Script Files

Important: To be used, a script file must be configured in a Script profile.

Softerm's script files act much like the operating system's batch and command files by automating repetitious operations. The script language is optimized for communications tasks. Because communications operations involve data, Softerm's Script language includes many file and data manipulation functions.

Script files are standard text files and can be created using almost any editor or word processing program which can create and save files in standard ASCII format. A description of the command syntax is included in this section.

Script Language Capabilities

The Script language format is constructed from the basic capabilities typically associated with programming languages. These capabilities include:

Functions
- File transfer, disk file manipulation, error processing, operator input

Text String Manipulation
- Maximum text string length of 255 characters
- Text string catenation

Numeric Operations
- Numeric range of -2,147,483,648 to +2,147,483,647
- Operations including Not, Addition, Subtraction, Multiplication, Division, Arithmetic AND, Arithmetic OR, Arithmetic XOR, Assignment

Variable Manipulation
- Assignment

Conditional Processing
- IF/ELSE, Less Than, Less Than or Equal To, Equal To, Not Equal To, Greater Than or Equal To, Greater Than, Logical AND, Logical OR

Script File Operation

When a script file is started, a window opens and, depending upon the setting of script window control functions, displays some or all of this information:

- A title that contains the Session name and file name of the running script.
- A menu that provides control over the script file operation:
  - Cancel script operation
  - Start and Cancel the Watch script function operation
- A client area where script commands are displayed during Watch and where the MESSAGE command displays text.
- The current path being used by the Script.
- The full path and file name of the active script.

Operation Message File
When operation of a script file starts and ends, appropriate information text messages are written to the file, SOFTERM.MSG, in the user's SOFTERM write file directory. This information typically will contain the following:

- **o** Session name
- **o** Script file name
- **o** Initiation message
- **o** Operation error message (if applicable)
- **o** Statement where operation error occurred
- **o** Stop message: normally or by operator

In Case of Error
When a function produces an error other than a simple file or data processing error, one of the following will occur:

1. Fatal errors will cause script file operation to be cancelled.
2. Non-fatal errors will cause script file operation to continue at the defined error processing label (determined by the ONERR function) or, if no error processing label is defined, script file operation will be cancelled. Read-only system variables are available that will contain the error code, error level and error text for the implementation of suitable error recovery procedures.

Script Language Syntax

This section provides a brief overview of the Softerm script language. Each component also is described fully in separate sections.

Syntax Rules

1. An input line consists entirely of the data between one '>' character which is in the first column and the next '>' character which is in the first column. The exception is, of course, the last command in the script file.

2. An input line may contain only a single script language statement. However, a statement may consist of multiple functions.

3. Text strings and variables containing text strings are interchangeable throughout the script language.

4. Numerics and variables containing numerics are interchangeable throughout the script language.

Variable Classification

Softerm's script language recognizes three classifications of variables:

- **o** User-defined variables, such as:
  > TXTVAR = 'This is a test.'
  > NUMVAR = 42

- **o** Assignable predefined system variables, such as:
  > BAUD = 9600

- **o** Read-only predefined system variables, such as:
  > IF ERRORLEVEL == 6

Function Identification

Functions are identified as such by enclosing their arguments between parentheses. This includes those functions that require no arguments. For example:

```
> EXAMPLE()      ;is a function
> EXAMPLE        ;is a variable
```

Result Codes
All functions return a result so that relatively complex and meaningful statements may be constructed. For example, while the following command is valid, little can be done with it:

```
> NEXTFILE (filedef)
```

However, by using the returned result, a useful statement is created, such as:

```
> SFILE = NEXTFILE (filedef)
> IF GETLENGTH (SFILE) < 5
> GOTO NO_MORE
> ENDIF
```

Important: Except where noted, this result code will be a zero value for success or an error code value in the case of a failure.

Arguments
A function argument may be any valid expression. For example:

```
> WRITELINE(HANDLE,'Error code = ' + VALTOSTR(ERRORCODE))
```

Function arguments must be separated with commas. Function arguments that have predefined default values may be specified as null. For example:

```
> EXAMPLE(arg1,,,arg4)
```

Trailing function arguments that have predefined default values may be omitted.

Remarks
A remark is all remaining data within a statement that follows a semi-colon.

```
> EXAMPLE(arg1,arg2)    ;this is a remark
```

Labels
A label that is the target of a discontinuity is signified by a colon following the label operand, such as:

```
> ERRORLABEL:
> BEGIN:
```

Label names are not case sensitive, but they must begin with a letter. Only the first 10 characters are significant.

User Variable
User variables are not case sensitive, but they must begin with a letter.

Script File Format

Script files exist on disk as text files. Within the text file, a function is defined by a key word and possible arguments. The general format of a command in a script file is:

> RESULT = KEYWORD (arg1,arg2,...,argn)

The > character is required to identify the line as the start of a statement and must be the first non-blank, non-tab character on the line. Any other character occurring as the first non-blank, non-tab character on the line indicates that the line is a continuation of a previous command. Carriage returns and line feeds and any blank or tab characters following the carriage return or line feed are considered to be white space and may be used to separate key words, arguments, and switches.

Inserting Special Functions
The acronyms listed in Appendix A, ASCII Character Codes, and the tilde (~) character are used to insert characters which normally are interpreted as special functions into an argument. Command lines can contain any of the 128 ASCII characters, including control characters. For example, the following character sequences can be used to allow function characters to be inserted in strings:

Sequence	Function
[CR]	Insert a carriage return into strings
[LF]	Insert a line feed into strings
[ddd]	Insert a character into strings where **ddd** is the decimal ASCII value of the character from 0-255. Up to 3 decimal digits can be used. For example, [7] will insert a BEEP.
~any char	Insert characters into strings. For example, use ~~ to insert a tilde and ~[to insert a bracket. **Note:** The tilde prefix can be redefined by the ESCAPE assignable predefined system variable for use within Script functions.

Note: The conversion from internal nomenclature to destination character is performed only when the text is prepared for transmission, such as by the Send, Receive and Xmit_Wait functions.

Directory Paths and File Names
File names used with these commands may include a drive specifier and a complete directory path from the root directory of the drive, including the special symbols back slash (\), period (.), or double period (..). If no directory path is specified, the directory path displayed on the bottom row of the progress frame window will be used. If the directory path specified does not begin with the root directory, Softerm assumes the directory path specified begins with the current default directory.

Wild card Characters
Softerm allows global or wild card characters to be used in file names for most file transfer commands performed by Softerm. The wild card characters asterisk (*) and question mark (?) indicate the portion of a file name which may be ignored or which may match any series of characters. The asterisk is used to match any string of characters and the question mark is used to match single characters. An * used alone or *.* will match all file names. An *. will match only file names with no extension.

The use of file name wild card characters is identical to the operation of these characters when specified for standard DOS operations. The command will be performed on all files whose file names meet the subset specification.

Script File Language

Softerm's script file script language consists of three basic groups:

1. Variables and Parameters
2. Functions
3. Directives

Variables and Parameters
The Softerm script language can use string variables, consisting of sequences of alphanumeric text up to 255 characters long, and numeric variables, consisting of an integer from -2,147,483,648 through +2,147,483,647.

Characters in a string are numbered from left to right beginning with 1. A number of functions are defined for the manipulation of the variables.

Important: If you set a variable equal to a numeric value, remember that Softerm stores numeric values in binary format; not decimal. Refer to the description of the VALTOSTR() function which converts binary data to decimal.

Variables are referenced by a name which consists of an alphanumeric string between 1 and 255 characters in length and which begins with an alpha character.

Note: The space used to store variable names and values is limited. If many variables are used in a script file, the length of variable names should be kept to a minimum.

Script Variables
Script Variables, also called Parameters, are a special type of string variable. These text variables are used to import values to, and export values from, script files.

Softerm recognizes five Script Variables, each of which can be 64 characters long. Values for these parameters can be assigned within script files. In addition, these parameters can be defined when a script file is initiated and used by the script file as arguments to modify the operation of the script file. For additional information, please refer to the Script File Profile and Session Window: Options Menu chapters.

Script Variables are referenced by the special symbols **SV1** through **SV5**, representing parameter 1 through parameter 5, respectively. When encountered in a script file, the value of the specified Script Variable is substituted in the script file for the parameter name before the command containing the parameter is evaluated. This allows the modification of Script Variable values by the script file.

When a **CHAIN** command is performed to stop the current script file and start another one, the new script receives all the defined variables. This allows the values of variables from one script file to be passed to another script file. All variables, except the **ERRORCODE** variable and the Script Variables, may be cleared with the VCLEAR command. This allows unused variable storage to be reclaimed.

Expressions
An expression is a series of one or more variables, constants or literals (operands) separated by operators. The evaluation of an expression yields either a string or a numeric result, depending on the type of operators in the expression.

Constants are numeric values and may be combined with other constants or numeric variables in

an expression.

Literals are string values and may be combined with other literals or string variables in an expression.

The following command illustrates the definition of a numeric variable:

```
> NUMBER = 5
```

The result of this command is to define NUMBER as a numeric variable and assign it the constant value of 5. The following commands do the same thing:

```
> NUMBER = 2
> NUMBER = 10 / NUMBER  ; / signifies division.
```

The following illustrates the definition of a string variable:

```
> STRING = "Hello, Mikey"
```

The result of this command is to define STRING as a string variable and assign it the literal value of "Hello, Mikey". The following command does the same:

```
> STRING = 'Hello, ' + 'Mikey'
```

The plus sign is a catenation operator, used to combine the two strings.

Expression Evaluation Order
Expressions are evaluated in left to right order (there is no operator precedence), unless parentheses are used to alter the evaluation order. Using parentheses will force the evaluation of the expression inside the parentheses before combining it with the operand to the left of the parenthesized expression.

