (even a one character string such as '.') appears in the list of variables to be recognized, that string represents a delimiter in the original data. The overall meaning of this statement is "Take anything in front of the first period ('.') and put it in variable A. Then take everything up to the next period and put it in B. Then everything to the third period goes in C, and everything up to the last period goes in D. The last element of the line is a period not in quotes, so it is a trash collector. It says, "Take everything in the original source after the forth period and discard it."


Return to Adapting a REXX Login Script


Back to Warp Table of Contents


Return to PC Lube and Tune