Script Reference

Print

Introduction

What Is a Script

Running a Script

Your First Script: What Is Your Name

Data Types

Integer, String, Character, Boolean, Date, Time

Variables

Variable Identifier, Variable Declarations, Predefined Variables

Expressions

Rule Of Precedence, Arithmetic Operators, Boolean Operators, Relational Operators

Statements

What Is a Statement

Comment, Assignment

If...Else...ElseIf...EndIf, Switch...EndSwitch

While...EndWhile Loop, Repeat...Until Loop

Break Statement

Stop Statement

#include Directive

Procedures

What Is a Procedure

Procedure Declaration, Parameter Declaration

Calling Procedure and Parameter Passing

Nested Procedures and Scope of Variables

Return Statement

Built-in Procedures by Category

COM I/O, Console I/O, File Handling, String Handling, Others

Built-in Procedures by Alphabet

Built-in Procedure by Alphabet

Print

Append	Opens a text file and moves the file pointer to the end.
Atoi	Converts a string to an integer.
ChDir	Changes current directory.
Close	Closes a file.
Concat	Appends one string to another.
Create	Creates a new text file or rewrites an existing one.
Date	Gets system date.
Delay	Suspends script execution for an interval.
Delete	Deletes one or more files.
FileExist	Determines if a file or directory is exist.
FileSize	Gets file size in bytes.
Get	Gets a string from remote system.
GetCh	Gets a character from remote system.
HangUp	HangUps the modem.
Image	Captures the terminal screen into an image file.
Input	Reads a string from the keyboard.
InputCh	Reads a character from the keyboard.
Itoa	Converts an integer to a string.
Length	Calculates the length of a string.
LogOff	Closes the log file.
LogOn	Opens a log file and captures incoming data into the file.
LogPause	Pauses capturing incoming data into the log file.
LogResume	Resumes capturing incoming data into the log file.
Open	Opens a text file for reading or writing.
Print	Prints integers or strings to the terminal screen.
Put	Sends strings or integers to the remote system.
Query	Queries option value.
Read	Reads a string from a text file.
ReadCh	Reads a character from a text file.
Receive	Receives (Downloads) one or more files from the remote system.
Rename	Renames a file.
Seek	Repositions the file pointer.
Send	Sends (uploads) one or more files to remote system.
Set	Sets option value.
StrDel	Deletes characters from a string.
StrIns	Inserts a string into another string.
StrPos	Scans a string for the occurrence of a given substring.
StrSet	Sets part of a string to a given character.
SubStr	Returns a substring from a given string.

Tell	Returns the current file pointer.
Time	Gets system time.
Waitfor	Waits for one of the given strings from the remote system.
WaitUntil	Waits until the specified time exceeded.
When	Sends a response string to the remote system whenever a given string is matched.
WhenIdle	Sends a string to the remote system if there is no COM Input/Output in the specified time.
Write	Writes integers or strings to a text file.

COM I/O

Print

Put	Sends strings or integers to the remote system.
Get	Gets a string from remote system.
GetCh	Gets a character from remote system.
HangUp	HangUps the modem.
Image	Captures the terminal screen into an image file.
LogOff	Closes the log file.
LogOn	Opens a log file and captures incoming data into the file.
LogPause	Pauses capturing incoming data into the log file.
LogResume	Resumes capturing incoming data into the log file.
Receive	Receives (Downloads) one or more files from the remote system.
Send	Sends (uploads) one or more files to remote system.
Waitfor	Waits for one of the given strings from the remote system.
WaitUntil	Waits until the specified time exceeded.
When	Sends a response string to the remote system whenever a given string is matched.
WhenIdle	Sends a string to the remote system if there is no COM Input/Output in the specified time.

Console I/O

Print	Prints integers or strings to the terminal screen.
Input	Reads a string from the keyboard.
InputCh	Reads a character from the keyboard.

File Handling

Print

Open	Opens a text file for reading or writing.
Append	Opens a text file and moves the file pointer to the end.
Create	Creates a new text file or rewrites an existing one.
Close	Closes a file.
ChDir	Changes current directory.
Delete	Deletes one or more files.
FileExist	Determines if a file or directory is exist.
FileSize	Gets file size in bytes.
Read	Reads a string from a text file.
ReadCh	Reads a character from a text file.
Rename	Renames a file.
Seek	Repositions the file pointer.
Tell	Returns the current file pointer.
Write	Writes integers or strings to a text file.

String Handling

Print

Atoi	Converts a string to an integer.
Concat	Appends one string to another.
Itoa	Converts an integer to a string.
Length	Calculates the length of a string.
StrDel	Deletes characters from a string.
StrIns	Inserts a string into another string.
StrPos	Scans a string for the occurrence of a given substring.
StrSet	Sets part of a string to a given character.
SubStr	Returns a substring from a given string.

Other Procedures

Print

Date	Gets system date.
Delay	Suspends script execution for an interval.
Query	Queries option value.
Set	Sets option value.
Time	Gets system time.

Introduction

Print

What Is a Script

TWScript allows you to write procedures to perform repetitive functions, such as a log on script, as well as specific applications, such as a host mode.

TWScript is designed to be a easy-to-use language. TWScript looks similar to the Pascal language and the syntax is as loose as that of BASIC language. For most of the users, only the WAITFOR, PUT and maybe the WHEN statements are necessary. For those users who have experience in the other programming language, TWScript is a powerful tool for communication related application.

A script file can be created using almost all editors. The source file is a normal ASCII text file which contains a sequence of instructions telling Telemate what to perform next. The source file should use the extension .TWS to indicate that it is a 'S'ource script file. When the run the script file, Telemate compile it automatically and produce a .TWC file which indicate a 'C'omplied script file.

Running a Script

There are several ways to run a script.

1. Pressing [Alt S] at the Terminal Window opens the File Dialog and you are asked to identify which script file to be executed.
2. Put the script name in the link-script field of the phone directory. When the BBS is connected, the link-script is executed automatically. Link-scripts are sometimes referred as logon scripts because they perform repetitive logon procedure.
3. You may assign a function key to execute a script file using the "^\" macro sequence. For example, the function key [Alt 4] is "^\host", when you press [Alt 4], the host script will be executed.
4. The SCRIPT statement allows you run another script in a script program. Please refer the description of the SCRIPT statement.

Your First Script: What Is Your Name

The first example program is a simple logon script. It waits for the logon prompts and sends the response to the remote system.

```
WAITFOR "What is your name?"  ; wait for the logon prompt
PUT "my name"                 ; send your name to remote system
WAITFOR "is your password?"   ; wait for the password prompt
PUT "my password"             ; send your name to remote system
```

A script can be as simple as that and it will help you to logon to the remote system automatically.

Data Types

Print

Integer

Ordinary number notation is used for integers. Decimal and engineering notation (e or E followed by an exponent) is not supported. Integer must within the range from -2147483648 to 2147483647.

String

A string is a sequence of zero or more characters from extended ASCII character set (0-255), enclosed by quotation marks. A string with nothing in it is called a empty string. Two sequential quotation marks in a string denote a single character, an quotation mark.