Numeric Constants
Numeric constants may be integer decimal, binary, octal, and hexadecimal numbers and may range in value from -2,147,483,648 through +2,147,483,647. The format of a numeric constant is:

Type	Format	Example
Decimal	_nnnn	15
Octal	_0nnnnO	017O
Hexadecimal	_0nnnnH	0FH
Binary	_0nnnnB	01111B

Important: A numeric constant not beginning with 0 is treated as a decimal.

Literals
Literals are strings enclosed in single (') or double quotes ("). They may contain any alphanumeric characters and may be up to 255 characters in length.

Operators
! Logical Not **+** Plus
- Minus ***** Multiply
/ Divide **&** Arithmetic AND
| Arithmetic OR **^** Arithmetic Exclusive OR
< Less Than **>** Greater Than

= Assignment
== Identical To
&& Logical AND
|| Logical OR
<= Less Than or Equal To
>= Greater Than or Equal To
!= Not Identical To

Unary Arithmetic Operators
The **!**, **+** and **-** symbols are unary arithmetic operators. They affect the value associated with the constant or numeric variable which follows the symbol.

! Invokes the NOT function, causing a boolean True/False evaluation of the expression following the not operator (**!**). A non-zero result is evaluated as False; a zero result is evaluated as True. For example, IF !EXIST (filename) will perform a set of instructions if the file is not found.

+ Is optional on all numeric values, and implies normal interpretation of the value.

- Causes a numeric value to be negated, transforming a positive value to a negative value, or a negative value to a positive one.

Binary Arithmetic Operators
The **+** and **-** signs also may be binary operators.

- **+** Addition
- **-** Subtraction
- ***** Multiplication
- **/** Division
- **&** Arithmetic AND function; the two operands are ANDed bit by bit to produce the resultant numeric value.
- **|** Arithmetic OR function; performs an inclusive OR on the operands.
- **^** Performs an arithmetic exclusive OR of the two operands.

Binary String Operators
The **+** sign also is a binary string operator. Separating two string values, either literal or variable, causes the catenation of the two strings to form the resultant string value.

Relational Operators
Relational operators, also called logical operators, are used in the IF directive to test the relation between numeric or string variables. When comparing strings, the operation is not case sensitive; that is, 'abc' is equivalent to 'ABC'. The result of a relational operation is either True (any non-zero result) or False (a zero result).

For example:

```
> TYPE = 'BEETLE'
> IF MAKER == 'FORD' && TYPE == 'MUSTANG'
```

yields a False value, while:

```
> IF (ROW >= 24) || (COLUMN >= 79)
> GOTO CONTINUE
> ENDIF
```

yields a True value when you reach row 25 or column 80 of the screen. Note: Screen coordinates are zero relative. That is, the point 0,0 defines the top left corner.

File Transfer Errorlevel Codes

These codes are returned to the Script file and can be used for error processing at an ONERR label. For additional information, refer to the ERRORLEVEL, ERRORTEXT, GETERRORTEXT(), IF(), and ONERR system variables, directives and functions.

Code	Type Of Error
0	None
1	Timeout
2	Line failure (Retry count expired or loss of carrier)
3	Operator cancel
4	Remote cancel
5	DOS error
6	Expression evaluation error

Variable Names
Variable names can contain only alphanumeric characters (A-Z, 0-9) and must begin with an alpha character (A-Z). Softerm is case insensitive when searching for variable names; there is no distinction between lower case and upper case. Thus, number and NUMBER refer to the same variable. Softerm decides that a string refers to a variable name rather than a literal string by whether or not it is enclosed in quotation marks.

System Variables and Reserved Names

Softerm includes special variables which allow you to access system information from within script files for use in expressions.

Important: These names are reserved for use by Softerm and must not be used as variable names. Very unpredictable results can occur. Assignable Predefined System Variables
These system variables can be used on the left side of an equality and set to a valid value. For example:

> DATABITS = 7

BAUD
Numeric variable which can be set to a valid baud rate in the range 50 through 57600, such as: > BAUD = 9600

DATABITS
Numeric variable which can be set to a value in the range 7 through 8, such as:
> DATABITS = 7

ESCAPE
Text variable which defines the lead-in character to be used. This variable defaults to the tilde (~) character. Additional information is available in the Inserting Special Functions heading above.

PARITY
Numeric variable which can be set to one of the following values: NONE, ODD, EVEN, MARK, SPACE, such as: > PARITY = NONE

RETRIES
Specify the maximum retry count for error conditions using all file transfer protocols

except Character protocol. RETRIES defaults to 3.

Possible error conditions include timeouts, block check errors, or any other error condition such as an inappropriate reply to a message.

The maximum retry count is specified as an argument, such as > RETRIES 5. A value from 0 through 255 may be entered (although a value of 0 is equivalent to a value of 1).

If an error condition occurs during a file transfer operation, and the RETRIES count is exhausted, the message Line Failure will be displayed.

RXPACING

Numeric variable which can be set to one of the following values: NONE, XON_XOFF, DTR, RTS, such as: > RXPACING = XON_XOFF

SPEED

Numeric variable which can be set to a value in the range 0 through 255 milliseconds, such as: > SPEED = 25

The default value is 0. The SPEED variable defines a transmit character delay which allows a delay between characters transmitted on the communications line. This variable could be used when the host computer is not able to accept characters as fast as the actual line speed allows.

STOPBITS

Numeric variable which can be set to 1 or 2.

TIMEOUT

Numeric variable which can be set to a value in the range 0 through 65535 seconds. The default value is 15. The TIMEOUT variable specifies the delay tolerated before cancelling a DIAL or XMIT_WAIT operation. A value of 0 indicates that there is an unlimited timeout period.

TXPACING

Numeric variable which can be set to one of the following values: NONE, XON_XOFF, XON_XOFF_PAIRS, CTS, DSR, DCD

Read-Only Predefined System Variables

These system variables are not user-settable. Instead, they are updated as required and, as a general rule, are used in conditional processing. For example:

> IF ERRORLEVEL == 3

CONNECT

Logical variable which may be tested TRUE or FALSE indicating if a connection currently is established.

CTS

Numeric variable which may be used to set TXPACING, such as: > TXPACING = CTS

DATE

String variable that holds the current date in MM/DD/YY format when referenced.

DAY
Numeric variable that holds the current day (1-31).

DCD
Numeric variable which may be used to set TXPACING, such as: > TXPACING = DCD

DSR
Numeric variable which may be used to set TXPACING, such as: > TXPACING = DSR

DTR
Numeric variable which may be used to set RXPACING, such as: > RXPACING = DTR

ERRORCODE
Numeric variable that contains the error value of the last function that terminated with a serious error. This variable will be set correctly only during ONERR processing.

ERRORLEVEL
Numeric variable in the range 0 through 6 which contains the Errorlevel Code (discussed above and in the description of the IF() function). This variable will be set correctly only during ONERR processing.

ERRORTEXT
Text variable which contains the text related to the ERRORCODE. ERRORTEXT may be displayed using the MESSAGE function. This variable will be set correctly only during ONERR processing.

FALSE
Opposite of TRUE. An expression which evaluates to a zero result is FALSE.

HOUR
Numeric variable that holds the current hour (0-23)

MINUTE
Numeric variable that holds the current minute (0-59).

MONTH
Numeric variable that holds the current month (1-12).

NONE
Numeric variable which may be used to set TXPACING and RXPACING, such as:
> RXPACING = NONE

OFF
Numeric variable with the value of zero. (False)

ON
Numeric variable with the value of one. (True)

PATH
String variable that holds the full path of the current directory.

PORT
Text variable that contains the name of the Connection Path used by the currently-running script file.

RETRIES
Numeric variable that defines the maximum retry count for a DIAL operation.

RTS
Numeric variable which may be used to set RXPACING, such as: > RXPACING = RTS

RX7MASK
Specifies whether received data will or will not be masked to 7bits (ie the lower ASCII character set). This variable defaults to YES (will be masked). If the session's terminal emulation has been restarted by use of the TERMINAL() function, masking of receive data will be controlled by the terminal emulation and not this variable.

RXDATA
Text variable containing the last 255 characters received by the most recent XMIT_WAIT function.

SECOND
Numeric variable which holds the current second (0-59)

SESSIONNAME
Text variable which holds the name of the current Session as displayed in the Session Manager window.

TIME
String variable that holds the current time in HH:MM 24-hour format.

TRUE
Used to test the result of a logical expression. An expression which evaluates to a non-zero result is TRUE. When it is returned from a function, TRUE will have the value of 1.

USERNAME
If Host Mode optional logon processing is in effect, this string variable holds the current user name of 32 characters, blank filled.

WEEKDAY
Numeric variable which contains the day of the week in the form 0 (Sunday) through 6 (Saturday).

XON_XOFF
Numeric variable which may be used to set TXPACING and RXPACING, such as:
> TXPACING = XON_XOFF

XON_XOFF_PAIRS
Numeric variable which may be used to set TXPACING, such as:
> TXPACING = XON_XOFF_PAIRS

YEAR
Numeric variable that holds the current year.

Script Functions and Directives

Softerm file transfers are controlled by the following directives and functions which may require additional parameters:

(semi-colon)	Comment
: (colon)	Label (name) a command sequence location
AUTOANSWER()	Enable and disable an auto-dial modem's Auto-Answer mode
BREAK()	Transmit a communications break signal
CANCEL	Cancel the operation of a script file
CHAIN()	Transfer script operation to a new script file
CHDIR() or **CD()**	Change default directory
CHRTOVAL()	Returns the numeric value of any ASCII character (for example, '1' is returned as 49)
CLOSE()	Close a specified open disk file
CLOSEALL()	Close all open files
CONVERSE()	Exit to on-line operation
COPY()	Copy a file
DELETE()	Delete a file
DIAL()	Establish a communications connection
ELSE	Alternate conditional processing
END	End of script file
ENDIF	End conditional processing.
EXIST()	Check for the existence of a particular file.
EXIT	Terminate the current script file and the current runtime session.
FINDLINE()	Search an open file for a particular text string.
FIRSTFILE()	Return information about the first file found in the given path/file name template.
FIXLENGTH()	Sets the length of a text string by padding with spaces or by truncating.
GETERRORTEXT()	Converts any numeric value to its related error text (if there is any). In particular, this would be used to create the error text for a result code which is returned by a function.

