PROCOMM® PLUS

ASPECT Script Language Reference Manual

January 1991

DATASTORM's Technical Support Department will be happy to answer questions about the use or syntax of any ASPECT command or feature. We cannot, however, offer advice on how to write or debug your scripts. We have tried to include as many programming hints and examples as possible in the ASPECT SCRIPT LANGUAGE REFERENCE MANUAL, and we will continue to supply a wide variety of sample scripts both on CompuServe and on our in-house BBS.

Contents

CHAPTER

1

Introduction

Overview

This book was specifically designed as a companion to the PROCOMM PLUS USER MANUAL; it assumes that you already have experience in programming, and therefore does not attempt to teach the basics of writing a program. You'll find basic information on ASPECT and how to apply it in your USER MANUAL.

Instead, this reference provides the ASPECT programmer with detailed information on each script component and the **ASPCOMP** compiler. For your convenience, each chapter handles a separate topic:

The rest of this chapter arranges the ASPECT command words into functional "groups"—a group contains commands with similar functions, syntax or usage. Additionally, we'll briefly introduce the PROCOMM PLUS script compiler.

Chapter 2 is an alphabetized description of each command. We'll also cover the basic syntax and conventions of an ASPECT script.

Chapter 3 provides more details on using the compiler—in particular, how to upgrade scripts you've written for earlier versions of PROCOMM PLUS.

Chapter 4 presents the most commonly-asked ASPECT questions. We'll provide possible solutions and example scripts that you can include in your own programming.

Chapter 5 is devoted to the expert ASPECT user! This chapter presents two advanced script examples (demonstrating external Host programming and remote commands).

Finally, at the back of the manual you'll find complete technical information (including error messages, the **PCEDIT** text editor and a complete ASPECT index).

You'll notice that each chapter begins with an *Overview* (like this one), which will introduce the topics to be discussed and the purpose of that chapter. Use these "road signs" to identify both the chapters you need to read and those that you can skip.

ASPECT Command Words and Their Uses

The most commonly-used ASPECT commands are listed below, grouped according to function. You'll find complete details about *every* command in Chapter 2 of this manual.

Branching

CALL	Passes control to another procedure; control returns to the subsequent command.
EXECUTE	Passes control to another script file, but doesn't allow a return.
GOSUB	A synonym for **CALL**.
GOTO	Provides an unconditional branch or jump to a label within the current procedure.
LONGJMP	Returns control directly to a location previously "marked" with the **SETJMP** command.
SETJMP	Creates a location within a program that can be returned to immediately with the **LONGJMP** command.

Program Control

BYE	Terminates the executing script file and exits PROCOMM PLUS (leaving the connection open).
CWHEN	Deactivates an active **WHEN** command.
DEFINE	Substitutes a text string or allows conditional processing during compilation.
EXIT	Exits the script file and returns to Terminal mode.
FOR	Repeats a command or series of commands (up to the **ENDFOR** command) a specified number of times.
IF	Defines command execution (up to the **ENDIF** command) as dependent on a specified condition or event.

MSPAUSE	Halts execution for a specified number of milliseconds.
PAUSE	Halts execution for a specified number of seconds.
PROC	Marks the beginning of an ASPECT procedure block (which is terminated with an **ENDPROC** command).
QUIT	Terminates a script and exits from PROCOMM PLUS.
SWITCH	Executes commands on the results of comparisons between a variable and a series of **CASE** values.
SUSPEND UNTIL	Halts script execution until a given time.
TERMINAL	Terminates a script and returns the system to Terminal mode.
WAITFOR	Halts execution until a specified string is received or until a specified length of time elapses.
WAITQUIET	Halts execution for a specified time until the receive data line has been inactive for a certain number of seconds.
WHEN	Forces an automatic response or **CALL**s a procedure when a particular target string is encountered.
WHILE	Repeats a block of commands (terminated with an **ENDWHILE** command) until a condition is "FALSE".

String Manipulation

ASSIGN	Assigns a text string or another string variable to a string variable.
FIND	Tests for an occurrence of the specified text within a variable.
RSTRCMP	Compares two strings and sets the SUCCESS flag to "TRUE" if the strings are identical. Unlike **STRCMP**, the strings can contain non-displayable characters.
STRCAT	Concatenates one string variable or quoted string to another string variable.
STRCMP	Compares two strings and sets the SUCCESS flag to "TRUE" if the strings are identical.

STRCPY	Assigns a string variable or quoted string to another string variable. This command is the same as **ASSIGN**.
STRFMT	Creates a formatted string using a template and modifies it with string or numeric variables.
STRLEN	Returns the length of a string or string variable into an integer variable.
STRLWR	Converts the contents of a string variable to all lowercase characters.
STRPEEK	Reads a byte in a string and places it in a specified integer variable.
STRPOKE	Sets a byte in a string to a specified value.
STRSET	Initializes a string variable with a specified value (between 0 and 255) up to a designated length.
STRUPDT	Overwrites one or more bytes in a string with a new string value.
STRUPR	Converts the contents of a string variable to all uppercase characters.
SUBSTR	Extracts a new string from an existing quoted string or a string variable.

Input/Output

ATGET	Gets a string or numeric value using specified attributes at a specified row and column.
ATSAY	Displays a string or number in the specified colors at a location on the screen.
COMGETC	Assigns a numeric variable with the next character in the receive data buffer. Returns a "-1" if the buffer is empty.
COMGETCD	Similar to **COMGETC**, this command will wait up to two seconds for a character before returning a value.
COMPUTC	Sends the specified character out the active communications port.
EOF	Tests for end-of-file condition.
FATSAY	Displays a formatted string at the specified location and with the specified color attributes.

FCLEAR	Clears all end-of-file and error flags for the input file.
FCLOSE	Closes a file.
FFLUSH	Writes the current I/O buffer contents to the output file.
FGETC	Reads a character from input file into an integer variable.
FGETS	Reads a string from input file into a variable.
FOPEN	Opens a file in read, write or append mode and assigns it to a file index.
FPUTC	Writes the character represented by an integer to the output file.
FPUTS	Writes the contents of a string variable to the output file.
FREAD	Reads a block of data from the input file into a string variable.
FSEEK	Repositions the file pointer.
FSTRFMT	Similar to **STRFMT** and **FATSAY**; writes a formatted string to the output file.
FTELL	Returns the current file position into a long variable.
FWRITE	Writes a block of data from a string to the output file.
GET	Stores the user input in a string variable.
INPORT	Reads data from a specified I/O port into an integer variable.
KEYGET	Reads a keystroke and optionally places the value into a numeric integer variable.
KFLUSH	Clears the keyboard buffer.
MATGET	Positions the cursor at a row and column and gets a string or numeric value using the specified attributes; the input is masked with asterisks on the screen.
MESSAGE	Displays a specified string on the local screen at the current cursor position.
MGET	Stores user input in a string or numeric variable. The display is masked with asterisks to provide security.
OUTPORT	Writes integer data to a specified I/O port.

PUSHBACK	"Pushes" the last character read from the receive data buffer back to the head of the buffer.
REWIND	Repositions the file pointer back to the beginning of the file and clears the end-of-file and error flags for the file.
RDFLUSH	Clears the Redisplay buffer.
RFLUSH	Clears the receive data buffer.
RGET	Receives a text string sent by a remote user and stores it in a string variable.
RDWRITE	Writes the Redisplay buffer to disk.
TERMWRT	Displays the character represented by a value at the current cursor position and then moves the cursor to the next available location. Unlike **WRITEC**, no conversion is performed on the value.
TRANSMIT	Sends the specified string to a remote computer.
WRITEC	Performs emulation conversion and then displays or acts on the provided character value. The cursor is updated when appropriate.