TWScript allows control characters to be embedded in strings. The ^ character followed by a letter (A-Z, a-z), @ or [denotes a character of the ASCII code [Ctrl A] to [Ctrl Z], NULL, or ESC respective.

Here are some examples of strings

```
"TELEMATE"
"This is a '""'."
"^K^D"              ; [Ctrl K] [Ctrl D]
```

Character

Character is represented as a string that contains only one character. For example, "A" is a character. Control codes are often used in script programs and are represented as

Code	String	Description
Ctrl-@	^@	NULL
Ctrl-A	^A	
...	...	...
Ctrl-H	^H	Backspace
Ctrl-I	^I	Tab
Ctrl-J	^J	Line feed, LF
...	...	...
Ctrl-M	^M	Carriage return, CR, [Enter]
...	...	...
Ctrl-Z	^Z	
Ctrl-[	^[	Escape

Boolean

Boolean (TRUE or FALSE) is represented as a integer. A integer is said to be TRUE if it is not equal to zero, FALSE if it is equal to zero.

When testing a boolean condition, you should not use the following code,

```
IF CONNECTED=1
   ...
ENDIF
```

because the variable CONNECTED may have other values. Instead, you should use

```
IF CONNECTED
   ...
ENDIF
```

to test if CONNECTED is TRUE. And

```
IF NOT CONNECTED
   ...
ENDIF
```

to test if CONNECTED is FALSE.

Date

Date is represented as a string with the format "MM-DD-YYYY". When a date string is compared with another date string, they are first converted to the internal format "YYYYMMDD" so that the comparison returns the correct result.

Note that the date format option in the main program does not affect the format in the script. For date conversion, see ConvertDate command in toolbox #3.

Time

Time is also represented as a string but with the format "HH:MM:SS". When two time strings are being compared, they are converted to the internal format "HHMMSS" such that the comparison can the correct result.

Variables

Print

Variable Identifiers

A variable identifier can contain letters and digits. However, a variable can only start with a letter. Case is not significant, in the other word, the variable <notdone> is the same as <NotDone>. A variable identifier can be of any length, but only the first 8 characters are significant. For examples, the variable identifier <Number12> is the same as <Number123>.

Variable Declarations

A variable declaration embodies a list of identifiers that designate new variables and their type. For example,

```
INTEGER lower,upper,step    ; integers
STRING  message,filename    ; strings
INTEGER true,false          ; boolean
STRING  ch,letter           ; character
```

The variable declaration part should be placed at the beginning of a script or of a procedure. In TWScript, variables need not declared before use if no procedure is defined. Therefore, the variable declaration part can be skipped.

Predefined Variables

TWScript built in procedures do not return value. Several predefined variables are set to the resulting value. They are CONNECTED, FOUND, SUCCESS, and LOGGING.

CONNECTED

Print

CONNECTED is set to the number of the connected dial entry. The variable CONNECTED returns to FALSE as soon as the carrier lost. On the other hand, if the state of the carrier changes from OFF to ON, CONNECTED is set to TRUE.

```
IF NOT CONNECTED
   PRINT "It is not connected to a remote system."
   STOP
ENDIF
SWITCH CONNECTED
   CASE 1: PRINT "connected to #1"
   CASE 3: PRINT "connected to #3"
   CASE 6: PRINT "connected to #6"
ENDSWITCH
```

The CONNECTED variable is also used in conjunction with the DIAL statement. If the dialing is successful and connect to a remote system, the variable is set to the number of the connected entry. If the DIAL statement aborts without connection, the variable CONNECTED is set to FALSE, indicating that the DIAL statement is aborted by operator or the number of attempt exceeds the dial attempt setting.

For example,

```
DIAL "1 3 6"
IF NOT CONNECTED
   PRINT "Dialing process aborted"
   STOP
ENDIF
SWITCH connected
   CASE 1: PRINT "connected to #1"
   CASE 3: PRINT "connected to #3"
   CASE 6: PRINT "connected to #6"
ENDSWITCH
```

Note that this variable reflects the online status only if your modem reports so. You should check your modem manual to ensure that the modem CD (carrier detect) signal is reflecting the actual online status. Most modems use a "&C1" AT command to set it and you should add it to the modem init string.

FOUND

Print

This variable is set to resulting value after the WAITFOR statement is executed. It is set to FALSE if time exceeded. Otherwise, it is set to the string number of the matched string.

Example

```
WAITFOR "NO CARRIER","thanks for calling","hang up now",100
SWITCH FOUND
   CASE 1: PRINT "NO CARRIER found"
   CASE 2: PRINT "thanks for calling found"
   CASE 3: PRINT "hang up now found"
ENDSWITCH
```

SUCCESS

Print

This variable is used by several statements. The resulting values and descriptions is shown in the following table.

Statement	Value	Description
Append	TRUE	the file is successfully opened or created
	FALSE	cannot create file
ChDir	TRUE	directory changed
	FALSE	invalid path
Close	TRUE	file closed
	FALSE	cannot close file
Create	TRUE	the file is successfully created
	FALSE	cannot create file
Delete	TRUE	the file is successfully deleted
	FALSE	cannot delete file
FileSize	TRUE	the file size is determined successfully
	FALSE	cannot open file
InputCh ch	TRUE	a character is read into <ch>
	FALSE	no character is available
LogOn	TRUE	the log file is successfully opened
	FALSE	the log file cannot be created
Open	TRUE	the file is successfully opened
	FALSE	the file does not exist
Read str	TRUE	a line is read into <str>
	FALSE	end of file encountered
ReadCh ch	TRUE	a character is read into <ch>
	FALSE	end of file encountered
Receive	TRUE	all files are received successfully
	FALSE	file transfer aborted

Rename	TRUE	the file is successfully renamed
	FALSE	cannot rename file
Seek	TRUE	the file pointer is moved successfully
	FALSE	disk error
Send	TRUE	all files are sent successfully
	FALSE	file transfer aborted
Tell pos	TRUE	the file pointer is stored in <pos>
	FALSE	disk error
Write	TRUE	write successary to file
	FALSE	cannot write to file

LOGGING

Print

This variable reflects the current file log status and has the values

Value	Description
0	Log file close or log file not in use
1	Log file open and capturing incoming data
2	Log file in pause state

Expressions

Print

Rule Of Precedence

Expressions are made up of operators and operands. In complex expressions, rule of precedence clarify the order in which operations are performed.

Operator	Precedence
*, /	first (high)
+, -	second
=, <>, <, >, <=, >=	third
not	forth
and, or, xor	fifth (low)

There are the rules of precedence

1. First, an operand between two operators of difference precedence is bound to the operator with higher precedence.
2. Second, an operand between two equal operators is bound to the one on its left.
3. Third, expressions within parentheses are evaluated prior to being treated as a single operand.

Arithmetic Operators

The types of operands for arithmetic operators are shown in the following table.