GETENVSTR() Gets a text string from the current runtime environment space.

GETLENGTH() Returns the numeric value of the length of a text string.

GOTO or JUMP Go to a Label.

HANGUP() Disconnect, lower DTR.

HOST() Initiate a Host mode -- unattended interaction

IF Begin conditional processing.

INPUT() Accept operator text input to the script.

LOG() Begin logging.

LOWER() Convert a string to all lower-case.

MESSAGE() Displays a message in the progress frame window, no wait.

MKDIR() Create a disk directory.

NEXTFILE() Returns an information text string for the next file matching the file name template specified in a FIRSTFILE function.

NOLOG() End logging.

ON() Branch in a Script file based on a received value.

ONERR Use this directive to provide a label to which Script processing will be redirected in the event of a serious error which otherwise would cause Script operation to terminate.

OPENNEW() Create and open a new disk file.

OPENOLD() Open an existing disk file.

PAUSE() Delay n seconds.

PROMPT() Displays an information message box until cancelled by the operator.

READ() Returns a text string from an open file.

READLINE() Returns a line of text from an open file.

RECEIVE() Transfer a file from a host to your PC.

RENAME() Rename a file.

RESUME Resume operation after error.

RETRY Retry last command.

RSTATTR() Reset file attributes.

SEND() Transfer a file from your PC to a host.

SETATTR() Set a file's attributes.

SETBOF() During file operations, sets the position of the internal file pointer to the beginning of the file.

SETEOF() During file operations, sets the position of the internal file pointer to the end of the file.

SKIP() During file operations, sets the position of the internal file pointer relative to the current file position.

STRFIND1() Returns the numeric position of the start of a specified text string within a text string.

STRFIND2() Returns the numeric position of any character from a specified text list within a text string.

STRGET1() Extracts a text string from a text string by length.

STRGET2() Extracts a text string from a text string terminated by a specific terminator character from a text string.

STRPUT() Replaces a text substring within a text string.

STRTOVAL() Returns the numeric value representing an ASCII numeric text string (for example, '456' is returned as 456).

TERMINAL() Allows the terminal emulation of the current session to be resumed or suspended.

UPPER() Convert a string to all upper-case.

VALTOCHR() Returns the ASCII character representing a numeric value (for example, 49 is returned as '1').

VALTOSTR() Returns the ASCII numeric text string representing a numeric value.

VCLEAR() Clear all variables.

VDEFINED() Check if a variable has been defined.

VDELETE() Clear a particular variable.

WAIT() Wait for an event and return a code.

WRITE() Writes a text string to an open file.

WRITELINE() Writes a line of text to an open file.

XMIT_WAIT() Transmit a string, wait for a reply.

Script Window Control Functions

These functions control the separate window which may be displayed by a script file. They are explained at the end of this chapter:

WATCH()	Display script commands and current file and directory names in the script window.
WINDOWPOS()	Set the upper left corner of a script file window.
WINDOWPRMS()	Set menu and display options to be used by a script window.
WINDOWSHOW()	Display a user-defined script window.
WINDOWSIZE()	Set the size of the client area of a script file window
WINDOWTITLE()	Provide a title for a script file window.

Script Functions Descriptions

AUTOANSWER (arg1)

Function: Enable or Disable an auto-dial modem's Auto-Answer mode

Syntax: > AUTOANSWER (on)
> AUTOANSWER (off)

If this command is processed when a connection has already been established, an error message similar to the following will be generated: "Can not Change Mode while Connected".

BREAK()

Function: Send a communications break signal of 255 milliseconds duration.

Syntax: > BREAK()

BREAK (arg1)

Function: Send a communications break signal for a specified number of seconds.

Syntax: > BREAK (seconds)

The argument may be a numeric value in the range 1 - 9 which specifies of the break signal duration in seconds.

CANCEL
Function: Cancel further script file processing.

Syntax: > CANCEL

This directive usually is used in combination with the ONERR directive and IF ERRORLEVEL conditional processing when it is desired to terminate processing of the script file due to the type of error encountered.

The current script file and log file are closed when a CANCEL command is processed.

CHAIN (arg1)

Function: Terminate the operation of the current script file and start another.

Syntax: > CHAIN (path_scriptname)

where:

PATH_SCRIPTNAME is the text string or defined variable providing the location and name of the script file to run.

Example:

> CHAIN ('c:\softerm\scripts\sethost.scr')

Use when a single script file cannot contain all the commands required, or it is desired to segment the operation of multiple operations. All file transfer variables including the variable heap, error processing retry count, transmit character delay, and timeout value are unchanged after the CHAIN. Logging will remain enabled if active.

Wild card characters are not allowed in the file name entered. When a CHAIN command is used, if the specified script file does not exist, an error message is displayed; the command is cancelled; and operation of the current script file is terminated.

CHDIR (arg1) and **CD (arg1)**

Function: Set a new default directory path.

Syntax: > CD (pathname)

where:

PATHNAME is the text string or defined variable providing the drive and directory to which to change.

Example:

> CD ('s:\database')

The new default directory including a drive specifier and directory path is entered as an argument on the command line. File names entered for file transfer commands may include a drive specifier and a complete directory path from the root directory of the drive. If no directory path is specified in a file name, the current default directory path will be used. If the directory path specified in a file name does not begin with the root directory, Softerm assumes the directory path specified begins with the current default directory.

CHRTOVAL (arg1)

Function: Returns the numeric value of any ASCII character (for example, '1' is returned as 49).

Syntax: > CHRTOVAL (ascii)

where:

ASCII is the ASCII character.

Example:

> NUM_VAL = CHRTOVAL ('Z')

The numeric variable NUM_VAL will contain the decimal value 90.

The CHRTOVAL function can be used to extract control codes from text variables. For instance, a Ctrl A would be returned as 1. CHRTOVAL also can be used to compute checksums (the sum of the ASCII values of each character in a string).

CLOSE (arg1)

Function: Close an open file.

Syntax: > CLOSE (file_handle)

where:

FILE_HANDLE must be the name of a variable that contains the handle returned by a previous file open function.

CLOSEALL()

Function: Close all open files.

Syntax: > CLOSEALL()

CONVERSE (arg1)

Function: Return to the on-line terminal operation mode; stopping the current script file.

Syntax: > CONVERSE (initial_string)

where:

INITIAL_STRING is an optional string (maximum 35 characters) which may be entered as initial keyboard input to the on-line terminal mode.

Example:

> CONVERSE ([CR])

The CONVERSE command is useful when an immediate return to the on-line terminal mode is required from a script file. The CONVERSE command is not valid when used in a script file run when terminal mode is not available.

COPY (arg1,arg2,arg3)

Function: Copy a file.

Syntax: > COPY (source,dest,if_exist)

where:

SOURCE is the source path and file name text string.
DEST is the destination path and file name text string.
IF_EXIST is the option to use if a file with the DEST name already exists. The options are:

Replace Overwrite the existing file with the new file.
Append Add the contents of the new file to the end of the existing file. This generally is

not a appropriate option to use with binary files.
Fail Cancel the Copy operation. This is the default.

Example:

```
> COPY ('c:\softerm\my.mdb', 's:\softerm\public\special.mdb', replace)
```

or

```
> SV3 = 'c:\softerm\my.mdb'
> SV4 = 's:\softerm\public\special.mdb'
> COPY (SV3,SV4,)
```

The destination file date and time stamp will be set to the date and time stamp of the original source file.

The source and destination filename templates may contain the global filename characters '*' and '?'.

Example:

```
> COPY ("\source\*.c','source\backup\*.c',FAIL)
```

The source and destination filename templates may specify a directory path only to request that all files in the source directory be copied to the destination directory.

Example:

```
> COPY ('\source','\source\backup',FAIL)
```

DELETE (arg1)

Function: Delete a file from the current or specified directory.

Syntax: > DELETE (path_filename)

where:

PATH_FILENAME is the text string or defined variable providing the location and name of the file to delete.

Example:

```
> DELETE ('c:\softerm\softemp.999')
```

If no drive or directory path is entered, the file is deleted from the current directory. You can use the wild card characters (?) and (*) in the file name and in the extension.

If the file does not exist, no error is reported and command processing will continue. All other DOS errors will cause error processing to be performed.

DIAL (arg1,arg2,arg3,arg4)

Function: Establish a communications connection.

Syntax: > DIAL (handshake_trans,phone_num1,telnet_id,phone_num2)

Note: The number of arguments and their types depend on the current connection path type and the applicable Admittance Data dialog.

where:

HANDSHAKE_TRANS is a string which will be transmitted automatically when a connection is made. If no string is entered, this argument defaults to either:

- The string used by a previous DIAL command, or
- The Handshake Transmission string as defined in the Admittance Data configuration for the current Session if no previous DIAL command has been issued.

PHONE_NUM is the Phone Number or equivalent, such as a Called DTE Address or Service Name. If no string is entered, this argument defaults to either:

- The string used by a previous DIAL command, or
- The field which precedes the Handshake Transmission field defined in the Admittance Data configuration for the current Session if no previous DIAL command has been issued.

Note: If the Admittance Data configuration has only the Handshake Transmission field, the DIAL command will have only one argument.

TELNET_ID is the Telephone Network ID string. If no string is entered, this argument defaults to either:

- The string used by a previous DIAL command, or
- The Telephone Network profile defined in the Admittance Data configuration for the current Session if no previous DIAL command has been issued.

Note: If the current Admittance Data configuration does not have a Telephone Network profile field, this argument is not applicable.

PHONE_NUM2 is the second telephone number (or equivalent).