Terminal Key Equivalents

BREAK	Sends a break of specified length to a remote computer.
CONNECT	Exits a script and returns to Terminal mode.
DIAL	Dials a dialing directory entry.
DLOAD	Loads a different dialing directory.
EMULATE	Sets the terminal type of the local system.
FETCH	Returns the current value of any **SET** command parameter.
GETFILE	Receives a file from a remote computer using the specified protocol.
HANGUP	Disconnects the line, ending a call.
HELP	Calls the Online Help Facility.
HOST	Enters Host Mode, where PROCOMM PLUS acts as a BBS.
KERMSERVE	Executes the specified KERMIT server command.
KLOAD	Loads or creates a Keymap file.

LOG	Enables logging of all data sent and received in Terminal mode to a disk file.
METAKEY	Executes the specified keyboard Meta key.
MDIAL	Dials a specified telephone number (unlike **DIAL**, which calls a Dialing Directory entry).
MLOAD	Loads or creates a keyboard Meta key file.
PARMREST	Restores the PROCOMM PLUS parameter settings from the PCPLUS.PRM file.
PARMSAVE	Saves the current parameter settings to the PCPLUS.PRM file (overwriting the previous settings).
PRINTER	Turns the printer on or off.
REDIAL	Redials numbers in the dialing queue.
SET	Sets the specified parameter to a specific value.
SENDFILE	Sends a file to a remote computer using the specified protocol.
SNAPSHOT	Copies the contents of the current screen to the disk file PCPLUS.SCR.
TERMKEY	Accepts a value representing a key and performs a program function.

Display and Sound

ALARM	Sounds an alarm for a specified number of seconds.
BOX	Displays a bordered box of the specified size and color.
CLEAR	Clears the screen.
CUROFF	Turns the cursor off.
CURON	Turns the cursor on.
DSCROLL	Scrolls a specified area of the screen down by the specified number of lines.
GETCUR	Returns the cursor's current row and column into numeric variables.
GETVATTR	Returns the current display attribute from the specified screen coordinates into a numeric variable.
GETVCHAR	Returns the current character from the specified screen coordinates into a numeric variable.

LOCATE	Places the cursor at a specific location.
PUTVATTR	Selects the current display attribute for the specified screen coordinates.
PUTVCHAR	Places a character at the specified screen coordinates.
SCROLL	Scrolls a specified area of the screen up by the specified number of lines.
SOUND	Makes a sound of a given frequency (in Hertz) and duration (in hundredths of a second).
TYPE	Displays the specified ASCII text file.
VIDREST	Restores the current video buffer data.
VIDSAVE	Saves the current video buffer data.

Directory and File Control

CHDIR	Changes the current DOS directory.
DELETE	Erases a file.
DIR	Displays a list of files.
DISKFREE	Returns the free space available on the specified drive into a long variable.
FINDFIRST	Locates a file (or files) with a file specification you provide.
FINDNEXT	Locates the next occurrence of a file which matches a previous **FINDFIRST** command.
GETDIR	Returns the current working directory path of the specified drive into a string variable.
GETFATTR	Returns the attributes of a specified file into a string variable.
GETFDATE	Returns the date stamp of a specified file into a string variable.
GETFSIZE	Returns the size (in bytes) of a specified file into a long variable.
GETFTIME	Returns the time stamp of a specified file into a string variable.
ISFILE	Determines if a file is present in the current DOS directory.
MKDIR	Creates a new directory using a specified path (or in the current directory if a path is not provided).

RENAME	Renames an existing file using specified paths (or the current directory). Files can be "moved" by renaming them to a different directory on the same drive.
RMDIR	Removes an existing directory using a specified path (or in the current directory if a path is not provided).
SETFATTR	Sets the attributes of a specified file.
SETFDATE	Sets the date stamp of a specified file.
SETFTIME	Sets the time stamp of a specified file.

Conversion Commands

ATOF	Converts an ASCII numeric string to a float value.
ATOI	Converts an ASCII numeric string to an integer value.
ATOL	Converts an ASCII numeric string to an long value.
FTOA	Converts a float to an ASCII string and stores it in a string variable.
ITOA	Converts an integer to an ASCII string and stores it in a string variable.
KEY2ASCII	Converts an integer to its ASCII character value and places the result into a string variable.
LTOA	Converts a long to an ASCII string and stores it in a string variable.

Arithmetic Commands

ADD	Adds two numbers or numeric variables and places the result in a numeric variable.
AND	Performs a bitwise comparison of two numbers and places the result in the specified numeric variable.
ANDL	Performs a logical AND comparison of two numbers and places the result in the specified numeric variable.
CEIL	Computes the smallest integer value *greater than* or *equal to* a floating point number you provide.

COMP	Performs a bitwise complement of a number and places the result in the specified numeric variable.
DEC	Decrements a numeric variable by one.
DIV	Divides one number by another and places the result in a numeric variable.
EQ	Performs equality test on two numbers.
FLOOR	Computes the largest integer value *less than* or *equal to* a floating point number you provide.
GE	Performs a greater-than-or-equal-to comparison.
GT	Performs a greater-than comparison.
INC	Increments a numeric variable by one.
INIT	Sets the value of a specified numeric variable.
LE	Performs a less-than-or-equal-to comparison.
LT	Performs a less-than comparison.
MOD	Returns the remainder left after the division of two numbers into a numeric variable.
MUL	Multiplies two numbers or numeric variables and store the result in a numeric variable.
NEG	Negates the value of the specified number and places the result in a numeric variable.
NEQ	Performs inequality test on two numbers.
NOT	Performs a logical NOT operation on a number.
OR	Performs a bitwise comparison of two numbers.
ORL	Performs a logical OR comparison on two numbers.
SHL	Performs a left shift operation on the bits of one number and places the result in the specified numeric variable.
SHR	Performs a right shift operation on the bits of one number and places the result in the specified numeric variable.
SUB	Subtracts one number or numeric variable from another and store the result in a numeric variable.
XOR	Performs a bitwise comparison of two numbers.

ZERO	Compares a number with zero.

Memory Manipulation

MEMFREE	Returns the free RAM available for running other programs into a long variable.
MEMPEEK	Returns the value of a single byte at the specified segment address and offset.
MEMPOKE	Sets the value of a single byte at the specified segment address and offset.
MEMREAD	Places the values of several bytes at the specified memory segment address and offset into the specified string variable.
MEMWRITE	Writes the contents of a string at the specified memory segment address and offset.

System, Date and Time Commands

DATE	Loads current system date into a string variable.
DOS	Executes the specified DOS command or batch file.
DOSVER	Returns the current DOS version (with major and minor version numbers) into a string variable.
GETENV	Reads the contents of an environment variable definition into a string variable.
HOOK	Executes an external program, passing the segment and offset of the current PROCOMM PLUS Setup structure as an argument.
PUTENV	Adds or modifies an environment variable.
RUN	Executes the specified program or command.
SHELL	Temporarily exits to DOS (leaving PROCOMM PLUS loaded in memory).
TIME	Loads the current system time into a string variable.