Operator	Operation	Operand Type

+	addition	integer
-	subtraction	integer
*	multiplication	integer
/	division	integer

For example, the formula to convert Fahrenheit temperature to Celsius equivalents is

```
celsius = (fahr-32) * 5 / 9
```

Boolean Operators

The types of operands for arithmetic operators are shown in the following table.

Operator	Operation	Operand Type
not	negation	boolean
and	logical and	boolean
or	logical or	boolean
xor	logical xor	boolean

Normal Boolean logic governs the results of these operations. For instance, a and b is TRUE only if both <a> an <b> are TRUE.

Relational Operators

The types of operands for arithmetic operators are shown in the following table. They all have the same precedence.

Operator	Operation	Operand Types
=	equal to	integer, string, date ,time
<>	not equal to	integer, string, date ,time
>	greater than	integer, string, date ,time
>=	greater or equal to	integer, string, date ,time
<	less then	integer, string, date ,time
<=	greater or equal to	integer, string, date ,time

When the operands are integer, date or time, the comparison will produce the usual result such as

Condition	Result	Operand Type
1 > 2	FALSE	integer
"12-31-1989" < "01-01-1990"	TRUE	date
"00:00:00" > "23:59:59"	FALSE	time

However, if the operands are strings, the operators "<", ">" and "<>" have another meaning such that you can determine if a string is a sub-string of another string.

```
s1 <  s2    if   s1 is a sub-string of s2
s3 >  s4    if   s3 is a super-string of s4
s5 <> s6    if   s5 is not a sub-string of s6 and
                 s5 is not a super-string of s6
```

Example

Condition	Result	Explanation
"hello" = "Hello"	TRUE	TWScript is not case sensitive.
"goodbye" < "bye"	TRUE	"bye" is a sub-string of "goodbye"
"dog" <> "car"	TRUE	The string is not equal
"abc" >= "xyz"	FALSE	"abc" is not a super-string of "xyz" and they are not equal.

Statements

Print

Statements describe algorithmic actions that can be executed.

Comment

Print

Any characters after ";" are ignored by the TWScript compiler.

Example

```
WAITFOR "What is your name?"  ; wait for the logon prompt
PUT "my name"                 ; send your name to remote system
WAITFOR "is your password?"   ; wait for the password prompt
PUT "my password"             ; send your name to remote system
```

Assignment

Print

Assignment statements replace the current value of a variable with a new value specified by an expression. The expression must be assignment-compatible with the type of the variable.

Example

The program prints the following table of Fahrenheit temperatures and their centigrade or Celsius equivalents.

```
0     -17
20    -6
40    4
...   ...
280   137
300   148
```

Here is the program itself.

```
; Print Fahrenheit-Celsius table
;       for f = 0, 20, ... , 300

INTEGER lower,upper,step
INTEGER fahr,celsius

lower = 0       ; lower limit of temperature table
upper = 300     ; upper limit
step  = 20      ; step size

fahr  = lower                   ; first item in the table
WHILE fahr <= upper
   celsius = (fahr-32) * 5 / 9  ; calcuate the values
   PRINT fahr,"^I",celsius      ; print the result
   fahr = fahr + step           ; next item in the table
ENDWHILE
```

If...Else...Elself...EndIf

Print

The general form for IF statement is

```
IF condition
   statements
```

```
ENDIF
```

The <statements> is executed only if <condition> is TRUE. Another form is

```
IF condition
   statements-1
ELSE
   statements-2
ENDIF
```

One and only one of the two statements associated with an <if-else> is done. If the <condition> is true, <statements-1> is executed; if not, <statements-2> is executed.

The construction

```
IF condition-1
   statements-1
ELSE
   IF condition-2
      statements-2
   ELSE
      IF condition-3
         statements-3
      ENDIF
   ENDIF
ENDIF
```

occurs so often that it is worth a new keyword ELSEIF. The statement can be re-written as

```
IF condition-1
   statements-1
ELSEIF condition-2
   statements-2
ELSEIF condition-3
   statements-3
```

ENDIF

See also

Switch...EndSwitch

Switch...EndSwitch

Print

The SWITCH statement is a special multi-way decision maker that tests whether an expression matches one of a number of values, an branches accordingly. The syntax for SWITCH statement is

```
SWITCH expression
   CASE value-1:
      statements-1
   CASE value-2,value-3:
      statements-2
   OTHERWISE:
      statements-3
ENDSWITCH
```

The SWITCH evaluates the expression and compares its value to all the cases. Each cases must be labeled by the values of the same type as the expression. Several values can be separated by comma. If a case matches the expression value, execution starts at that case and ends at the next case label. The case labeled OTHERWISE is executed if none of the other cases is satisfied. A OTHERWISE is optional; if it isnt there and if none of the cases matches, no action at all takes place. However, OTHERWISE, if exists, must be placed after all the case labels.

See also

If...Else...ElseIf...EndIf

Example

The program counts digits, blanks, others.

```
nDigit = 0          ; digit
nBlank = 0          ; blank
nOther = 0          ; others
OPEN "MYFILE"       ; open the file "MYFILE"
READCH ch           ; read the first character
WHILE success       ; if not end of file
SWITCH ch
   CASE "0","1","2","3","4","5","7","8","9":  ; is digit
      nDigit = nDight + 1
   CASE " ":                                  ; is blank
      nBlank = nBlank + 1
   OTHERWISE:                                 ; others
      nOther = nOther + 1
   ENDSWITCH
   READCH ch        ; read next character
ENDWHILE
CLOSE               ; close the file
PRINT nDigit        ; print the results
PRINT nBlank
PRINT nOther
```

While...EndWhile Loop

Print

The WHILE loop is the most general loop. The syntax is

```
WHILE condition
   statements
```

```
ENDWHILE
```

The <condition> is tested. If it is true, the body of the loop (all the statements before the keyword ENDWHILE is executed. Then the condition is re-tested, and if true, the body is executed again. When the test becomes false the loop ends, and execution continues at the statements that follows the loop.

See also

Repeat...Until, Break Statement

Example

For example, to print the value from 1 to 100, you can write

```
i = 1                ; start from 1
WHILE i<=100         ; check if it in the range
   PRINT i           ; print the number
   i = i + 1         ; increase the counter by 1
ENDWHILE
```

Repeat...Until Loop

Print

While the WHILE loop test the condition at the top, the REPEAT loop test it at the bottom. It tests at the bottom after making each pass through the loop body; the body is always executed at least once.

Consider the loop

```
REPEAT
   statements
UNTIL condition
```

The statements is executed, then the condition is evaluated. If it is false, the statements is evaluated again, and so on. If the condition becomes true, the loop terminates.

See also

While...EndWhile, Break Statement

Example

For example, to print the value from 1 to 100, you can write

```
i = 1                ; start from 1
REPEAT
   PRINT i           ; print the number
   i = i + 1         ; increase the counter by 1
UNTIL i>100          ; repeat until it is NOT in the range
```

Break Statement

Print

It is sometimes convenient to be able to control loop exits other than by testing at the top or bottom. The BREAK statement provides an early exit from the loops. A BREAK statement causes the innermost loop to be broken immediately.