When using an IBM acs connection path, it may be necessary to specify a CBX Datagroup and the Account ID. The DIAL() function might resemble:

```
> DIAL (,CBXData_Group,,Accnt_ID)
```

When using a Tcp/Ip connection path it may be necessary to specify a host port to be used other than the default of 23. The DIAL() function might resemble:

```
> DIAL (,'softgate',,'24')
```

These examples does not specify a handshake transmission or a telephone network profile name.

Important: If an argument is not specified, DIAL will attempt to make the connection using either the value contained in the current Admittance Data configuration field or the last value which was used, whichever is more current. To ensure no value is used for an argument, place a null-length string in the argument's position. For example, the following command would dial the Softronics' Customer Service BBS without transmitting an initial string and without using a Telephone Network ID:

```
> DIAL ('','17195939530','')
```

The DIAL function uses the values of the current TIMEOUT and RETRIES parameters. For example, if TIMEOUT is set to 45 seconds and RETRIES is set to 8, the DIAL function will attempt to make the connection 8 times and will wait 45 seconds each time before cancelling the operation.

Note: Values for TIMEOUT less than 30 seconds are defaulted to 30 seconds for the DIAL function.

ELSE

Function: Specify alternate conditional processing when used in conjunction with a previous IF command.

Syntax: > ELSE

If the conditional result of an IF command is true, all commands following the IF command are processed until an ELSE or the corresponding ENDIF command is encountered. Commands following an ELSE command up to the corresponding ENDIF command are ignored.

If the conditional result of an IF command is false, all commands following the IF are ignored until an ELSE command or ENDIF command is encountered. Commands following the ELSE command are processed until the ENDIF command corresponding to the IF command is reached.

This command is ignored if there is no previous corresponding IF command. The following example demonstrates how the ELSE command is used:

```
> IF conditional expression
>
>   commands processed if condition is true
>
> ELSE
>
>   commands processed if condition is false
>
> ENDIF
```

For example:

```
> IF RESPONSE == 'Y'
> GOTO DO-IT
> ELSE
> NUMVAR = 1
> CHAIN ('NORESP.SCR')
> END
> ENDIF
```

END

Function: Terminate the operation of a script file.

Syntax: > END

The current script file and any log file are closed when an END command is processed.

ENDIF

Function: Terminate the conditional processing of a previous corresponding IF statement and resume normal command processing.

Syntax: > ENDIF

This command is ignored if there is no previous corresponding IF command.

EXIST (arg1)

Function: Test for the existence of a file and return a TRUE or FALSE result.

Syntax: > EXIST (path_filename)

where:

PATH_FILENAME is the text string or defined variable providing the location and name of the file for which you are checking.

If no drive or directory path is entered, the test is performed in the current directory.

This function usually is used in an IF statement, such as

```
> IF EXIST ('v:\database\test.dbf')
> ; do one thing
> ELSE
> ; do something else
> ENDIF
```

The following example also is valid:

```
> FILENAME = 'c:\softerm\manual.txt'
> VALUE = EXIST (filename)
> IF VALUE == TRUE
> ...
> ENDIF
```

EXIT

Function: Terminate the current script file and the current runtime session.

Syntax: > EXIT

This is similar to the END directive, except that EXIT also terminates the current runtime session.

FINDLINE (arg1,arg2)

Function: Search an open file for a particular text string and return a TRUE or FALSE value result.

Syntax: > RESULT = FINDLINE (filedef,search_string)

where:

FILEDEF must be the name of a variable that contains the handle returned by a previous file open function.

SEARCH_STRING is the text string or the variable containing the text string to locate.

This function will treat the open file defined by the file definition variable, FILDEF, as a line-type file and will search, from the current position of the pointer in the file, each line of text for the match string.

If the value of RESULT is TRUE, the position of the internal file pointer will have been set to the first character of the line containing the specified text.

```
> NAME = 'JONES, JOHN'
> RESULT = FINDLINE (FILDEF,NAME)
```

On completion of the command, a result code is returned to the variable RESULT. A result code of True means the search was successful; a False value indicates that the command was not successful. If the returned result is True, the current file position will be set to the first character of the text line containing the match string.

The statement:

```
> IF (FINDLINE(FILDEF,VName)) == FALSE
```

will search from the current position in the open file for the value contained in the variable VName.

(VName must have been defined by a previous statement, such as VName = 'Chevrolet', or the value must have been read into it by a previous READ or READLINE function).

If the value is not found (the returned value of the operation is FALSE), you can process an error-handling or branching routine. For instance, it could mean that the value assigned to the variable VName has been entered incorrectly; or the internal file pointer is past the point in the file where the value assigned to VName is.

FIRSTFILE (arg1,arg2)

Function: Return information about the first file found in the given path/file name template.

Syntax: > FIRSTFILE (template,filedef)

where:

TEMPLATE is the file name template text string.

FILEDEF must be the name of a variable to receive a unique identifier value that will be used as

an argument in any subsequent NEXTFILE functions.

Examples:

```
> INFOSTRING = FIRSTFILE('C:\*.*',FILDEF)
    or
> FILENAME = 'C:\*.*'
> INFOSTRING = FIRSTFILE(FILENAME,FILDEF)
```

The returned text variable (INFOSTRING) will be a null length string if no matching file name is found or will contain a fixed-format text string as follows:

bytes:	
1-12	file name
13	blank
14-21	file size
22	blank
23-30	file date in mm-dd-yy format
31	blank
32-36	file time in hh:mm format
37	blank
38	A attribute character, archived file
39	D attribute character, file name is a directory
40	V attribute character, file name is a volume name
41	S attribute character, system file
42	H attribute character, hidden file
43	R attribute character, read-only file

The internal resources allocated for the unique identifier are released automatically under the following circumstances:

1. When no file is available for this FIRSTFILE function.
2. When no more files are available to a subsequent NEXTFILE function.
3. By processing a CLOSE(filedef) function.
4. By processing a CLOSEALL() function.

FIXLENGTH (arg1,arg2)

Function: Sets the length of a text string by padding with spaces or by truncating.

Syntax: > FIXLENGTH (source_string,length)

where:

SOURCE_STRING is the source text string.

LENGTH is the numeric value of the length.

Examples:

```
> SRCSTR = 'Enter File Name"
> OUTPUTSTRING = FIXLENGTH (SRCSTR,20) + ':'
    or
> LEN = 20
```

> OUTPUTSTRING = FIXLENGTH (SRCSTR,LEN)

GETERRORTEXT (arg1)

Function: Converts any numeric value to its related error text (if there is any).

Syntax: > GETERRORTEXT (numeric)

where:

NUMERIC can be any numeric value, although the function is designed primarily to use the result codes which are returned by a script function.

Examples:

```
> RESULT = OPENNEW ('TEST.DOC',FILEDEF)
> IF RESULT != 0  ; If file creation failed
 ; We'll display the reason
> MESSAGE (GETERRORTEXT (RESULT))
> ENDIF
  or
> NUMERIC = 63
> MESSAGE (GETERRORTEXT (NUMERIC))
```

Important

1. Not all numeric values have associated error text.
2. Unlike the ERRORLEVEL and ERRORTEXT variables, this function may be used during local error checking, rather than only during ONERR processing.
3. Although the command:

   > OPENNEW ('TEST.DOC',FILEDEF)

   is perfectly legal, you'll notice that there is no variable to hold a returned result code. It is not possible to verify the success or failure of the call.

GETENVSTR (arg1)

Function: Returns a text string from the current runtime environment space

Syntax: > GETENVSTR (env_string)

where:

ENV_STRING is the name of a text variable or a text string representing the name of the environment variable whose text value is to be returned. The value of ENV_STRING is not case sensitive.

Example:

```
SET sendfiles = 'c:\3m\exe\win\*.txt"
WIN SOFTERM /r "sessionname" /p
```

The script file would then include the statement:

```
>  filestosend = GETENVSTR('sendfiles')
```

GETLENGTH (arg1)

Function: Returns the numeric value of the length of a text string.

Syntax: > GETLENGTH (source_string)

where:

SOURCE_STRING is the source text string.

Example:

```
> SRCSTR =  'Softerm Modular -- The best choice for Windows and OS/2'
> STRLENGTH = GETLENGTH (SRCSTR)
```

GOTO label

Function: Transfer command processing to the command immediately following a specified Label.

Syntax: > GOTO Label

Label corresponds to the name specified in a Label. A Label name must begin with an alpha character and only the first 10 characters are significant.

GOTOs may occur in either a forward or backward direction, and processing will continue with the first command following a label whose name matches the GOTO command label. If no label is found with a matching name, operation is cancelled.

Note: A Label is written in the form:

```
> Label_Name:
```

The GOTO command will cancel all conditional processing when used with IF, ELSE, and ENDIF commands.

GOTO and Processing Speed
All GOTO operations cause the script file to return to the beginning and conduct a forward search for the Label. This, naturally, causes a slowdown of the processing time.

You may speed up operation significantly by rearranging the order of the script file so that all initial setup operations are actually placed at the end of the file, and placing a GOTO LabelName (such as BEGIN) at the very beginning and a GOTO OtherLabelName at the end of the initialization sequence returning processing to the physical beginning of the file, such as:

```
> GOTO Begin
> RealOperations:
    .
Sequence of operations involving numerous GOTO
commands
```

.
> Begin:
.
Initialization routine
.
> GOTO RealOperations

HANGUP()

Function: End the current communications connection.

Syntax: > HANGUP()

The DTR (data terminal ready) control signal is low-ered and Softerm waits 3 seconds for the connection to be broken. Then Softerm continues processing of a script file with the next command. Note: When used with a Communications Server, the connection to the service is broken.

Note: After a HANGUP command has been processed, the DTR control signal remains low until a communications command, such as DIAL or XMIT is processed.

HOST (arg1,arg2,arg3,arg4,arg5)

Function: Initiate operation of a host mode environment. Note: This function is described in the chapter, Host Mode.