Terminal Emulation Commands

BLANKON	Sets the foreground equal to the background attribute.
BLINKON	Turns on the blinking attribute.
BOLDON	Turns on the bold attribute.
CURDN	Moves the cursor down one line.

CURLF	Moves the cursor one column to the left.
CURRT	Moves the cursor one column to the right.
CURUP	Moves the cursor up one line.
DELCHAR	Deletes the character at the current cursor position.
DELLINE	Deletes the entire line at the current cursor position.
DIMON	Turns on the dim attribute.
EBOL	Erases text from the current cursor position to the beginning of the line.
EBOS	Erases text from the current cursor position to the beginning of the screen.
EEOL	Erases the line under the current cursor position.
EEOS	Erases text from the current cursor position to the end of the screen.
HOME	Moves the cursor to the first row and column on the screen.
INSCHAR	Inserts a character at the current cursor position.
INSLINE	Inserts a new line at the current cursor position.
LINEFEED	Moves the cursor down one line.
NORMON	Turns on the normal attribute.
RCA	Interprets the next two received characters as row and column positions for the cursor.
REVON	Turns on the reverse attribute.
TERMRESET	Resets Terminal mode.
ULINEON	Turns on the underline attribute.

Miscellaneous Commands

COMMENT	Designates the beginning of a comment block. Any characters from this command to the ENDCOMMENT command following it will not be included in the compiled code.
ENDCOMMENT	Terminates a COMMENT block.
INCLUDE	Merges commands from other source files during compilation.

Using ASPCOMP

Like many other programming languages, ASPECT allows you to "compile" your programs.

During this process, the commands, data and text contained in an ASPECT Source Program (or *.ASP* file) are reduced from their full-length English format into code that PROCOMM PLUS can read directly. Strings are also encrypted for security.

After compiling, the resulting ASPECT Script Executable (or *.ASX* file) is smaller in size and takes less time to run. Additionally, since the string data is encrypted, it's much more secure—no one can read sensitive passwords or information!

You can compile a script in two ways:

- First, you can compile scripts ahead-of-time, before running PROCOMM PLUS. This method is faster, and it requires less memory. However, if you make frequent changes to a script, you'll have to recompile each time you modify it.
- Alternately, PROCOMM PLUS can automatically load the compiler and attempt to compile the script when you run it. This method is more convenient for those users who modify their scripts often, since the intermediate compile step is automatic; however, if PROCOMM PLUS doesn't have enough free RAM memory to load **ASPCOMP**, the compile process will abort.

Whenever PROCOMM PLUS prompts you for a script name, you can "force" it to recompile a source program by typing in the entire script name (including the *.ASP* extension). If the script name has no extension, PROCOMM PLUS will attempt to execute a compiled script with that name first; if no such script exists, PROCOMM PLUS assumes that the file is a source program (which must be compiled before execution).

You can run **ASPCOMP** from the DOS prompt:

- Type *ASPCOMP [options] scriptname* and press Enter.

The *scriptname* can include a path.

The options are:

/Dx[=text]	Defines a macro *x*. This symbol can also be initialized to a text string (similar to a macro in the "C" programming language). For more information on ASPECT macros and their uses, refer to the **DEFINE** command in the next chapter.

/M[L]	Generates a symbol map for this script (including any compiler warning messages). The *L* switch creates an additional source line number reference table.
/En	Determines the maximum errors to allow before aborting the compilation; if more than *n* errors are found by ASPCOMP, it will abort automatically. The default is 20 errors.
/Wn	Specifies the warning level for this compilation; warnings may indicate a potential problem in the source script. Unlike errors, however, warnings aren't considered "fatal"; the compiling process will not abort because of a warning. Valid values are 0, 1 and 2. At level 0, ASPCOMP suppresses all warning messages. At level 1 (the default), unreferenced global, local and parameter variables are reported. Level 2 reports all level 1 errors and additionally reports the names of "unreferenced" procedures as they're removed by the compiler (unreferenced procedures aren't **CALLed** anywhere within the source script, so they're effectively useless).

Spaces are required between options.

Running Compiled Scripts

To illustrate the use of ASPCOMP, let's create a simple ASPECT source program in the PCPLUS directory called *"TINY.ASP"*—its sole purpose is to display the sentence *"Hello, I'm a compiled ASPECT script!"* in the center of your screen. Our TINY.ASP looks like this:

```
PROC MAIN                ;initialize the MAIN procedure
  CLEAR                  ;clear the screen
  FATSAY 12 20 31 "Hello, I'm a compiled ASPECT script!"
ENDPROC                  ;end the MAIN procedure
```

Now we'll compile and run TINY.ASP from within PROCOMM PLUS. From Terminal mode, the process would be:

- Press Alt-F5 to run a script.
- PROCOMM PLUS prompts you for the name of the script—enter *TINY* and press Enter.

Since there is no file named "TINY.ASX", PROCOMM PLUS assumes that the script is in source program form (and that it must be compiled before use). You'll see a message alerting you that the script is being compiled, and then the new TINY.ASX will run.

Once you exit from PROCOMM PLUS, you'll note that TINY.ASX was saved as a separate file.

To compile TINY.ASP outside of PROCOMM PLUS, the process (from the DOS prompt) would be:

- Type *ASPCOMP TINY* and press Enter.

Whichever method you use to compile TINY.ASX, you'll notice that it's smaller than TINY.ASP by several bytes (of course, the larger the source program, the more bytes you'll save by compiling it). If you view TINY.ASX within PROCOMM PLUS, you'll also see that it's now tokenized (making it impossible for someone else to "decode" any of the data strings within TINY.ASX).

Key Processing in ASPECT

PROCOMM PLUS allows you to enter keystrokes while a script is executing. By default, the program checks for an available key before each command is executed; if Esc was pressed, PROCOMM PLUS asks you whether or not to terminate the current script. Any other key will be processed exactly as if it were typed in Terminal mode; for example, if you press PgUp, the Upload Protocol window is displayed. If the letter A is pressed, it's sent out the active COM port.

Certain ASPECT commands can be terminated by Esc; they require a period of time to pass before the command would normally complete (for example, **PAUSE, RGET, SUSPEND UNTIL, WAITFOR** and **WAITQUIET**). The GET-style commands (**ATGET, GET, MATGET** and **MGET**) can also be aborted by Esc. When any of these commands are terminated with Esc, the **FAILURE** flag is set and PROCOMM PLUS asks you whether or not to terminate the current script.

The **SET KEYS** command can be set ON to change the way keys are processed in a script. Esc will still terminate some commands (as mentioned above); however, the termination prompt is suppressed, and it's the script's responsibility to process both the abnormal termination of commands and keystrokes entered by the user. You can use commands like **KEYGET** and **TERMKEY** to monitor this processing.

The next chapter will describe each ASPECT command in-depth, along with the basic syntax and conventions used by ASPECT programmers.

CHAPTER

2

ASPECT Script Reference

Overview

This chapter provides alphabetized descriptions of all ASPECT command words (along with the parameters required by each word), mathematical operators, expressions and system variables. We'll cover the basic conventions used throughout this manual, as well as the rules that govern all ASPECT scripts.

The Elements of ASPECT

The basic "building blocks" of ASPECT are commands, parameters and expressions. Each line in a source program (or *.ASP* file) begins with:

- a *command word* (usually followed by one or more command *parameters*, which control the function of the command in some manner).