The following program accept strings from keyboard, using a BREAK to exit from the loop when the string contains an [Esc] ("^[").

Example

```
REPEAT
```

```
   INPUT s
   IF "^[" <= s      ; test if [Esc] is a sub-string of s
      BREAK          ;   before printing it
   ENDIF
   PRINT s
UNTIL s = ""         ; repeat until an empty string is entered
```

Stop Statement

Print

The STOP statement terminates the execution of the current script program. It is usually used when an error is encountered.

Example

```
OPEN "MYFILE"                ; open a file
IF NOT SUCCESS
   PRINT "File not found."
   STOP                      ; terminate if file not found
ENDIF
```

#include Directive

Print

This compiler directive allows you to write reusable procedures in a eparate file and imports the procedures conveniently. The syntax is

```
#include "include_file"
```

The <include_file> will be inserted as if it is physical appear in this oint. The <includefile> must be a pathname with extension. For example,

```
#include "TOOLBOX.TWS"
#include "MYLIB.TWS"
```

inserts the TOOLBOX.TWS and MYLIB.TWS into this point.

The #include directive can be nested as deep as 10 levels.

Procedures

Print

What Is a Procedure

In TWScript, a procedure is equivalent to a subroutine or function in Fortran, or procedure in Pascal, C etc. A procedure provides a convenient way to encapsulate some computation in a black box, which can then be used without worrying about its innards.

For example, to swap the values of two variables, <i> and <j>, you have to write three lines as follows

```
temp = i        ; put <i> into a temporary variable
i = j           ; put <j> into <i>
j = temp        ; now put the value of <i> into <j>
```

Suppose in your script there are a lot of swapping, it will be convenient to define a procedure called <swap> to perform the swapping. Then your main program will look like

```
i = 1
j = 2
swap i,j        ; now i=2, j=1
PRINT i         ; 2
PRINT j         ; 1
```

This is only a small usage of procedure. Imagine if the procedure is very complex, you can use it again and again once it is tested and performs what you need.

Procedure Declaration

The syntax of a procedure definition is

```
PROCEDURE <procname> <parameter declaration>
<local variable declaration>
<statement part>
ENDPROC
```

<procname> is any valid identifier. <parameter declaration> can be omitted if the procedure does not required parameters. See next section for details on parameter declaration.

<local variable declaration> made within a given procedure are visible only within that procedure. This part should ALWAYS be included in order to make the program easy to be traced although the declarations may be omitted.

<statement part> can be considered as a sub-program of the main program and obeys all the rule described above.

Parameter Declaration

Parameter declaration is similar to variable declaration. For example,

```
PROCEDURE sample STRING s,t,INTEGER i,j
```

declares the procedure <sample> with two string parameters <s> and <t> and two integer parameters <i> and <j>. You may use any combination of STRING and INTEGER to define parameters. For example,

```
PROCEDURE sample INTEGER i,STRING p,q,r,s,INTEGER j,k,STRING z
```

We can write the <swap> procedure as

```
PROCEDURE swap INTEGER value1,value2 ; swap two integers
INTEGER temp          ; declare a local variable
temp = value1         ; store the first value
value1 = value2       ; replace with the second value
value2 = temp         ; replace with the first value
```

```
ENDPROC
```

After that, the main program may look like

```
i = 1
j = 2
k = 3
swap i,j            ; now i=2, j=1
swap j,k            ; now j=3, k=1
PRINT i,j,k         ; 2,3,1
```

Calling Procedure and Parameter Passing

Once the procedure is defined, it can be called by putting the procedure name in the statement part. In the example, the statements

```
i = 1
j = 2
swap i,j            ; now i=2, j=1
PRINT i,j           ; 2,1
```

call the <swap> procedure with the parameters <i> and <j>.

In the above example, the method used to pass <i> and <j> to the procedure <swap> is called pass by reference because <i> and <j> can be changed when return. There is another method called pass by value of which the parameter cannot be changed.

Consider the following procedure

```
PROCEDURE countDown INTEGER num ; print from <num> to 1
WHILE num >= 1              ; check if it is in the range
   PRINT num                ; print it
   num = num-1              ; decrease until num=0
   ENDWHILE
ENDPROC

countDown 100               ; pass by value
n = 100
countDown n*2               ; pass by value (200)
PRINT n                     ; n = 100
countDown (n)               ; pass by value (100)
PRINT n                     ; n = 100
countDown n                 ; pass by reference
PRINT n                     ; n = 0 !
```

If we supply <countDown> with the parameter 100, n*2 or (n), they are pass by value. On the other hand, <n> is pass by reference in the last statement.

The general rule of method of passing parameters is:

> When the parameter is a constant, an expression or surrounded by parentheses, it is pass by value. Otherwise, it is a variable and is pass by reference.

The rule applies to string parameter as well. To prevent confusion, we should rewrite the <countDown> procedure by introducing a local variable.

```
PROCEDURE countDown INTEGER num ; print from <num> to 1
INTEGER count               ; introduce a local variable
count = num                 ;  if neccessary
WHILE count >= 1            ; check if it is in the range
   PRINT count              ; print it
   count = count-1          ; decrease until 0
ENDWHILE
ENDPROC                     ; this does not change <num>
```

Nested Procedures and Scope of Variables

TWScript allows nested procedures; you can declare one procedure inside of another. For example,

```
PROCEDURE outer
STRING i,j

   PROCEDURE inner
   INTEGER i
   i = 100
   PRINT "i = ",i
   PRINT "j = ",j
   ENDPROC

i = "This is string 'i'."
j = "This is string 'j'."
inner
PRINT "i = ",i
PRINT "j = ",j
ENDPROC
```

The <outer> procedure declares two string variables, <i> and <j>. These two variables can be accessed by both the <outer> procedure and the <inner> one because they are declared before the <inner> procedure.

The <inner> procedure declares an integer variable <i>. Although the two <i>s share the same name, they are not identical. The inner <i> is visible in the <inner> procedure but not in the <outer> procedure.

On the other hand, the outer <i> can no longer be accessible in the <inner> procedure. The statement

```
i = 100
```

does not affect the outer <i>. This is called the scope rule.

The output of the program is

```
i = 100
j = This is string 'j'.
i = This is string 'i'.
j = This is string 'j'.
```

Return Statement

Like the BREAK statement, it is sometimes convenient to be able to return from a procedure at the middle of the procedure.

The following procedure accept strings from keyboard and print it, using a RETURN to return from the procedure when the string contains an [Esc] ("^[").

```
PROCEDURE acceptString STRING s
INPUT s
IF "^[" <= s     ; test if [Esc] is a sub-string of s
   RETURN        ;   before printing it
ENDIF
PRINT s
ENDPROC
```

Append

Print

Syntax

APPEND filename

Description

Opens a text file, creates it if necessary, and moves the file pointer to the end of the file.

APPEND opens the text file as a read/write file and strip the ending EOF [Ctrl Z].