Syntax: > HOST (idletime,maxtime,term_on,force,script)

where:

IDLETIME is a numeric value in minutes that defines the maximum time which can elapse at the READY prompt without input from the caller. A zero value allows unlimited idle time.

MAXTIME is a numeric value in minutes that defines the maximum connect time allowed per connection. A zero value allows unlimited connect time per connection.

TERM_ON defines the condition under which the Host mode environment will be automatically terminated. Available options are:

NONE (default)
IDLE
MAXTIME
NOCARRIER

FORCE defines whether the Host mode security feature will be forced if the Host mode was initiated when a communications connection already existed. Available options are:

NO (default)
YES

SCRIPT is a text string that defines a script file to be run whenever a new communications connection is established.

The following are valid HOST() commands:

```
> HOST (30,,,)
> HOST (0,,,"C:\SOFTERM\SCR\HOST.SCR")
> HOST (0,NOCARRIER,YES,)
```

IF (expression)

Function: Provide conditional processing within a script file by testing for a True or False evaluation of an expression.

Syntax: > IF expression

Variables, constants, literals and functions may be included in the expression to be evaluated.

Softerm provides a special variable called ERRORLEVEL that allows you to test for any error conditions which may occur during the operation of a script file. This is a reserved variable name that represents the completion state of the last command to be processed. The possible values of the ERRORLEVEL variable are:

Errorlevel	Type of Error
0	None
1	Timeout
2	Line Failure (Retry count expired or loss of carrier)
3	Operator Cancel
4	Remote Cancel
5	DOS Error
6	Expression evaluation error

The ERRORLEVEL variable is used in conjunction with the ONERR directive to provide an error processing routine when an error occurs during the processing of any command. The ERRORLEVEL variable always contains the value of the error code from the previous command. If the ONERR directive is not used, script file operation will be cancelled by default when an error occurs.

You can use the IF command to test for an error condition as follows:

```
> IF ERRORLEVEL != 2
> IF ERRORLEVEL == 6
```

Softerm also provides a special function which allows you to test for the existence or not of a filename on disk. The following examples show how the EXIST() function can be used with the IF command to test for filenames:

```
> IF EXIST("BRUCE.TXT")
    or
> FILENAME = "C:\SOFTERM\TEMP\SOFTERM.SS2"
> IF EXIST(FILENAME)
    or
> IF !EXIST(FILENAME)
```

IF commands can be nested. Each IF command must be matched to a corresponding ENDIF command for proper processing. If ELSE commands are used with nested IF commands, they will be matched with the first previous IF command which has not been terminated by an ENDIF

command.

The following is an example of nested IF's:

```
> IF condition (1)
>    commands processed if condition (1) true
>
>   IF condition (2)
>     commands processed if condition (2) true
>
>   ELSE
>     commands processed if condition (2) false
>
>   ENDIF terminates condition (2)
>
> ELSE
>   commands processed if condition (1) false
>
> ENDIF terminates condition (1)
```

Operators and variables can be combined to create expressions which can be evaluated and tested for True or False in an IF command. Following are some examples of the use of expressions in IF commands:

```
> IF STRINGVAR == 'SMITH'
>
> IF NUMVAR == ((3 * 5) + VALUE1)
>
> IF (GETLENGTH(STRING)) > 7

> IF (REPLY != 'Y'( && (REPLY != 'N')
> GOTO REPLYLOOP
> ENDIF
```

The script file parameters SV1-SV5 text replacement variables are substituted before processing of the IF command. Care should be taken in substituting a variable name or a string. For example:

```
> STRING = 'C:\SOFTERM\SOFTERM.EXE'
> SV1 = STRING
> IF EXIST(SV1)
> PROMPT (SV1' EXISTS IN THAT DIRECTORY')
> ENDIF
```

will substitute the value of STRING before processing the IF EXIST statement. Including quote marks around the SV1 would result in a check on the existence of the file named SV1. Similarly, you could use this method:

```
> SV1 = 'C:\SOFTERM\SOFTERM.EXE'
> IF EXIST(SV1)
> PROMPT (SV1' EXISTS IN THAT DIRECTORY')
> ENDIF
```

INPUT (arg1,arg2,arg3)

Function: Displays a dialog and allows text input to a script.

Syntax: > INPUT (title,prompt,input_var)

where:

TITLE is the text to be displayed in the Input window's title bar.

PROMPT is the text to be displayed. The text string may be up to 32 characters long.

INPUT_VAR is the variable to receive the input, which can be a maximum of 64 characters. If INPUT_VAR is initialized to a value before the INPUT command is processed, the value will be shown when the dialog is presented.

The INPUT dialog includes the following information:

- o The Script File name
- o The text to be displayed
- o A text input area
- o An OK button
- o A Cancel button

Selecting OK accepts the displayed input text and continues. Selecting Cancel or Close continues script processing, but ignores any changes to the input variable.

As with other functions, INPUT returns a code which is separate from the input variable. INPUT returns 0 for OK and a non-zero value for Cancel or Close.

An INPUT command might be included in a script file to prompt for a filename to be transmitted, or for a password in a logon sequence.

The following are valid INPUT commands:

```
> REPLYLOOP:
> SET REPLY = ''
> INPT = INPUT ('Kermit Send','Enter file name to send',REPLY)
> IF REPLY == ''
> GOTO REPLYLOOP
> ENDIF
> IF INPT != 0
> GOTO CANCEL
> ENDIF
> SET SV1 = REPLY
> SEND (KERMIT,SV1,SV1,,ON)
 or
> SET CHOICE = 'Y'
> INPUT (,'Enter more data?',CHOICE)
> IF (CHOICE == '') || (CHOICE == 'Y')
> GOTO MOREDATA
> ENDIF
```

JUMP label

Function: Transfer command processing to the command immediately following a specified Label.

Syntax: > JUMP Label

JUMP functions exactly as the GOTO directive.

LOG (arg1)

Function: Create a log file of all commands run by a script file.

Syntax: > LOG (d:\path\filename.ext)

If the specified log file already exists, it is deleted and a new file is opened. Wild card characters are not allowed in the log file name entered. The log file is created and written in standard text file format.

An initial entry is written to the log file when the command begins and a copy of the current command line with including arguments and switches is written to the log file. The final Characters, Blocks, and Errors counts displayed during SEND(), RECEIVE(), and HOST() commands also are written to the log file.

Once initiated by the LOG command, logging remains active until a NOLOG, END or CONVERSE command is performed, or the current script file is cancelled.

Once the log file is opened, any subsequent commands will be recorded. Each command recorded in the log file will include the current date and time in MM/DD/YY HH:MM:SS format. An I/O error while writing to the log file will cause it to be closed. The log file created may be printed or viewed.

Note: Multiple LOG files cannot be open at the same time. If a LOG command is performed while a log file already is open, the first log file will be closed and the new log file opened. If the command specifies the same file name as the previous log file, the previous log file is replaced.

LOWER (arg1)

Function: Returns a lower case text string.

Syntax: > LOWER (source_string)

SOURCE_STRING is the source text string (maximum length 255 characters).

Example:

```
> LOWCASE = LOWER ('THIS IS A TEST")
    or
> TXT = 'THIS IS A TEST'
> LOWCASE = LOWER (TXT)
```

The variable LOWCASE will contain the string 'this is a test'.

MESSAGE (arg1)

Function: Displays a message in the progress frame window.

Syntax: > MESSAGE (text)

where:

TEXT is the information text to be displayed. The string can contain a maximum of 255 characters. The text will remain until it scrolls off the progress frame window.

Note: If the script progress frame window is covered by another window, the text string will not be visible.

MKDIR (arg1)

Function: Create a directory path.

Syntax: > MKDIR (path_name)

where:

PATH_NAME is the directory path text string.

NEXTFILE (arg1)

Function: Returns an information text string for the next file matching the file name template specified in a FIRSTFILE function.

Syntax: > NEXTFILE (filedef)

where:

FILEDEF is the name of a variable that contains the unique identifier value that was returned by the previous FIRSTFILE function.

The returned information text string will be a null if there are no files matching the specified template; else the string will take the format defined for the FIRSTFILE function.

NOLOG ()

Function: Cancel the logging started by the LOG() function.

Syntax: > NOLOG()

ON (arg1,arg2,.....,arg11)

Function: Provides conditional branching to a label within a script file based upon a numeric value.

Syntax: > ON (numeric_val,label_1,label_2,...,label_10)

where:

NUMERIC_VAL is a numeric value in the range 1 to 10 typically obtained from a preceding XMIT_WAIT() function. If the value is less than 1 or greater than 10, or if no Label is defined for the value, processing will continue immediately with the first statement following the ON command.

LABEL_1 to LABEL_10 are the label names that correspond to the numeric values 1 to 10.

Important: The value for 'arg1' often will be provided by using a preceding XMIT_WAIT() function.

For example:

```
> ARG1_VAL = XMIT_WAIT (xmit_str,arg2,...,arg11)
> ON (ARG1_VAL,label_1,label_2,...,label_10)
```

ONERR label

Function: When a serious error -- one which would cause Script file processing to terminate -- occurs, transfer command processing to the specified Label.

Syntax: > ONERR Label

Note: Related directives and a reserved variable are: IF, ERRORLEVEL, RESUME and RETRY.

If an ONERR directive is used in a script file, when a serious error condition occurs processing will continue with the first command following a LABEL whose name matches the ONERR command label. If no LABEL is found with a matching name, command processing is cancelled.

If the ONERR directive is not used in a script file, processing of the file is cancelled when a serious error condition occurs.

Caution: When using the ONERR directive, it is possible to create looping conditions. For example, running a script file using Softrans protocol to RECEIVE a file which does not exist when the ONERR processing is set to RETRY the RECEIVE operation will cause looping.

Error conditions recognized by this directive include: expression evaluation errors; line timeout errors when using the character protocol; line failure errors which occur when the RETRIES count expires; operator cancel errors; remote cancel errors when using the Softrans and Kermit protocols; disk errors; and line disconnects due to loss of carrier.