OR

- an *expression* consisting of one or more *operands* acted upon by one or more *operators*; for example, the expression

  N0=N1+N2

  contains 3 operands (N0, N1 and N2) and 2 operators (the "=" and "+" signs). Operands usually occur as a numeric constant or variable, but they can also be the result of a sub-expression within the expression (like the "=" sign in our example above, which uses the result of the sub-expression **N1+N2**).

OR

- a comment, denoted with a semicolon (;) or with the **COMMENT** command. Any characters to the right of a semicolon (or in a **COMMENT** block) are removed from the compiled script, allowing you to heavily comment your source programs without losing speed or efficiency.

OR

- a *label* (in the form LABEL:) denoting a target location in a procedure which you can "jump" to with the **GOTO** command word.

Commands, parameters and expressions conform to rules called "syntax"; like the syntax of written languages, command syntax ensures that the function of each ASPECT statement is clear and precise.

Naming Elements in ASPECT

All of the "named" elements in ASPECT—for example, command word names, procedure names, labels, variable names and **DEFINE** macros—follow the same basic syntax rules. Names have a maximum length of 80 characters.

Only the first 10 characters are treated as unique (therefore, two separate variables with names identical for the first 10 characters would be considered the same, and would generate a compiler error).

Names begin with any alphabetic character (A-Z or a-z) or an underscore (_). Additional characters can be alphabetic, the underscore character or numeric (0-9). Examples of valid names are "NUM", "_FINAL_", "S0", "e" and "Term5".

All pre-existing names (like command words and system variables) are considered "reserved" within ASPECT, and can't be used as unique names; a full listing of these reserved names appears at the end of this manual in Appendix C.

Command Words and Parameters

- Each command word begins on a separate line, and must be completed (with all added parameters) on that line—however, exceptionally long source lines can be "extended" to more than one line if a backslash (\) appears as the last character. No comments may follow the backslash. For example, both of these source lines are valid:

```
MESSAGE "ENTER YOUR NAME, PLEASE: "
WAITFOR "Enter your long message, please: " \
FOREVER
```

  Note that if a quoted string constant is extended to two lines, any spaces occurring before the text on the second line will be included in the constant.

- The maximum length of a command line in a source file is 255 characters, including spaces and line termination characters (like a carriage return or line feed).

- Command words and parameters cannot be abbreviated—they must be spelled completely.

- Command words and parameters must be separated from each other by a space.

- Commands and parameters are **not** case-sensitive (except for string information between quotes, as in *"Hello"*, and format specifiers used in the **FATSAY**, **FSTRFMT** and **STRFMT** commands, as in %d).

- Parameters always follow a command word on the same line (except when the backslash extender is used—see the paragraph at the beginning of this section).
- You **must** enter the parameters listed with a command (unless a parameter is enclosed in brackets or otherwise identified as optional).

Operators

ASPECT supports numeric expressions composed of operators and operands. The more commonly-used operators include !, ~, ++, --, *, /, %, +, -, << and >>. Operands include any numeric variables or constants.

For complete technical information on all of the supported operators, please see Appendix D at the back of this manual.

Predefined Variables

For convenience, ASPECT provides both numeric and string "predefined" variables (which need not be declared at the beginning of a script). Predefined variables are not reset when chaining script files, so they can be used to pass values; their values are initialized to 0 or null when a script is initially executed. Note that ASPECT also accepts *user-defined* variables—predefined variable names are not required.

There are 10 predefined *string* variables, labelled S0 through S9; each can contain a maximum of 80 characters.

The predefined *numeric* integer variables are labeled N0 through N9; they're typically used for math calculations.

System Variables

System variables are "read-only" variables to which ASPECT assigns specific values; they're available throughout an ASPECT script. For example, although you cannot change the value of the variable **$ROW** (the current row position of the cursor), you can read it anywhere within a script and use it in any statement where a *row* value is accepted.

Descriptions of all ASPECT system variables appear at the end of this chapter.

User-defined Variables

ASPECT uses named elements called *variables* to store and manipulate values within a script. Variables are passed as parameters to commands, and can be used within expressions. User-defined variable names must adhere to the rules for valid element names, and it's recommended that you use easily-recognized names that identify their use within a source program.

There are four types of variables: INTEGER, STRING, FLOAT and LONG. Complete information on the valid ranges and function of each variable type is provided in the section titled "ASPECT Conventions" later in this chapter.

User-defined Constants

User-defined string constants and variables can contain a sequence of up to 80 characters. Any ASCII character value may occur in a string variable (some characters are not displayable). To use a quotation mark within a text string, precede it with the special escape character ' (ASCII 96)—for example, MESSAGE *"She said '"Hello'""*. The NULL character (ASCII 0) normally indicates the end of a text string (and is not considered a part of the 80-character limit). For security, **ASPCOMP** encrypts all string constants during compilation.

User-defined integer and long constants can be specified in decimal or hexadecimal, while floating point constants must be entered in decimal format only. No spaces are allowed between the characters in a numeric constant.

For further information on each type of constant, please refer to ASPECT Conventions below.

Global and Local Variables

Up to 128 *global variables* of each type can be declared (a maximum of 512 total). Global variables are declared anywhere within a source program **outside** of any procedure (typically, this is at the top of a source program, before any procedures appear).

A global variable can be referenced within any procedure; if a procedure changes a global variable, its previous value is not saved.

Globals cannot be passed to another script by chaining—only the predefined N*x* and S*x* variables can pass information to another script.

Local variables are declared with the same commands as globals—however, they can only be referenced within the procedure where they're declared (unlike globals, which are declared outside of any procedure), and only 128 total local variables (of any type) can be used in a procedure.

The same local variable name can be used in more than one procedure, but each occurrence is a completely different case and has no effect on any other occurences.

Global and local variables must be declared before they are referenced; several variable definitions may occur on the same line if each definition is separated with a comma. Variables may also be initialized when they're declared, as in:

```
integer number1, number2=13
```

Escape Sequences

ASPECT uses *escape sequences* in string and character constants to represent nondisplayable characters and control codes (which have special meanings to hardware devices). They consist of a backwards single quote character (ASCII 96, the un-shifted tilde key, sometimes called the "back tic") followed by an alphabetic character or series of digits. The display screen will not be affected by control codes when using ASPECT commands which write directly to video memory (such as **ATSAY** and **FATSAY**).

The following escape sequences are supported:

Sequence	ASCII	Description
'a	7	Bell or alert
'b	8	Backspace
'f	12	Form feed
'n	10	New line/line feed
'r	13	Carriage return
't	9	Horizontal tab
'v	11	Vertical tab
''	39	Single quote
'"	34	Double quote
'`	96	Backwards single quote
'ddd		ASCII character, octal notation
'xddd		ASCII character, hex notation

If a backwards single quote precedes a character not listed above, the backwards single quote will be ignored.

When a "ddd" sequence is used, it may consist of one to three digits (ranging from 0 to 255). Values greater than 255 will generate an error. It's usually best to use all three digits (as in '004), since a letter or digit immediately following an escape sequence may represent either an octal or hex character (which would then be interpreted as part of the escape sequence).