Return value

Upon successful completion, SUCCESS is set to TRUE. If the file cannot be created, SUCCESS is set to FALSE.

Atoi

Print

Syntax

ATOI s,i

Description

Converts a string to an integer.

Return value

<i> contains the converted value or 0 if <s> cannot be converted to a number.

See also

Itoa

ChDir

Print

Syntax

CHDIR path

Description

Changes current directory.

CHDIR causes the directory specified by <path> to become the current working directory. <path> must specify an existing directory.

A drive can also be specified in <path> but it changes only the current directory on that drive; it doesn't change the active drive.

Return value

Upon successful completion, SUCCESS is set to TRUE; otherwise, SUCCESS is FALSE.

See also

Delete, FileExist, Rename

Close

Print

Syntax

CLOSE

Description

Closes a file.

Return value

Upon successful completion, SUCCESS is set to TRUE; otherwise, SUCCESS is FALSE.

See also

Open, Read, Write

Concat

Print

Syntax

CONCAT dest,src

Description

Appends one string to another.

CONCAT appends a copy of <src> to the end of the string <dest>.

If the backspace character "^H" is encountered in <src>, the last character of the concatenating string is erased.

Return value

<dest> contains the concatenated string.

See also

Length, StrDel, StrIns

Example

```
s = "abc"           ; s = "abc"
CONCAT s,"xyz"      ; s = "abcxyz"
CONCAT s,"^H"       ; s = "abcxy"
CONCAT s,"pq"       ; s = "abcxypq"
```

Create

Print

Syntax

CREATE filename

Description

Creates a new text file or rewrites an existing one.

CREATE creates the new text file <filename>. If the file already exists, the old file is deleted.

Return value

Upon successful completion, SUCCESS is set to TRUE; otherwise, SUCCESS is FALSE.

See also

Append, Open

Date

Print

Syntax

DATE today

Description

Gets system date.

DATE fills the string <today> with the current date.

Dates can be compared with the usual '<', '>' and '=' relational operator.

The date format option in the main program does not affect the format in the script language.

Return value

<today> contains the system current date in MM-DD-YYYY format.

See also

Time

Example

```
DATE today
IF today>="01-01-91" AND today<"01-05-91"
   PRINT "Happy New Year!"
ENDIF
```

Delay

Print

Syntax

DELAY t

Description

Suspends script execution for an interval.

With a call to DELAY, the script program is suspended from execution for the number of tenth seconds specified by <t>.

Example

```
DELAY 15  ; wait for one and a half second
```

Delete

Print

Syntax

DELETE filename

Description

Deletes one or more files.

DELETE deletes one or more files specified by <filename>.

Wildcards are allowed in <filename>.

Return value

On successful completion, SUCCESS is set to TRUE; otherwise, SUCCESS is FALSE.

See also

FileExist, Rename

FileExist

Print

Syntax

FILEEXIST filename,existFlag

Description

Determines if a file or directory is exist.

FILEEXIST checks the file named by <filename> to determines if it is exists. Wildcards are allowed in <filename>.

Return value

If the file is exist, <existFlag> is set to TRUE; otherwise, <existFlag> is FALSE.

See also

Delete, Rename

FileSize

Print

Syntax

FILESIZE filename,size

Description

Gets file size in bytes.

FILESIZE returns the length, in bytes, of the file specified by <filename>.

Return value

If <filename> exist, <size> is set to the file size and SUCCESS is set to TRUE; otherwise, SUCCESS is set to FALSE.

See also

Read, Seek

Get

Print

Syntax

GET s

Description

Gets a string from remote system.

GET collects a string, terminated by a carriage return, from the remote system.

GET does not eliminate the BackSpace character [Ctrl H] from the input, therefore a CONCAT statement should be perform after the GET statement to eliminate the BackSpace character.

Return value

<s> contains the collected string. The ending carriage return is discarded.

See also

GetCh

Example

```
GET s                ; get a string from remote system
filename = ""        ; prepare the variable
CONCAT filename,s    ; eliminate the BackSpace
```

GetCh

Print

Syntax

GETCH ch

Description

Gets a character from remote system.

GETCH gets a single character from the remote system. If no character is available, GETCH return immediately. GETCH statement, like the INKEY$ function in BASIC, does not wait.

Return value

If a character is successfully read into <ch>, SUCCESS is set to TRUE; otherwise, SUCCESS is set to FALSE.

See also

Get

Example

```
REPEAT          ; to wait for a character,
   GETCH ch     ;  repeat the GETCH until
UNTIL SUCCESS   ;  a character is available
```

HangUp

Print

Syntax

HANGUP

Description

HangUps the modem.

HANGUP sends the modem hangup string to modem. If the hangup string is the "^#" macro sequence, Telemate hangs up the modem by dropping DTR.

Image

Syntax

IMAGE

IMAGE filename

Description

Captures the terminal screen into an image file.

IMAGE captures the terminal screen into an image file by appending the screen to the file <filename>. If no <filename> is supplied, the last image filename is used.

See also

LogOn

Input

Print

Syntax

INPUT s

Description

Reads a string from the keyboard.

INPUT collects a string, terminated by a [Enter], from the keyboard.

When the INPUT statement in the script is executed, the following input characters are placed in a keyboard buffer no matter if they are to be read or not. As a result, they NOT sent to the remote system automatically. It is important to give up the control of keyboard whenever no more keyboard input is needed using the CLEAR KEY statement.

INPUT does not eliminate the BackSpace character [Ctrl H] from the input, therefore a CONCAT statement should be perform after the INPUT statement to elimate the BackSpace character.

Return value

<s> contains the collected string. The ending [Enter] is discarded.

See also

InputCh

Example

```
INPUT s              ; get a string from remote system
filename = ""        ; prepare the variable
CONCAT filename,s    ; eliminate the BackSpace
CLEAR KEY            ; give up keyboard control such that
                     ;  following keys are sent to remote
                     ;  system automatically
```

InputCh

Print

Syntax

INPUTCH ch

Description

Reads a character from the keyboard.

INPUTCH gets a single character from the keyboard. If no character is available, INPUTCH returns immediately.

INPUTCH statement, like the INKEY$ function in BASIC, does not wait. When the INPUTCH statement in the script is executed, the following input characters are placed in a keyboard buffer no matter if they are to be read or not. As a result, they are NOT sent to the remote system automatically. It is important to give up the control of keyboard whenever no more keyboard input is needed using the CLEAR KEY statement.

Return value

If a character is successfully read into <ch>, SUCCESS is set to TRUE; otherwise, SUCCESS is set to FALSE.

See also

Input

Example

```
REPEAT          ; to wait for a character,
   INPUTCH ch   ;  repeat the INPUTCH until
UNTIL SUCCESS   ;  a character is available
CLEAR KEY       ; give up keyboard control such that
                ;  following keys are sent to remote
                ;  system automatically
```

Itoa

Print

Syntax

ITOA i,s

Description

Converts an integer to a string.

ITOA converts the integer <i> to a string and store it into <s>.