Note: An operator cancel or a line timeout which terminates a character mode RECEIVE operation is not considered to be a serious error.

OPENNEW (arg1,arg2)

Function: Creates and opens a new file.

Syntax: > OPENNEW(filename.new,filedef)

where:

FILENAME.NEW is the file name text string.

FILEDEF must be the name of a variable to receive the file open handle that will be used as an argument in subsequent functions that access the same file.

Examples:

```
> RESULT = OPENNEW('PAYROLL.DOC',FILDEF)
    or
> FILENAME = 'PAYROLL.DOC'
> RESULT = OPENNEW(FILENAME,FILDEF)
```

On completion of the command, a result code is returned to the variable RESULT. A result code of 0 (zero) means the command was successful; a non-zero value indicates that the command was not successful.

OPENOLD (arg1,arg2)

Function: Opens an existing file.

Syntax: > OPENOLD(filename.old,filedef)

where:

FILENAME.OLD is the file name text string.

FILEDEF must be the name of a variable to receive the file open handle that will be used as an argument in subsequent functions that access the same file.

Examples:

```
> RESULT = OPENOLD('GRADES.CL3',FILDEF)
    or
> FILENAME = 'GRADES.CL3'
> RESULT = OPENOLD(FILENAME,FILDEF)
```

On completion of the command, a result code is returned to the variable RESULT. A result code of 0 (zero) means the command was successful; a non-zero value indicates that the command was not successful.

Note: On an OPENOLD command, the internal file pointer will be set at the start of the file.

PAUSE (arg1)

Function: Delay the start of the next script file command.

Syntax: > PAUSE (seconds)

The desired pause time in seconds is entered as the argument. A value from 1 to 255 seconds may be specified for the pause. The delay indicated is processed immediately and no further script commands are processed until the pause interval is complete. This function may be necessary when transferring files with some host computer systems to allow preparation time before the next command is processed.

PROMPT (arg1,arg2)

Function: Displays an information message dialog box until dismissed by the operator.

Syntax: > PROMPT (title,text)

where:

TITLE is the text which will be displayed in the prompt's title bar.
TEXT is the information text to be displayed. The string can contain a maximum of 255 characters.

Note: If the window in which the Script file is running is not the current window, the text string will not be visible and no action can be taken.

READ (arg1,arg2)

Function: Returns a text string from an open file.

Syntax: > READ (filedef,numeric_val)

where:

FILEDEF must be the name of a variable that contains the handle returned by a previous file open function.
NUMERIC_VAL is the numeric value of the length of data to be read.

Examples:

```
> TXT = READ (FILEDEF,20)
    or
> READCOUNT = 20
> TXT = READ (FILDEF,READCOUNT)
```

TXT is a variable to receive the data read, and READCOUNT is a numeric variable or constant indicating the amount of data to be read (255 characters maximum). The returned TXT string will be a null length string if the current file position is at the End-of-File. If less data is available than READCOUNT, the read still will be performed without error and the amount of data read may be determined using the GETLENGTH function.

READLINE (arg1)

Function: Returns a line of text from an open file.

Syntax: > READLINE (filedef)

where:

FILEDEF must be the name of a variable that contains the handle returned by a previous file open function.

Example:

```
> TXT = READLINE (FILEDEF)
```

TXT is a variable to receive the data read. A line of data is a maximum of 255 characters terminated by CR, LF, CR+LF or LF+CR. The terminator sequence is not returned with the read data. The returned TXT will have a length of 1 and will contain a control-Z (26 decimal) character if the internal file pointer is at End-of-File.

You can copy, profile and run the following script to demonstrate the READLINE() function:

```
> windowpos (0,0)
> windowtitle ('ReadLine Test - ' + sessionname)
> windowsize (24,80)
> windowprms (off,200,201)
> watch (off,on,3)
> windowshow ()
> delete ('\openold.000')
> copy ('\config.sys','\openold.000',replace)
> openold ('\openold.000',handle)
> read:
> text = readline (handle)
> if ((getlength (text) == 1) && (chrtoval (text) == 26))
>   message ('Press any key to terminate script')
>   timeout = 0
>   watch (on)
>   wait ()
>   close (handle)
>   delete ('\openold.000')
>   end
> endif
> message (text)
> goto read
```

RECEIVE (arg1,arg2,arg3,arg4,arg5,arg6,arg7)

Function: Transfer a file from a host to your PC.

Syntax:
```
> RECEIVE    (protocol,
              host_filename,
              local_filename,
              if_exist,xmit,
              display_stats,
              display_filename)
```

where:

PROTOCOL is a text string defining either a file transfer protocol profile or a file transfer protocol name. If a profile has the same name as a protocol, this argument will use the profile.

HOST_FILENAME is the file name as it appears on the host system.

LOCAL_FILENAME is the name to be assigned to the file on the local PC.

IF_EXIST is the option to use if a file with the LOCAL_FILENAME already exists. The options

are:

Replace	Overwrite the existing file with the new file.
Append	Add the contents of the new file to the end of the existing file. This generally is not a appropriate option to use with binary files.
Fail	Cancel the Receive operation. This is the default.

XMIT is the initial transmit text string.

DISPLAY_STATS specifies whether the file transfer statistics should or should not be displayed.

Note: The statistics display is described in the Session Window: File Menu chapter. The allowable values are:

ON	Display statistics
OFF	Do not display statistics

DISPLAY_FILENAME specifies whether the names of the files being transferred should or should not be displayed in the Script window. The allowable values are:

ON	Display names (the default)
OFF	Do not display names

Examples:

```
> RECEIVE ('character','letter.txt', 'letter.txt',append, 'type letter.txt[CR]',on,off)

    or

> SV1 = 'kermit'
> SV2 = 'project.dbf'
> SV3 = 'kermit -s '+SV2+'[CR]'
> RECEIVE (SV1,,SV2,replace,SV3,off,on)
```

RENAME (arg1,arg2)

Function: Renames a file.

Syntax: > RENAME (old_name, new_name)

where:

OLD_NAME is the current file name text string.
NEW_NAME is the new file name text string.

Example: > RENAME ('SETHRPT.TXT','SETHRPT.BK1')

RESUME

Function: Used in ONERR directive processing to resume processing with the next command after the command on which an error occurred.

Syntax: > RESUME

This directive requires no additional parameters. If it is processing as a result of ONERR processing, the next command after the command on which the error occurred will be processing. This directive is ignored when an error has not occurred.

RETRY

Function: Used with ONERR directive processing to retry the command on which an error occurred.

Syntax: > RETRY

If this command is processed as a result of ONERR processing after an error has occurred, the command on which the error occurred will be re-processed.

Example:

```
> ONERR Error
> DIAL ()
.
> Error:
> IF ERRORLEVEL == 1
> RETRY
> ELSE
> GOTO EXIT
> ENDIF
```

Important: Incorrect use of the RETRY directive can cause looping.

This directive is ignored when an error has not occurred.

RSTATTR (arg1,arg2)

Function: Resets a file attribute.

Syntax: > RSTATTR (fname,attrib)

where:

FNAME is the file name text string.

ATTRIB is a text list which can contain any combination of the A, H, and R attribute characters, as described in the FINDFIRST function. The volume, sub-directory, and system attributes may not be reset.

Examples:

```
> RSTATTR('C:\SOFTERM\SOFT.PWD','AH')
    or
> FILENAME = 'C:\SOFTERM\SOFT.PWD'
> RSTATTR(FILENAME,'H')
    or
> RESULT = RSTATTR(FILENAME,'AHR')
```

On completion of the last example, a result code is returned to the variable RESULT. A result code of 0 (zero) means the command was successful; a non-zero value indicates that the command was not successful.

SEND (arg1,arg2,arg3,arg4,arg5,arg6)

Function: Transfer a file from your PC to a host.

Syntax:
```
> SEND (protocol,
        local_filename,
        host_filename,
        xmit,display_stats,
        display_filename)
```

where:

PROTOCOL is a text string defining either a file transfer protocol profile or a file transfer protocol name. If a profile has the same name as a protocol, this argument will use the protocol.

LOCAL_FILENAME is the name as it appears on the local PC.

HOST_FILENAME is the name to be assigned to the file on the host system.
XMIT is the initial transmit text string.

DISPLAY_STATS specifies whether the file transfer statistics should or should not be displayed.
Note: The statistics display is described in the Session Window: File Menu chapter. The allowable values are:

ON	Display statistics
OFF	Do not display statistics

DISPLAY_FILENAME specifies whether the names of the files being transferred should or should not be displayed in the Script window. The allowable values are:

ON	Display names (the default)
OFF	Do not display names

Examples:

```
> SEND ('character', 'letter.txt', 'letter.txt', 'accept letter.txt[CR]', on, off)
    or
> SV1 = 'kermit'
> SV2 = 'project.dbf'
> SV3 = 'kermit -r '+SV2+'[CR]'
> SEND (SV1,, SV2, SV3, off, on)
```

SETATTR (arg1,arg2)

Function: Sets a file attribute.

Syntax:
```
> SETATTR (fname,attrib)
```

where:

FNAME is the file name text string.

ATTRIB is a text list which can contain any combination of the A, H, and R attribute characters, as described in the FINDFIRST function. Refer to the RSTATTR function for additional information.

SETBOF (arg1)

Function: Sets the current file position to beginning-of-file.

Syntax: > SETBOF (filedef)

where:

FILEDEF must be the name of a variable that contains the handle returned by a previous file open function.

SETEOF (arg1)

Function: Sets the current file position to end-of-file.

Syntax: > SETEOF (filedef)

where:

FILEDEF must be the name of a variable that contains the handle returned by a previous file open function.

SKIP (arg1,arg2)

Function: Sets the file position relative to the current file position.

Syntax: > SKIP (filedef, skipcount)

where:

FILEDEF must be the name of a variable that contains the handle returned by a previous file open function.

SKIPCOUNT is the numeric reposition value.