Character Constants

A character constant is an integer value representing an ASCII character within the range 0 through 255. It consists of a single quote (ASCII 39) followed by an ASCII character and a terminating single quote. An escape sequence may also be used to represent an ASCII character within a character constant. For example:.

```
integer letter_a = 'a'          (integer constant 97)
integer escape = ''033'         (integer constant 27)
```

Labels and Procedures

Labels use a special name format. They must end in a colon and begin on a separate line—for example:

```
LABEL1:
```

Labels are used only with the **GOTO** command word (each **GOTO** specifies a label as a destination).

Procedure names are user-defined elements provided as parameters for the **PROC** and **CALL** command words—for example:

```
proc update
```

defines the beginning of a procedure block named UPDATE; you could later branch to that procedure with the statement:

```
call update
```

ASPECT Conventions

The following conventions are used to describe ASPECT commands and their parameters:

Lowercase italic	Substitute a variable or constant for a parameter shown in *lowercase italic*.
[ ]	Any parameter enclosed in square brackets ([ ]) is **optional**.
|	If multiple parameters are separated from each other by a vertical bar (|), choose only a **single** parameter; the vertical bar indicates "or".
" "	Quotation marks should be entered wherever shown.
...	You may insert any number of parameters in the location marked by an ellipsis (...).
. . .	Three vertical dots indicate any number of command statements in a *command block*. Examples of commands that use command blocks are the **PROC** and **WHILE** statements.
attribute	An integer value which specifies the background and foreground colors to be used with a command. Legal values range from 0 to 255. To compute the attribute value, select a foreground and a background color from the following list—note that the bright intensity colors can only be used as foreground (text) colors:

Regular Intensity		Bright Intensity	
0	Black	8	Dk Grey
1	Blue	9	Lt Blue
2	Green	10	Lt Green
3	Cyan	11	Lt Cyan
4	Red	12	Lt Red
5	Magenta	13	Lt Magenta
6	Brown	14	Yellow
7	Lt Grey	15	White

Multiply the background color by 16 and add the result to the foreground number. The final result is the attribute value. For example, the attribute for a blue background and a high intensity white foreground is 31: (1*16)+15=31.

character An integer representing an ASCII character value between 0 and 127 (or 255, if 8-bit values are included). Control characters range between 0 and 31, while displayable characters are usually considered to range from 32 to 126.

column A vertical column on the screen, specified either as an integer constant or variable. The maximum value of column is equal to the number of columns on the display screen minus 1; column 0 is always the leftmost column of the screen.

disk An integer specifying a disk drive within the current command. A value of 0 represents the current or default drive, 1 represents drive A, 2 represents drive B and so on.

float	Any floating point constant or variable. Float numbers range from approximately +/- 2.225e-308 to 1.797e+308. Float constants must be in decimal format (zero or more decimal digits from 0 to 9 followed by a period and zero or more decimal digits for the fractional portion, as in "1.32"). At least 1 digit must appear before or after the decimal point. Any numeric constant containing a period is assumed to be a floating point number. Several mathematical commands and their associated operators are not compatible with floating constants and variables—these include the complement, shift and bitwise commands and operators **COMP, SHL, SHR, AND, OR, XOR**, "~", "<<", ">>", "&", "|" and "^". No spaces are allowed between the characters in a float constant.
filespec	A string specifying the path and/or filename to be used in a command.
floatvar	Any floating point variable.
index	A zero-based integer indicating a single element (where several are available). For **WHEN, CWHEN, SETJMP, LONGJMP, VIDSAVE** and **VIDREST**, the index values range from 0 to 2. For file I/O operations like **FOPEN** and **FCLOSE**, index values range from 0 to 5.
integer	Any integer constant or variable. Integers are signed, and can range between -32768 and 32767. Integer constants may be specified in decimal or hexadecimal format. Any constant appearing in decimal format (within the range of integer values) is considered an integer type, as is any number composed of 4 or less hexadecimal digits (if the number is longer than 4 hex digits, it's assumed to be a long constant). No spaces are allowed between the characters in an integer constant.
intvar	Any integer variable.
length	An integer specifying a string length, numeric input length or floating point precision; possible values range from 0 to 80.

long	Any long constant or variable. Long numbers are signed, with a range of -2,147,483,648 through 2,147,483,647. Long constants may be specified in decimal or hexadecimal format. A constant without a period may be forced as a long constant by appending the letter "L" to the end (the character can be upper or lower case). No spaces are allowed between the characters in a long constant. Truncation will occur if necessary.
longvar	Any long variable.
name	An unquoted character string containing the name of a procedure, macro, label or variable. For more information, see "Naming Elements in ASPECT" in this chapter.
number	Any integer, long or floating point constant or variable. No spaces are allowed between the characters in a numeric constant.
numvar	Any integer, long or floating point variable.
Nx	A "predefined" numeric integer variable, where *x* ranges from 0 to 9.
offset	An integer representing a memory address or offset value within a memory segment. The value is treated as unsigned within a command statement, with values ranging from 0 to 65535. Values greater than 32767 must be entered as hexadecimal.
port	An integer representing an I/O port address value—**this value should not be confused with the COM port Dialing Directory field or the Line Settings window.** The value is treated as unsigned within a command statement, ranging from 0 to 65535. Port values greater than 32767 must be entered as hexadecimal.
protocol	A valid protocol name; used by the **SENDFILE** and **GETFILE** commands.
row	A horizontal row on the screen, specified as an integer constant or variable. The maximum value of row is equal to the number of rows on the display screen minus 1; row 0 is always the top row of the screen.

segment	An unsigned integer value corresponding to a memory segment address. The value is treated as unsigned within a command statement, ranging from 0 to 65535. Values greater than 32767 must be entered as hexadecimal.
string	Any quoted string constant or variable. A quoted string constant can be up to 80 characters in length, and is surrounded by quotation marks (for example, ***"Hello"***).
stringvar	Any string variable.
strindex	A zero-based integer specifying a character position in a string; possible values range from 0 to 79.
Sx	A "predefined" string variable, where *x* ranges from 0 to 9.
terminal	A valid terminal emulation name; used by the **EMULATE** and **SET EMULATION** commands.

A Note on Examples

To add clarity, examples in this manual are presented in lowercase. You may use either upper- or lowercase (or both) in your scripts to make your code easier to read; for instance, uppercase can be used to make command words or procedure names "stand out" in your script! The only case-sensitive elements within ASPECT are quoted strings and the format specifiers used in the **FATSAY**, **FSTRFMT** and **STRFMT** commands.

Using Remote ASPECT Commands

PROCOMM PLUS also allows a remote user to execute ASPECT script command statements (enabling you to control your computer from a distant location). Remote commands are sent from another computer with the sequence:

Ctrl-D *command* **<CR>**

where Ctrl-D (or ^D) = decimal 4 and **<CR>** = decimal 13. For example, to force your system to disconnect, a remote computer might send Ctrl-D HANGUP <CR>.

To send this same command from a Meta key or from a script file, the ^D and carriage return must be translated; therefore, when executed from a script the command would be **TRANSMIT "^DHANGUP^M"**.

This remote capability will work with all terminal emulations, but you must enable remote commands in the Setup facility (see the discussion of the Setup "General Options" in your USER MANUAL). Most users can't type fast enough to enter remote commands manually (a maximum of two seconds may elapse between characters); therefore, use Meta keys or script files to send them whenever possible.

You'll find more information on remote commands and an example of their use in Chapter 5 of this manual.

ADD

Adds the first two numbers and stores the result in the last numeric variable.