Return value

<s> contains the string representation of the value <i>.

See also

Atoi

Length

Print

Syntax

LENGTH s,len

Description

Calculates the length of a string.

Return value

<len> is the length of <s>. If <s> is an empty, <len> is 0.

LogOff

Print

Syntax

LOGOFF

Description

Closes the log file.

LOGOFF closes a previous opened log file.

Return value

LOGGING is set to 0 (FALSE).

See also

LogOn, LogPause, LogResume

LogOn

Print

Syntax

LOGON

LOGON filename

Description

Opens a log file and captures incoming data into the file.

LOGON opens the log file <filename> and starts capturing the incoming data into it.

If <filename> is not supplied, the filename specified in the last LOGON statement is used. If there is no prevous LOGON statement, the one in the log file field of the phone directory is used. If this field is empty, "TM.LOG" is used.

Return value

Upon successful completion, SUCCESS is set to TRUE and LOGGING is set to 1 (TRUE); otherwise, SUCCESS is 0 (FALSE).

See also

LogOff, LogPause, LogResume

LogPause

Print

Syntax

LOGPAUSE

Description

Pauses capturing incoming data into the log file.

LOGPAUSE pauses capturing incoming data. The LOGGING variable should be checked to determine if a log file open. A value of 1 indicates a log file is open.

Return value

LOGGING is set to 2.

See also

LogOff, LogOn, LogResume

LogResume

Print

Syntax

LOGRESUME

Description

Resumes capturing incoming data into the log file.

LOGRESUME resumes capturing incoming data. The LOGGING variable should be checked to determine if the log file is in pause. A value of 2 indicates the log file is in pause.

Return value

LOGGING is set to 1.

See also

LogOff, LogOn, LogPause

Open

Print

Syntax

OPEN filename

Description

Opens a text file for reading or writing.

OPEN opens the text file <filename> for reading and writing and strip the ending EOF [Ctrl Z].

OPEN closes the previously open file automatically if no CLOSE command is issued to that file.

Return value

Upon successful completion, SUCCESS is set to TRUE. If <filename> does not exist, SUCCESS is FALSE.

See also

Append, Close, Create, FileExist, FileSize, Read, Seek, Tell, Write

Example

```
OPEN "MYFILE"       ; open the file "MYFILE"
IF NOT SUCCESS      ; report if not found
   PRINT "File not found."
   STOP
ENDIF
READ s              ; read the first line
PRINT s             ; print it
CLOSE               ; close the file

OPEN "FILE1"        ; open FILE1
OPEN "FILE2"        ; close FILE1 and open FILE2
```

Print

Print

Syntax

PRINT

PRINT s

PRINT i

PRINT s1,s2,i1,i2,s3, ...

PRINT s1,s2,i1,i2,s3, ... ,

Description

Prints integers or strings to the terminal screen.

PRINT outputs variables or constants of integer or string to the terminal screen. Each two arguments are separated by a comma.

PRINT supplies a newline by default. If a comma follows, no newline is supplied.

Example

```
PRINT "hello, world"   ; say hello to everyone
PRINT "hello, ",       ; say that again
```

```
PRINT "world",
PRINT

PRINT "x = ",x          ; output multiple strings or integers
PRINT "My name is ",firstName," ",lastName
```

Put

Print

Syntax

PUT

PUT s

PUT i

PUT s1,s2,i1,i2,s3, ...

PUT s1,s2,i1,i2,s3, ... ,

Description

Sends strings or integers to the remote system.

Like the PRINT statement, PUT sends variables or constants of types integer or string to the remote system. Each two arguments are separated by a comma. Integers are converted to their string representation automatically before transmitting to the remote system.

PUT supplies a carriage return [Ctrl M] by default. If a comma follows, no carriage return is supplied.

Control characters can be sent by using the '^' prefix. For example, "^C" represents the [Ctrl C], "^M" the carriage return [Enter] and "^[" the Escape key [Esc].

The following macro symbol have special functions.

Symbol	Function
^^	Send the character '^'
^~	Send the character '~'
~	Pause 0.5 second
^!	Send the user ID from the dial entry
^&	Send the password from the dial entry
^$	Send the memo from the dial entry
^(	Initialize the modem
^)	Switch the modem to answer mode
^*	Hang up modem
^#	Drop DTR
^%	Send break signal

See also

Waitfor

Example

```
PUT "first last"    ; transmit first name and last name
                    ;  then a carriage return (^M)
PUT "first ",       ; same as above
PUT "last",
PUT
```

```
PUT                 ; these two lines are
PUT "^M",           ;  equivalent
firstName = "first" ; set first and last name
lastName = "last"
password = "^&"     ; use the password field in TM.FON
PUT firstName," ",lastName,"^M~~~~",password
                    ; send name, [Enter], wait for 2 seconds
                    ;  then send the password, [Enter]

PUT "~^#~~^)",      ; wait 0.5 second, drop DTR,
                    ;  wait 1 second, then sends the
                    ;  modem answer string
```

Query

Print

Syntax

QUERY <OPTION>,<VALUE>

Description

Queries option value.

The QUERY statement gives you the ability to peek most of the system options.

All the options that the SET command accept can be queried by the QUERY command.

Option	Value	Description
AutoStop	0 / 1	return the AutoStop flag

This command should be used to store the value of an option before modifying the option and then restore the option to its original before the execution stops.

See also

Set

Example

```
INTEGER flag
QUERY AutoStop, flag    ; query the AutoStop flag
PRINT flag
```

Read

Print

Syntax

READ s

Description

Reads a string from a text file.

READ reads characters from the file into <s>.

READ does not place the newline sequence CR-LF into the string.

Return value

On success, <s> contains the string read and SUCCESS is set to TRUE. SUCCESS is FALSE on end-of-file or error.

See also

Close, Open, ReadCh, Seek, Tell

Example

```
OPEN "MYFILE"       ; count lines in the file MYFILE
IF NOT SUCCESS
   PRINT "File not found."
   STOP             ; stop if file not found
ENDIF
n = 0               ; number of line = 0
READ str            ; read a line
WHILE SUCCESS       ; repeat until end of file
   n = n + 1        ; increase counter
   READ str         ; read the next string
ENDWHILE
CLOSE               ; close the file
PRINT "There are ",n," lines in the file"
OPEN "TM.FON"       ; read a record from the phone directory
recno = 10          ; read the record #10
reclen = 131        ; record length of a entry
SEEK recno*reclen   ; seek to the record position
READ record         ; read the record
CLOSE
```

ReadCh

Print

Syntax

READCH ch

Description

Reads a character from a text file.

READCH reads a single character from the file into <ch>.

In a text file, a CR [Ctrl M] followed by a LF [Ctrl J] indicates end-of-line.

Return value

On success, <ch> contains the character read and SUCCESS is set to TRUE. SUCCESS is FALSE on end-of-file or error.

See also

Close, Open, Read, Seek, Tell

Receive

Print

Syntax

RECEIVE protocol

RECEIVE protocol,filename

Description

Receives (Downloads) one or more files from the remote system.