This function will adjust the current position of the pointer in the file referenced by the file definition variable FILDEF by the number of characters defined by the argument SKIPCOUNT. SKIPCOUNT is a numeric string indicating forward or backward position adjustment in the range -2,147,483,648 to +2,147,483,647 or a numeric variable defining the reposition count. Omission of the leading sign in the case of a numeric string implies a forward reposition.

```
> SKIP (FILDEF,-10)
    or
> SKIPCOUNT = -10
> RESULT = SKIP (FILDEF,SKIPCOUNT)
```

On completion of the command, a result code is returned to the variable RESULT. A result code of 0 (zero) means the command was successful; a non-zero value indicates that the command was not successful.

STRFIND1 (arg1,arg2,arg3)

Function: Returns the numeric position of the start of a specified text string within a text string. A value of zero is returned if the specified string is not found.

Syntax: > STRFIND1 (var_name,text,which)

where:

VAR_NAME is the name of the variable to be searched and may be defined as the read-only predefined system variable, RXDATA, which contains the last 255 characters received by the most recent XMIT_WAIT function.

TEXT is the text string to find (maximum length 255 characters).

WHICH is the occurrence to be found. A value of zero will locate the final occurrence.

Important: STRFIND1() is not case sensitive; lower case will match upper case when performing the search.

Examples:

```
> SRCH = 'FT'
> POS = STRFIND1(SRCH,'Softerm does the job!',1)
```

The numeric variable POS will contain 3, which is the value of the position of the first occurrence of the specified text string in the text string defined by the text variable SRCH.

```
> TXT = 'Four score and seven years ago,...'
> POS = STRFIND1 (RXDATA,TXT,0)
```

This is similar to the first example, except a text variable is used to hold the search string and the last 255 characters received by the most recent XMIT_WAIT function are searched for the last occurrence of the search text.

STRFIND2 (arg1,arg2,arg3)

Function: Returns the numeric position of any character from a specified text list within a text string. A value of zero is returned if no characters from the specified list are found in the specified string.

Syntax: > STRFIND2 (var_name,list,which)

where:

VAR_NAME is the name of the variable which contains the source string. This may be defined as the read-only predefined system variable, RXDATA, which contains the last 255 characters received by the most recent XMIT_WAIT function.

LIST is the text string that is the list of characters (maximum length 255 characters).

WHICH is the occurrence to be found. A value of zero will locate the final occurrence.

Important: STRFIND2() is not case sensitive; lower case will match upper case when performing the search.

Examples:

```
> POS = STRFIND2 (RXDATA,VALTOCHR(30),0)
```

The numerical variable POS will contain the position of the last occurrence of the Record Separator character in the RXDATA string. If POS is zero, no Record Separator character was located in the string.

```
> POS = STRFIND2 (TXT_VAR,'XYZ',1)
```

The numerical variable POS will contain the position of the first occurrence of 'x', 'X', 'y', 'Y', 'z' or 'Z' in the TXT_VAR string. If POS is zero, none of the specified characters was located in the string.

```
> POS = STRFIND2 (TXT_VAR,'\',0)
```

The numerical variable POS will contain the position of the last occurrence of the path separator '\' in the TXT_VAR string. If POS is zero, the character was not located in the string.

STRGET1 (arg1,arg,arg3)

Function: Extracts a text string from a text string by length.

Syntax: > STRGET1 (var_name, start_pos, length)

where:

VAR_NAME is the name of the variable which contains the source string. This may be defined as the read-only predefined system variable, RXDATA, which contains the last 255 characters received by the most recent XMIT_WAIT function.

START_POS is the numeric value of the extraction start position. This value generally is obtained from one of the STRFIND functions.

LENGTH is the numeric value of the length of the string to be extracted (maximum length 255 characters).

Example:

```
> POS = STRFIND1(SRCH,'soft',1)
> TXT = STRGET1 (SRCH,POS,21)
```

Using the results of the STRFIND1() example, the STRGET1() command would return the text variable TXT with 'Softerm does the job!'

STRGET2 (arg1,arg,arg3)

Function: Extracts a text string from a text string terminated by a specific terminator character from a text string.

Syntax: > STRGET2 (var_name,start_pos,termchar_list)

where:

VAR_NAME is the name of the variable which contains the source string. This may be defined as the read-only predefined system variable, RXDATA, which contains the last 255 characters received by the most recent XMIT_WAIT function.

START_POS is the numeric value of the extraction start position. This value generally is obtained from one of the STRFIND functions.

TERMCHAR_LIST is the termination character list.

Example:

```
> POS = STRFIND1(SRCH,'soft',1)
> TXT = STRGET2 (SRCH,POS,'!')
```

Using the results of the STRFIND1() example, the STRGET1() command would return the text variable TXT with 'Softerm does the job!'

STRPUT (arg1,arg2,arg3)

Function: Puts a text substring within a text string, replacing any existing characters.

Syntax: > STRPUT (source, text, start_pos)

where:

SOURCE is the source text string (maximum length 255 characters).

TEXT is the replacement text string (maximum length 255 characters).

START_POS is the numeric value of the start position. This value generally is obtained from one of the STRFIND functions.

STRTOVAL (arg1)

Function: Returns the numeric value representing an ASCII numeric text string (for example, '123' is returned as 123).

Syntax: > STRTOVAL ('numeric_string')

where:

'NUMERIC_STRING' is the ASCII numeric text string.

Example:

```
> NUM_VAL = STRTOVAL ('754')
```

The numeric variable NUM_VAL will contain the decimal number 754 (seven hundred fifty-four).

STRTOVAL() is used to convert ASCII text to a binary value. It typically is used to convert operator input to a numeric variable that can be used for arithmetic operations. For example, the following commands implement a simple command loop:

```
> ;operator input is placed in the
> ; text variable TEXTCOUNT
> COUNT = STRTOVAL(TEXTCOUNT)
> LOOP:
> IF !COUNT
> GOTO LOOPEND
> ELSE
> COUNT = COUNT - 1
> GOTO LOOP
> LOOPEND:
```

Before operator input can be used in an arithmetic operation, it must be converted to a numeric variable. STRTOVAL() performs this function.

TERMINAL (arg1)

Function: Resume or suspend the current rntime session's terminal emulation.

Symtax: > TERMINAL (option)

where:

OPTION is either START to resume the terminal emulation or STOP to suspend the terminal emulation.

When a script file is executed, any terminal emulation that is executing as part of the runtime session is suspended until script execution is complete. During execution of a script file, any terminal specific queries received from the remote must be processed by the script file and any responses required by the remote must be sent from within the script file. Upon completion of the script file, the terminal emulation is resumed and it is possible that some internal terminal emulation settings may not be set correctly since the script file will have processed all of the receive data. The TERMINAL() function allows the terminal emulation to be resumed from within the script file so thatit can process all receive data and generate any required responses. In addition, any transmit data in an XMIT_WAIT() function will be processed and transmitted by the terminal emulation as though it had come from the keyboard. Thus, if the terminal emulation keyboard is in a scancode mode, the ASCII transmit string will be converted automatically to the appropriate scan codes. The script file will also have access to the receive data in order that any XMIT_WAIT functions will still work. When script file execution is complete, the terminal emulation window (and the scrollback buffer) will be displaying the data that was received during the execution of the script file.

UPPER (arg1)

Function: Returns an upper case text string.

Syntax: > UPPER (source_string)

where:

SOURCE_STRING is the source text string (maximum length 255 characters).

Example:

```
> UPCASE = UPPER ('this is a test")
    or
> TXT = 'this is a test'
> UPCASE = UPPER (TXT)
```

The variable UPCASE will contain the string 'THIS IS A TEST'.

VALTOCHR (arg1)

Function: Returns the ASCII character representing a numeric value (for example, 49 is returned as '1')

Syntax: > VALTOCHR (numeric)

where:

NUMERIC is a numeric value in the range 0 through 255.

Example:

```
> ASC_CHAR = VALTOCHR (65)
```

The ASCII text variable ASC_CHAR will contain the ASCII character 'A'.

VALTOSTR (arg1)

Function: Returns the ASCII numeric text string representing a numeric value.

Syntax: > VALTOSTR (numeric_val)

where:

NUMERIC_VAL is the numeric value. Example:

```
> ASC_VAL = VALTOSTR (754)
```

The text variable ASC_VAL will contain the ASCII string '754'.

VALTOSTR() is the inverse of STRTOVAL(); it converts a numeric value to a string value. For example, to report the counting process, the above command sequence could be modified as follows:

```
> ;operator input is placed in the
> ; text variable TEXTCOUNT
```

```
> COUNT = STRTOVAL(TEXTCOUNT)

> LOOP:
> IF !COUNT
> GOTO LOOPEND
> ENDIF
> SV1 = VALTOSTR(COUNT)
> MESSAGE ('Current count is 'SV1)
> COUNT = COUNT - 1
> GOTO LOOP
```

If you set a variable equal to a numeric value, remember that Softerm stores numeric values in binary format, not text. Therefore, you would need to use the function VALTOSTR() (Value-To-String) to use the variable as an ASCII character.
The reserved variable, ERRORCODE, contains the numeric value of the last error code generated. For instance, if you had an error trap routine that displayed the error value, you would need to convert the value contained in ERRORCODE to an ASCII value:

```
> ERROR:
> SV1 = VALTOSTR(ERRORCODE)
> PROMPT (SV1)
```

VCLEAR()

Function: Deletes all variables and their associated resources except the ERRORCODE variable and the Script Variables.

Syntax: > VCLEAR()

Note: To remove only a single variable and its resources, use the VDELETE() function.

VDEFINED (arg1)

Function: Tests for the existance of a variable and returns a TRUE or FALSE value result.

Syntax: > VDEFINED (variable_name)

where:

VARIABLE_NAME is the name of the variable.

This function is used in an IF command to check if a variable has been defined. If the variable exists, True is returned, else False is returned. The result of this function may be inverted by using the not operator (!).