ADD *number number numvar*

Example

```
integer items=5
add items 12 N2
integer first=4, last=4, tot
add first last tot
```

N2 = ITEMS + 12

TOT = 8

Comments

Make sure the result of your addition isn't beyond the range of the numeric variable's data type; PROCOMM PLUS does no error checking on this value.

As with any mathematical expression, the result will be truncated (if necessary) to fit the range of values for the data type.

The *command* form "ADD N1 N2 N3" is equivalent to the *operator* form "N3=N1+N2".

See also

MUL and INC.

ALARM

Sounds an alarm to alert you to an event.

ALARM [*integer*]

integer	An integer value that specifies the amount of time the alarm will sound (in seconds). The default value is the Alarm Time specified in the Display/Sound Options Setup submenu. This parameter can also be set with the **SET ATIME** command.

Example

ALARM 5	Sounds the alarm for 5 seconds. If the optional integer is omitted, the Setup default will be used.

Comments

If the **SET ALARM** command is OFF or if Alarm Sound in the Setup is set OFF, this command has the same result as a **PAUSE** command.

See also

SET ATIME, SET ALARM and SOUND.

AND

This command performs a bitwise comparison of two numbers and places the result in the specified numeric variable. For each two bits compared, the resulting variable is assigned the value 1 or 0 in the same corresponding bit position. A 1 is assigned if both bits are 1, while a 0 is assigned if either or both of the bits are 0.

AND *number number numvar*

Comments

AND cannot be used with floating point numbers.

The *command* form "AND N0 N1 N2" is equivalent to the *operator* form "N2=N0&N1".

See also

OR and XOR.

ANDL

This command performs a logical-AND operation on two numbers and places the result in the specified integer variable. The result is 1 if both numbers are non-zero. The result is 0 if either or both of the numbers are zero.

ANDL *number number intvar*

Comments

The *command* form "ANDL N0 N1 N2" is equivalent to the *operator* form "N2=N0&&N1".

See also

OR and ANDL.

ASSIGN

Assigns a text string or another string variable to a string variable.

ASSIGN *strvar string* [*length*]

Examples

string digits
assign digits "12345"

Sets the value of variable DIGITS to 12345; the *operator* equivalent for this statement would be DIGITS="12345".

string astr, bstr
assign astr bstr

Sets the value of variable ASTR to the value contained in string variable BSTR. The *operator* equivalent for this statement would be "ASTR=BSTR".

Comments

The optional length value determines the maximum number of characters to copy from *string*.

See also

STRCPY, STRCAT, STRUPDT, SUBSTR and STRSET.

ATGET

Relocates the cursor at a row and column and gets a string or number (entered at the keyboard) using the specified color attributes.

ATGET *row col attribute length strvar | numvar* [DEFAULT]

length	An integer value between 1 and 80 that determines the maximum number of characters that will be accepted. A beep will sound if the user attempts to enter more characters than you specify.
strvar | *numvar*	The variable where the string or number is stored.
[DEFAULT]	An optional parameter indicating that the present variable contents should be displayed as a default entry. If the user presses Enter alone, the default value will be used.

Example

```
string response

vidsave 0                                  Saves the screen.
box 5 15 9 60 7                            Displays box.
atsay 7 18 15 "Name: "                     Gives prompt.
atget 7 25 112 32 response                 On row 7 column 25, using reverse
                                           video (112), get a 32-character string
                                           into RESPONSE.
if success
   fatsay 24 0 15 "  Your name is %s  " response
   pause 3
endif

vidrest 0
```

Comments

When responding to an ATGET command, the user must press Enter to signal that input is complete. The carriage return is not included in the text string stored in a string variable.

The cursor position will remain at the location where the user pressed Enter or Esc.

ATGET can be exited with the Esc key (the **IF FAILURE** statement returns "TRUE").

See also

GET, KEYGET, MATGET, ATSAY, FATSAY and MGET.

ATOF

Converts the contents of an ASCII string to a floating point value.

ATOF *string floatvar*

string	String to convert to a float variable. Conversion begins at the first floating point digit.
floatvar	Variable where the float value is stored.

Example

```
float amount=0
atof "$5.47" amount
```

AMOUNT = 5.47.

Comments

If the string doesn't contain a float value, ATOF returns a value of 0.

See also

ATOI, ATOL and FTOA.

ATOI

Converts the contents of an ASCII string to an integer value.

ATOI *string intvar*

string	String to convert to an integer. Conversion begins at the first digit.
intvar	Variable where the integer is stored.

Examples

```
string temp="12345"
atoi temp n0
```

N0 = 12345.

```
string test="123the end"
atoi test digits
```

DIGITS = 123.

```
integer answer
assign s3 "   345  7"
atoi s3 answer
```

ANSWER = 345.

```
integer intnum
string title="The end..."
atoi title intnum
```

INTNUM = 0.

Comments

The function stops reading the string at the first non-numeric character and then processes the numeric characters into the corresponding integer value. If the string doesn't contain an integer value, ATOI returns a value of 0.

Conversion begins at the first digit encountered in the string.

ATOI will not convert hexadecimal numbers.

See also

LTOA, FTOA and ITOA.

ATOL

Converts the contents of an ASCII string to a long value.

ATOL *string longvar*

string	String to convert to a long value.
longvar	Variable where the long is stored.

Example

```
long speed
atol "38400" speed
```

SPEED = 38400.

Comments

The function starts reading at the first digit encountered and stops reading the string at the first non-numeric character; ATOL then processes the numeric characters into the corresponding long value. If the string doesn't contain a long value, ATOL returns a value of 0.

ATOL will not convert hexadecimal numbers.

See also

ATOI, ATOF and LTOA.

ATSAY

Displays a constant or variable on the local screen at the specified position and with the specified attribute colors.

ATSAY *row column attribute string | number*

Example

atsay 10 0 4 "Hello, Bruce!"	Displays "Hello, Bruce!" in red at row 10, column 0.

See also

MESSAGE, ATGET and FATSAY.

BLANKON

Sets the foreground attribute equal to the background attribute, effectively blanking any subsequently-displayed characters.

BLANKON

See also

BLINKON, BOLDON, DIMON and NORMON.

BLINKON

Turns on the blinking attribute for Terminal mode display.

BLINKON

See also

REVON, BOLDON, DIMON and NORMON.

BOLDON

Turns on the bold attribute for Terminal mode display.

BOLDON

See also

REVON, ULINEON, DIMON and NORMON.

BOX

Draws a colored box on the screen.

BOX *row column row column attribute*

row	Row number of the upper-left corner of the box.
column	Column number of the upper-left corner of the box.
row	Row number of the lower-right corner of the box.
column	Column number of the lower-right corner of the box.

Example

box 0 0 23 10 79	Draws a white framed box with red background from row 0, column 0 (upper-left corner) to row 23, column 10 (lower-right corner).

See also

ATSAY and SCROLL.

BREAK

Sends a break to a remote computer system.

BREAK

Examples

set break 500	Sets the break length to 500 milliseconds.
BREAK	Sends a break.

Comments

The length of the break can be specified within Setup (or with the SET BREAK command).

See also

SET BREAK.

BYE

Terminates the executing script file and exits PROCOMM PLUS without disconnecting.

BYE

Example

transmit "Logoff" **bye**	Sends a message and quits PROCOMM PLUS (the connection is not broken).

Comments

To perform the same function and disconnect the user, use the QUIT command.