RECEIVE receives (download) one or more files from the remote system using the protocol <protocol>. <protocol> can be one of the following or the menu key for external protocol.

Protocol	Filename needed
"Zmodem"	No
"Ymodem"	No
"Ymodem-G"	No
"Xmodem"	Yes
"Xmodem-CRC"	Yes
"Xmodem-1K"	Yes

For protocols, such as Xmodem, which does not pass the name, <filename> should contains the name being received. Wildcards are not allowed. If <filename> is "" and <GuessFile> option is on, the guessing name is used.

Zmodem has the ability to start automatically which is called Zmodem AutoDownload - the <zAutoDownload> option. To prevent the download starts before the command RECEIVE "Z" is issued, the <zAutoDownload> option should be turned off before telling the remote system to start the transfer.

Return value

If the file transfer is successful, SUCCESS is set to TRUE; SUCCESS is FALSE if the transfer abort.

See also

Send, Waitfor

Example

```
RECEIVE "Xmodem-CRC","file.zip"   ; Xmodem needs the filename
RECEIVE "Ymodem"                  ; Ymodem doesnt

RECIEVE "Xmodem-CRC",""           ; use the guessing name as filename

WAITFOR "command",10     ;  the remote system to send the files,
PUT "d z *.zip"          ;  otherwise, the systems AutoDownload
RECEIVE "Zmodem"         ;  procedure will take the control and
                         ;  the RECEIVE command will return a
                         ;  wrong SUCCESS value, after the
IF SUCCESS
   PRINT "File received successfully"
ELSE
   PRINT "File transfer aborted"
ENDIF
```

Rename

Print

Syntax

RENAME oldname,newname

Description

Renames a file.

RENAME changes the name of a file from <oldname> to <newname>.

Directories in <oldname> and <newname> need not be the same, therefore, RENAME can be used to move a file from one directory to another. Wildcards are not allowed.

Return value

On successfully renameing the file, SUCCESS is set to TRUE; otherwise, SUCCESS is FALSE.

See also

Delete, FileExist

Seek

Print

Syntax

SEEK filepos

Description

Repositions the file pointer.

SEEK sets the file pointer to the new position <filepos>. At the beginning of a file, the file pointer is 0.

If <filepos> is -1, the file pointer is moved to the end of the file.

Return value

If the pointer is successfully moved, SUCCESS is set to TRUE; otherwise, SUCCESS is FALSE.

See also

Read, ReadCh, Tell, Write

Send

Print

Syntax

SEND protocol,filenames

Description

Sends (uploads) one or more files to remote system.

SEND sends (uploads) one or more files to the remote system using the protocol <protocol>. <protocol> can be one of the following or the menu key for external protocol.

Protocol	Send multiple files
"Zmodem"	Yes
"Ymodem"	Yes
"Ymodem-G"	Yes
"Xmodem"	No
"Xmodem-CRC"	No
"Xmodem-1K"	No

<filenames> is a list of filenames to be sent, wildcard characters * and ? can be used. If <filename> is "" and filename guessing option is on, the guessing name is used. Multiple files are separated by a space. For example,

```
"\TMW\TMW100.ZIP \TMW\README.TXT"
"*.ZIP \UTIL\*.EXE *.TXT"
```

For protocols, such as Xmodem, which cannot transfer multiple files, only the first file in <filenames> is used.

Return value

If the file transfer is successful, SUCCESS is set to TRUE; SUCCESS is FALSE if the transfer abort.

See also

Receive, Waitfor

Example

```
SEND "Ymodem"                  ; use the guessing name as filename
SEND "Zmodem","*.rep"          ; send multiple files

IF SUCCESS
   PRINT "File sent successfully"
ELSE
   PRINT "File transfer aborted"
ENDIF
```

Set

Print

Syntax

SET <OPTION>,<VALUE>

SET <OPTION>,<TOKEN>

Description

Sets option value.

The SET statement gives you control over many of the system options. The possible options and values are listed below.

Option	Value	Description
AutoStop	Off (0) On (1)	when On, the script will stop automatically upon carrier lost, this setting is reset to Off at the beginning of every script. This flag is usually set on the very beginning of a script

Example

```
SET AutoStop, On    ; stop the script when disconnected
```

StrDel

Print

Syntax

STRDEL str,pos,count

Description

Deletes characters from a string.

STRDEL deletes <count> characters from the position <pos> of the string <str>.

The first character position is 1.

Return value

<str> contains the new string.

See also

Concat, Length, StrIns, StrPos, StrSet, SubStr

Example

```
str = "abcXYZdef"
STRDEL str,4,3      ; delete "XYZ"
PRINT str           ; "abcdef"
```

StrIns

Print

Syntax

STRINS str,substr,pos

Description

Inserts a string into another string.

STRINS inserts a string <substr> into another string <str> at position <pos>.

If <pos> is larger than the length of <str>, the gap is filled with spaces. The first character position is 1.

Return value

<str> contains the new string.

See also

Concat, Length, StrDel, StrPos, StrSet, SubStr

Example

```
str = "abcdef"
STRINS str,"XYZ",4  ; insert "XYZ"
PRINT str           ; "abcXYZdef"

str = "abc"
STRINS str,"xyz",10 ; <count> is larger than the length of <str>
PRINT str           ; "abc      xyz"
```

StrPos

Print

Syntax

STRPOS str,substr,pos

Description

Scans a string for the occurrence of a given substring.

STRPOS scans <str> for the first occurrence of the substring <substr>. Case is not sensitive.

The first character position is 1.

Return value

If <substr> is a substring of <str>, <pos> is the position of the substring; otherwise, <pos> is 0.

See also

Concat, Length, StrDel, StrIns, StrSet, SubStr

Example

```
STRPOS "abcdef","def",pos
PRINT pos                   ;  4
```

StrSet

Print

Syntax

STRSET str,ch,pos,count

Description

Sets part of a string to a given character.

STRSET sets the string <str> starting from position <pos> with <count> character <ch>.

If <pos> is larger than the length of <str>, the gap is filled with spaces. The first character position is 1.

Return value

<str> contains the new string.

See also

Concat, Length, StrDel, StrPos, StrIns, SubStr

Example

```
str = "abc"
STRSET str,"X",3,10
PRINT str            ; "abXXXXXXXXXX"
```

SubStr

Print

Syntax

SUBSTR src,pos,count,dest

Description

Returns a substring from a given string.

SUBSTR returns a substring in the string <src> starting at <pos> of length <count> into the string <dest>.

The first character position is 1.

Return value

<dest> contains the substring.

See also

Concat, Length, StrDel, StrPos, StrIns, StrSet

Example

```
SUBSTR "abcdef",1,3,dest
PRINT dest                   ; "abc"
```

Tell

Print

Syntax

TELL filepos

Description

Returns the current file pointer.

TELL returns the current file pointer. <filepos> is measured in bytes from the beginning of the file. At the beginning of the file, <filepos> is 0.

Sometime it is neccessary to open more than one file, you can implement it by storing the file position of the original file, then restore it after the other file operation is completed.