Examples:

```
> VAL = 6
> IF VDEFINED(VAL)
```

would return a True, while:

```
> IF !VDEFINED(VAR_NAME)
```

would return False if VAR_NAME had been defined previously.

VDELETE (arg1)

Function: Deletes the specified variable and any associated resources.

Syntax: > VDELETE (variable_name)

where:

VARIABLE_NAME is the variable to be deleted.

WAIT ()

Function: Wait for an event and return an event code numeric value

Syntax: > WAIT()

The returned event code will be:

0	to indicate a timeout.
-1	to indicate a carrier transition, either up to down or down to up.

Any other code will be the ASCII character of a keystroke.

Examples:

To delay until any event takes place, simply use:

```
> WAIT()
```

To test for a particular event, use:

```
> EVENT = WAIT()
> IF EVENT == 0
> GOTO LABEL_A
> ENDIF
> GOTO LABEL_B
```

WRITE (arg1,arg2)

Function: Writes a text string to an open file.

Syntax: > WRITE (filedef,text)

where:

FILEDEF must be the name of a variable that contains the handle returned by a previous file open function.

TEXT is the text to write.

A newline sequence (CR/LF) is not appended to the text written to the file. The position pointer in the file will move to the next character after the text is written.

Examples:

```
> WRITE (FILEDEF,'Testing...')
    or
> TXT = 'Total Orders Shipped This Month = '
> RESULT = WRITE (FILDEF,TXT)
```

On completion of the command, a result code is returned to the variable RESULT. A result code of 0 (zero) means the command was successful; a non-zero value indicates that the command was not successful.

WRITELINE (arg1,arg2)

Function: Writes a line of text to an open file.

Syntax: > WRITELINE (filedef,text)

where:

FILEDEF must be the name of a variable that contains the handle returned by a previous file open function.

TEXT is the text to write.

A newline sequence (CR/LF) will be written to the file after the text has been written to the file, and the position pointer in the file will move to the next character after the newline sequence.

Examples:

```
> WRITELINE (FILEDEF,'Testing...')
    or
> TXT = 'Total Orders Shipped This Month:'
> RESULT = WRITELINE (FILDEF,TXT)
```

If an error occurs, processing will continue at the label specified by the ONERR directive, or the script will terminate.

XMIT_WAIT (arg1,arg2,.....,arg12)

Function: Send a text string, wait for 1 of 10 optional specified replies, and return the numeric value of the response received.

Syntax: > XMIT_WAIT (text_string, echo_option, reply_1,..., reply_10)

where:

TEXT_STRING is the string to transmit. The string can contain a maximum of 255 characters.

ECHO_OPTION can be ON or OFF (the default). If set to ON, the caller's input in response to

the transmitted string will be echoed back. Otherwise, the input will not be visible to the caller.

REPLY_1 through REPLY_10 are 10 optional text string responses. These strings can contain a maximum of 32 characters.

A value of zero is returned if a timeout occurs without receiving any of the 10 strings.

Important: This function often will be used to provide the 'arg1' value to an ON() function.

Example:

```
> ARG1_VAL = XMIT_WAIT (xmit_str,,arg2,...,arg12)
> ON (ARG1_VAL,label_1,label_2,...,label_10)
```

If the initial transmit string is not defined, the contents of the receive buffer will not be flushed when the function is executed. This allows the XMIT_WAIT() funciton to be used to process data received prior to the execution of the function. When the initial transmit string is defined, the recive buffer is flushed in order that the XMIT_WAIT() function can process the dta that is received in response to the transmitted string.

Window Control Functions

WATCH (arg1,arg2,arg3)

Function: Enable or disable the display of script file commands and error messages in the script window.

Syntax: > WATCH (on_off,errors,errortimeout)

where:

ON_OFF specifies the default setting for displaying script commands as they are processed. If set to ON, the commands will be displayed. If arg1 of the WINDOWPRMS() function is set to ON, the script window will contain a menu which can be used to override this argument.

ERRORS defines whether or not error and information messages will be displayed. If set to ON, error messages will be displayed, even if the WINDOWSHOW() function is not used.

ERRORTIMEOUT is a numeric value which sets the amount of time, in seconds, that an error message will be displayed before it is cleared. Valid values are:

0	Do not clear the error message.
1 - 255	Valid number of seconds

Examples:

```
> WATCH (ON,ON,0) ; display commands as they
                  ;  are processed,
                  ; display error messages,
                  ;  and leave them displayed
                  ;  until cancelled
> WATCH (OFF,ON,100)    ; do not display commands,
                  ; do display error messages
                  ;  and leave them until
                  ;  cancelled by the
                  ;  operator or until 100
                  ;  seconds have elapsed.
```

WINDOWPOS (arg1,arg2)

Function: Set the upper left corner of a script file window.

Syntax: > WINDOWPOS (horizontal, vertical)

where:

HORIZONTAL is the horizontal displacement, in pixels, from the left edge of the screen.

VERTICAL is the vertical displacement, in pixels, from the top edge of the screen.

Note1: Screen positions are zero-relative to the top left corner. That is, the top left corner has the coordinates (0,0).

Note2: The screen position function is based on pixels, rather than text column and row. The base screen size depends on the video adapter and monitor being used. A rough conversion is 8 pixels of horizontal movement is approximately equal to one character column, and 14 pixels of vertical movement is approximately equal to one character row.

Note3: If this function is not used, the script window will use the same start coordinates as its parent runtime, terminal emulation window.

Example:

```
> WINDOWPOS (50,32)     ; the start point is
                        ; approximately at text
                        ; column 10 (50/5) and
                        ; row 2 (32/16)
```

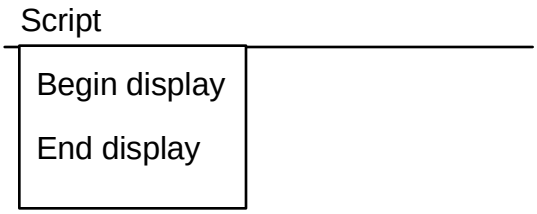
WINDOWPRMS (arg1,arg2,arg3)

Function: Set menu and display options to be used by a script window.

Syntax: > WINDOWPRMS (menu,scrname_pos,pathname_pos)

where:

MENU can be ON or OFF. If set to ON, the script window will have a menu bar and one menu, similar to:



If arg1 of the WATCH() function is set to OFF, the Begin display option will be active and the and display option will be greyed. Select Begin display to show the script commands, as they are processed, in the client area. Conversely, if arg1 of the WATCH() function is set to ON, the End display option will be active and the Begin display option will be greyed.

SCRNAME_POS is the numeric priority which determines if and where the name of the running script file will be displayed. Valid numeric values are:

0	Do not display the name of the running script file
1 - 127	Display the name above the client area
129 - 255	Display the name below the client area

PATHNAME_POS is the numeric priority which determines if and where the name of the current directory will be displayed. Valid numeric values are:

0 Do not display the name of the current directory
1 - 127 Display the name above the client area
129 - 255 Display the name below the client area

Note: If both names are to be displayed either above or below the client area, the name having the lower value will be displayed first.

Examples:

```
> WINDOWPRMS (OFF,0,0)  ; neither menu nor names
                        ; are displayed
> WINDOWPRMS (ON,1,129) ; the script menu is
                        ; displayed, the script
                        ; name is displayed
                        ; above the client area,
                        ; and the current
                        ; directory is displayed
                        ; below the client area.
```

WINDOWSHOW ()

Function: Cause the script window to be displayed.

Syntax: > WINDOWSHOW ()

If this function is not used, no script window will be displayed. If arg2 of the WATCH() function is set to ON, however, error and information messages still will be displayed.

WINDOWSIZE (arg1,arg2)

Function: Set the size of the client area of a script file window.

Syntax: > WINDOWPOS (rows, columns)

where:

ROWS is the number of screen rows which will be used by the script window's client area.

COLUMNS is the number of screen columns which will be used by the script window's client area.

Note1: If this function is not used, the script window will be the size of its parent runtime, terminal emulation window.

Note2: If this function is used after the script window has been displayed using the WINDOWSHOW function, it only may be used to make the window smaller.

Example:

```
> WINDOWSIZE (1,15)
```

WINDOWTITLE (arg1)

Function: Provide a title for a script file window.

Syntax: > WINDOWTITLE ('title')

where:

TITLE is the text string to place in the script window's title bar.

Note: If this function is not used, the title will default to:

Softerm script - session_name

Examples:

```
> WINDOWTITLE ('Time')  ; Time is the title
> WINDOWTITLE ('')      ; title bar is blank
```

CLOCK.SCR

This example shows how the script window control functions work together to display the current time. Create CLOCK.SCR using any editor which can save files in standard ASCII form, and make a Script profile so you can run this example from a Session Window. You might want to play with the different argument settings to see how they affect the operation and appearance.

```
> WINDOWPOS (0,0)           ;start the window in the upper left corner
> WINDOWTITLE ('Time')       ;the title
> WINDOWSIZE (1,14)          ;the window client area is 1 row by 14 columns
> WINDOWPRMS (OFF,0,0)   ;no menu and no file or directory name display
> WATCH (OFF,ON,10)          ;don't show commands as they're processed, error messages will
                            ; be displayed for 10 sec.
> WINDOWSHOW ()              ;enable the window
> TIMEOUT = 1                ;1-second delay for the WAIT() function
> LOOP:
> MESSAGE ('   ' + time + '   ') ;see Note
> IF (WAIT() != 0)            ;if a key is pressed
> END                         ;end operation
> ENDIF
> GOTO LOOP                   ;else, continue
```

Note: The Message consists of the catenation of 3 spaces, the system time (11 characters), and 3 more spaces. This centers the time in the 14 columns specified for the client area.

The Message function writes the text string to the last line of the client area. Because the specified area is only 1 line deep, it appears as though the time is continually updated. Actually, the messages are scrolled up off the display.