See also

CONNECT, EXIT, QUIT and TERMINAL.

CALL

Branches to a specified procedure and allows a return.

CALL *name* [WITH *param*[*param*]...[*param*]]

Example

```
proc main
   integer number=1
   call proc2 with number
   call proc3 with &number
endproc
```

This structured MAIN procedure CALLs two other procedures, passing NUMBER to each. Note that the "&" symbol is used in the CALL to PROC3.

```
proc2
   intparm x
   ++x
endproc
```

The value for NUMBER remains 1 in the MAIN procedure (it was passed by *value*).

```
proc3
   intparm y
   ++y
endproc
```

The value for NUMBER now becomes 2 in the MAIN procedure (it was passed by *reference*).

Comments

An optional parameter list can be included with the CALL command; up to 10 parameters can be specified, and their order must be the same as the order in which they are declared in the CALLed procedure. A parameter can be a constant or a variable.

Variables are normally passed by *value*, where only the variable's value is sent and changes made to it are not returned to the CALLing procedure (the variable remains unaffected).

The "&" character may precede a variable parameter to indicate that it's being passed by *reference*; any changes made to the corresponding parameter by that procedure will be recorded in the variable after control returns.

See also

RETURN, PROC, ENDPROC and GOSUB.

CASE

Provides conditional **SWITCH** command processing based on a match between a specified string or integer and the target string or integer.

CASE *integer|string*

Comments

A match (case-insensitive) causes processing to continue with each command on subsequent lines until the **ENDCASE** command is encountered. No match results in the processing of any subsequent **CASE** or **DEFAULT** command.

The **SET SWITCHCASE** command allows case-sensitive string matches.

See the **SWITCH** command for further information on CASE.

See also

SWITCH, DEFAULT, EXITSWITCH, ENDCASE and SET SWITCHCASE.

CEIL

Computes the smallest integer value (with no fractional amount) *greater than* or *equal* to a floating point number.

CEIL *float numvar*

Example

```
float price=3.2
integer high
ceil price high
```

HIGH = 4.

See also

FLOOR.

CHDIR

Changes the current directory and/or drive.

CHDIR *filespec*

filespec	Identifies the DOS drive (for example, *A:*), pathname and directory. The drive identifier is only required if you change the current drive.

Examples

chdir "A:\"	Changes the current drive to the root directory of drive A.
chdir "C:\COMM"	Changes the current drive to C: and the current directory to \COMM.
chdir "\COMM\PT2"	Changes the current directory to \COMM\PT2.

See also

MKDIR, RMDIR and GETDIR.

CLEAR

Clears the screen with the specified colors (changing the default color used for terminal displays).

CLEAR [*attribute*]

attribute	The optional attribute value allows you to change the background and foreground colors and to reset the current colors. Legal values are 0 - 255.

Examples

clear	Clears screen to currently set colors.
clear 31	Clears the screen to a Blue background and a high intensity White foreground and also resets the current colors.

Comments

For a complete listing of attribute values, see the section titled "ASPECT Conventions" earlier in this chapter.

See also

SCROLL.

COMGETC

Assigns a numeric variable to the next character value in the receive data buffer.

COMGETC *intvar*

Example

```
integer nxchar
comgetc nxchar
```

Assigns the variable NXCHAR to the value of the next receive buffer character.

Comments

If the receive buffer is empty, COMGETC returns a value of -1.

Any active **WHEN** statement will be checked against the character value retrieved by COMGETC.

See also

COMGETCD and SET RXDATA.

COMGETCD

Pauses up to two seconds, then assigns a numeric variable to the next character value in the receive data buffer.

COMGETCD *intvar*

Example

```
integer newchar
comgetcd newchar
```

Assigns the variable NEWCHAR to the value of the next receive buffer character.

Comments

Similar to **COMGETC**, COMGETCD will wait up to two seconds for a character to enter the receive buffer; if the buffer remains empty, COMGETCD returns a value of -1.

Any active **WHEN** statement will be checked against the character value retrieved by COMGETCD.

See also

COMGETC and SET RXDATA.

COMMENT

Denotes the start of a comment block. Any commands or remarks following this command are ignored by the compiler until an **ENDCOMMENT** is found. COMMENT blocks cannot be nested.

COMMENT

See also

ENDCOMMENT.

COMP

This command performs a bitwise complement (1's complement) of a number and places the result in the specified numeric variable. For each bit in the number, the resulting variable is assigned the value 1 or 0 in the same corresponding bit position. A 1 is assigned for each 0 bit in the number. A zero is assigned for each 1 bit in the number.

COMP *number numvar*

Comments

COMP cannot be used with floating point numbers.

The *command* form "COMP N0 N2" is equivalent to the *operator* form "N2=~N0".

See also

OR and XOR.

COMPUTC

Sends the specified character value at the communications port.

COMPUTC *integer*

Example

computc 'A'	Sends the character A out the active port.
computc 94	Sends the ^ character out the active port.

Comments

Range checking is suppressed for COMPUTC. Use **TERMKEY** to send the current emulation's Keyboard Mapping code for keys like function and cursor control keys.

See also

COMGETC and TERMKEY.

CONNECT

Exits the script file and returns to Terminal mode.

CONNECT

Comments

EXIT, TERMINAL and CONNECT all perform the same function.

See also

BYE, EXIT, QUIT, and TERMINAL.

CURDN

Moves the cursor down one line.

CURDN

See also

CURLF, CURRT and CURUP.

CURLF

Moves the cursor one column to the left.

CURLF

See also

RCA, CURRT and CURUP.

CUROFF

Turns cursor off.

CUROFF

See also

CURON, GETCUR, LOCATE and SET BLOCKCUR.

CURON

Turns cursor on.

CURON

See also

CUROFF, GETCUR, LOCATE and SET BLOCKCUR.

CURRT

Moves the cursor one column to the right.

CURRT

See also

RCA, HOME and CURUP.

CURUP

Moves the cursor up one line.

CURUP

See also

RCA, HOME and CURDN.

CWHEN

Deactivates an active **WHEN** command.

CWHEN *index* | DISCONNECT

Example

```
when 0 "MORE?" \
transmit "Y^M"
CWHEN 0
```

Deactivates a previous **WHEN** command indexed as 0. Note the use of the backslash (\) extender character.

Comments

The integer parameter identifies which of the **WHEN** commands to deactivate—up to three **WHEN** commands can be active, with valid index values of 0, 1 and 2.

The DISCONNECT option allows the clearing of a previous **WHEN DISCONNECT** command.

See also

WHEN.

DATE

Gets current system date into a string variable.

DATE *strvar*

strvar	The variable where the date is placed, in the format MM/DD/YY.

See also

TIME and $DATE system variable.

DEC

Decrements a numeric variable by one.

DEC *numvar*

Comments

It is possible to decrement a number beyond the range of the numeric variable's data type; ASPECT does no error checking on this value.

The operator equivalent for the statement "DEC N1" is "N1–".

See also

INC and SUB.

DEFAULT

Provides for processing when no match is found among the **CASE** statements in a **SWITCH** command block. See the **SWITCH** command for further information.

DEFAULT

Comments

DEFAULT can occur anywhere among the separate **CASE** statements.

See also

SWITCH, CASE and EXITSWITCH.

DEFINE

Defines a macro name and the text to substitute for it during compilation.