Return value

<filepos> is the current file pointer.

See also

Seek, Read, Write

Example

```
i = 1
```

```
filepos = 0                ; at at beginning of file
WHILE i<=10
   OPEN "bbsname"          ; open data file
   SEEK filepos            ; move the previous position
   READ name               ; read a bbs name
   TELL filepose           ; store the current position
   CLOSE                   ; close the file
   PhoneFind name,number   ; find the board in TM.FON
                           ;  suppose the PhoneFind procedure
                           ;  open the TM.FON
   IF number<>0            ; yes, it is in TM.FON
      PRINT name," found is entry #",number
   ENDIF
ENDWHILE
```

Time

Print

Syntax

TIME now

Description

Gets system time.

TIME fills the string <now> with the current time.

Times can be compared with the usual '<', '>' and '=' relational operator.

Return value

<now> contains the system current date in HH:MM:SS 24 hour format.

See also

Date

Example

```
TIME now
IF now>"07:00:00" AND now<="07:59:59"
   PRINT "Good morning."
ENDIF
```

Waitfor

Print

Syntax

WAITFOR t

WAITFOR s1,s2, ... , sN

WAITFOR s1,s2, ... , sN, t

Description

Waits for one of the given strings from the remote system.

The 'WAITFOR t' format set the default waiting time to <t>, in second. If <t> is 0, no time checking is performed. If <t> is -1, no time checking is performed, but the waiting time is considered to be

exceeded when a key is hit.

The 'WAITFOR s1,s2, ... , sN' format waits until one of the given string is received from the remote system or the default waiting time exceeded.

The 'WAITFOR s1,s2, ... , sN,t' format set the default waiting time to <t>, in second and waits until one of the given string is received from the remote system or the waiting time exceeded.

Case is not sensitive when comparing the given string with the characters received from the remote system.

Sometimes the remote system need a slight delay between the prompt and the response, you can use DELAY or the "~" half-second marco before the PUT command.

Return value

If the waiting time exceeded, FOUND is set to 0 (FALSE); otherwise, FOUND is set the string number of the matched string.

See also

Delay, Put, Receive, Send, WaitUntil, When, WhenIdle

Example

```
WAITFOR "first",30        ; wait for "first" in 30 seconds
IF NOT FOUND
   PRINT "Not found."     ; if not found, stop
   STOP
ELSE
   DELAY 5
   PUT "^!"               ; else send the user ID
ENDIF

; The following is usually placed at the end of a script file.
; It waits for disconnection and print which string is found.
WAITFOR "NO CARRIER","thanks for calling","hang up now",0
PRINT "Ending connection"

; The following simulates the AutoDownload feature for protocols
; does not support this feature.
WAITFOR "Download protocol is",0
WAITFOR "Xmodem","Ymodem",5
IF FOUND
   SWITCH FOUND
      CASE 1: RECEIVE "Xmodem-CRC","" ; Xmodem, use guessing name
      CASE 2: RECEIVE "Ymodem"        ; Ymodem download
   ENDSWITCH
ENDIF
```

WaitUntil

Print

Syntax

WAITUNTIL t

Description

Waits until the specified time exceeded.

WAITUNTIL pauses the execution and waits until the time <t>, in HH:MM:SS 24 hour format, exceeded.

See also

Waitfor

Example

```
WAITUNTIL "23:10:30"  ; pause until 11:10:30pm.
```

When

Print

Syntax

WHEN <waitString>,<resonseString>

WHEN <waitString>,""

Description

Sends a response string to the remote system whenever a given string is matched.

The WHEN statement is usually used in the beginning of the script file and it is active until the end of the script file.

Whenever the <waitString> is received from the remote system, the <responseString> is transmitted. The response string can be changed. To cancel a WHEN statement, "" should be put in in <responseString>.

Case is not sensitive when comparing the <waitString> with the characters received from the remote system.

See also

Waitfor, WhenIdle

Example

```
WHEN "Press ENTER","^M"     ; set response strings
WHEN "More [y,n]?","y^M"
WAITFOR "main menu",0       ; wait for "main menu"
WHEN "Press ENTER",""       ; cancel response string
WHEN "More [y,n]?","n^M"    ; change response string
REPEAT                      ; make the WHEN statements
UNTIL NOT CONNECTED         ; active until disconnected.
```

WhenIdle

Print

Syntax

WHENIDLE t,s

Description

Sends a string to the remote system if there is no COM Input/Output in the specified time.

WHENIDLE monitors the COM Input/Output and sends the string <s> to the remote system if COM I/O is idle for the specified time <t>, in second.

Like the WHEN statement, WHENIDLE is active until the end of the script program.

<s> can contain macro sequence, such as "^*", the modem hangup string.

If <t> is equal to 0 or <s> is a empty string, the WHENIDLE statement will be cancel.

See also

Waitfor, When

Example

```
WAITFOR "message command",0 ; wait for command prompt
PUT "r"                     ; read message
WHENIDLE 30," ^H"           ; prevent inactive timeout
REPEAT                      ; make the WHENIDLE statement
UNTIL NOT CONNECTED         ; active until disconnected

; There are several usages of this command. The above example
; shows the way to prevent inactive timeout. However, it can
; also be used to handle inactive timeout in the host script
; mode.  For example,

WHENIDLE 180,"Inactive timeout^M^J~^*"

; this command checks if the COM I/O is idle for 3 minuates,
; sends a timeout message to the user and, finally, sends the
; hangup string, the hangup macro sequence "^*", to the modem.

; Sometimes the phone line is too noisy that some characters
; cannot be received correctly. The worst case is that those
; characters appear in the WAITFOR string. To prevent inactive
; timeout or waste of connect time, the WHENIDLE can be used as
; follows.

WHENIDLE 5,"^M"             ; send a carriage return if idle
                            ; for 5 seconds
WAITFOR "first name"        ; now start the log on sequence
PUT "^!"
WAITFOR "password"          ; send password macro
PUT "^&"
WHENIDLE 0,""               ; cancel the WHENIDLE statement
```

Write

Print

Syntax

WRITE

WRITE s

WRITE i

WRITE s1,s2,i1,i2,s3, ...

WRITE s1,s2,i1,i2,s3, ... ,

Description

Writes integers or strings to a text file.

Like the PRINT statement, WRITE sends variables or constants of types integer or string to the remote system. Each two arguments are separated by a comma. Integers are converted to their string representation automatically before transmitting to the remote system.

WRITE supplies a carriage return [Ctrl M] and a line feed [Ctrl J] by default. If a comma follows, no carriage return and line feed is supplied.

Return value

Upon successful completion, SUCCESS is set to TRUE; otherwise, SUCCESS is FALSE.

See also

Read, Seek, Tell

Example

```
CREATE "COUNT"      ; write a hundred line into "COUNT"
counter = 1         ; initiate the counter
WHILE counter<=100  ; repeat 100 times
   RITE "This is line ",counter
   NDWHILE
CLOSE               ; close the file
```