DEFINE *name text*

Examples

define red 7	Defines the macro RED as the string value 7.
define myname "Anne J." **transmit myname**	Defines the macro MYNAME as the quoted text string "Anne J." and transmits it to the remote system.
define wblue 31 **define bgrey 113** **atsay 10 20 wblue "NAME: "** **atget 10 20 bgrey "PASSWORD:"**	Defines the macros WBLUE and BGREY and uses them as attributes for later ATSAY statements.

Comments

Similar to macros in the "C" programming language, DEFINE allows for *textual substitution* within a source program.

A macro definition is activated at the point where the DEFINE occurs, and it remains active until the definition is removed (with the UNDEF command) or the source program ends.

If an active macro name is encountered. the text DEFINEd for that macro is substituted for the name, and compilation continues at the beginning of the substituted text; this can save time and keystrokes when writing source programs, since often-repeated text can be inserted with macros. If you must frequently modify a script, macros can make that task much easier, too (simply change the text of a macro instead of each occurrence of the macro name).

You can also test for the existence of a DEFINEd macro name (the text portion of the DEFINE is ignored); use $IFDEF or $ELSEIF to perform *conditional compilation.* If the name exists, the commands which follow (from the $IFDEF or $ELSEIF up to the corresponding $ELSEIF, $ELSE or $ENDIF) will be included in the compiled script—otherwise, the commands will be ignored. The $ELSE command will cause all commands (up to the corresponding $ENDIF) to be included if the previous $IFDEF or $ELSEIF condition had failed. $IFDEF commands can be nested.

There is no limit to the number of macros within a source program. Macro text can consist of zero or more items, each separated by a space.

You can also DEFINE a macro at compile time with the **ASPCOMP /D** switch. Quotation marks must be used if the macro DEFINEs a string, and the command line allows a maximum of one token in the macro text. Your macro will be expanded during compilation to include the command line data.

To reDEFINE a macro, you must first remove its current definition with the **UNDEF** command.

Any occurrence of a macro name prior to its definition (except within a **$IFDEF** or **$ELSEIF** command) will generate an error.

For more advanced examples of DEFINE, refer to Chapter 5 of this manual.

See also

$IFDEF, $ENDIF, $ELSEIF and UNDEF.

DELETE

Deletes a specified file.

DELETE *filespec*

Example

```
string file="temp.fil"
delete  file
if failure
   fatsay "Can't delete %s" file
endif
```

Deletes the file TEMP.FIL. If the operation fails, an error message is displayed.

Comments

The filespec may include a directory path. Wildcards are not allowed.

A DELETE operation can be tested with the **IF SUCCESS** and **IF FAILURE** statements.

DELCHAR

Deletes the character at the current cursor position. Any text from the current cursor position to the end of the line will be shifted one column to the left.

DELCHAR

See also

DELLINE and INSLINE.

DELLINE

Deletes the entire line at the current cursor position. Lines below the current cursor position will be scrolled up one line.

DELLINE

See also

DELLINE and INSLINE.

DIAL

Calls an entry in your Dialing Directory.

DIAL "[*ldcode*]*entry*[*ldcode*]..." | *strvar*[*strvar*]

entry	Indicates the Dialing Directory entry number you wish to call. Ensure that this number accurately represents the entry you wish to use; if you sort your Dialing Directory, this number may change and your script will have to be updated.
ldcode	An entry can be prefixed and/or appended with a dialing code (see "Adding or Changing a Dialing Code" in your USER MANUAL).
strvar	A variable which you can set to a valid Directory entry number and use in place of a numeric entry.

strvar	A variable in which PROCOMM PLUS returns the number of the Dialing Directory entry with which a connection has been made.

Examples

dial "5"	Dials entry 5 in the Dialing Directory.
dial "A5B"	Dials code A, Dialing Directory entry 5 and ldcode B.
string numbers **assign numbers "1 2 3 4"** **dial numbers s1**	Dials entry numbers 1 through 4 until a connection is made, and stores in S1 the number of the entry with which the connection was made.
if not fromddir **dial "5"** **endif**	Calls entry 5, unless the script file is linked to the Dialing Directory.

Comments

Each entry in the dialing string must be separated from the next by a space or comma; however, no spaces should occur within an entry (including its *ldcodes*).

If either the dialing string or string variable contain a **list** of entry numbers, they are put into a dialing *queue* (for more information about the dialing queue, see your USER MANUAL).

The **IF NOT FROMDDIR** statement can be used to check if the script was executed from a Dialing Directory entry.

See also

DLOAD, MDIAL and REDIAL.

DIMON

Turns on the dim attribute for Terminal mode display.

DIMON

See also

BLINKON, BOLDON, ULINEON and NORMON.

DIR

Displays a paginated list of files.

DIR *[filespec]*

filespec	A valid path/filename or filename search pattern. Wildcards are allowed, and this value can be a variable.

Example

```
string spec
assign spec "*.ASP"
dir spec
dir "C:\PCPLUS\*.ASX"
```

Displays all files with the extension *.ASP* in the current directory.

Displays all files in the directory C:\PCPLUS with the extension *.ASX*.

Comments

This is the script equivalent of Alt-F in Terminal mode. The default filespec is *.* (or all files).

DISKFREE

Returns the free disk space of the specified drive into a long variable.

DISKFREE

DISKFREE *disk longvar*

disk	An integer number designating the drive; 0 indicates the current drive, 1 specifies drive A, 2 indicates drive B and so on.

Example

```
long free
diskfree 0 free
```

Places the free space available on the current drive into the long variable FREE.

See also

GETDIR and MEMFREE.

DIV

Divides the first number by the second number and stores the result in the last numeric variable.

DIV *number number numvar*

number	The dividend.
number	The divisor.
numvar	The result of the division.

Examples

init hundred=100, ten=10, result **div hundred ten result**	RESULT = HUNDRED/TEN = 100/10 = 10.
div hundred 33 N9	N9 = HUNDRED/33 = 100/33 = 3.
div 369 3 figure	FIGURE = 369/3 = 123.

Comments

For integer and long values, any remainder from the division will be discarded (for example: 3/2 = 1, 2/3 = 0).

Any division by 0 in your script results in an error message during execution.

See also

MOD and MUL.

DLOAD

Loads a different Dialing Directory.

DLOAD *filespec*

Comments

The .DIR file extension is optional.

If the directory does not exist (either in the current directory or in the directory specified with the "SET PCPLUS=" environment variable), DLOAD will create an empty directory with that name.

DOS

Executes a DOS command or another program within a PROCOMM PLUS script file.

DOS *string* [WAIT | NOCLEAR]

string	Any executable command as it would appear on the DOS command line (optionally including a DOS path).
WAIT	Suspends processing after termination of the DOS function until the user presses a key.
NOCLEAR	Prevents PROCOMM PLUS from saving and clearing the screen before executing the command (and restoring the screen upon returning from the program).

Examples

```
string c1
c1="TYPE temp.fil > PRN"
dos c1
```

Prints the file TEMP.FIL.

```
dos "del FILE.EXT" wait
```

Deletes the file FILE.EXT and waits for the user to press a key.

```
dos "sortdisk" noclear
```

Runs the program SORTDISK (without clearing the screen beforehand).

```
if failure
   message "Can't execute command!"
endif
```

Comments

If the path to the executable command is not specified, the program or external DOS command (such as FORMAT, which is not internal like DIR) must be either in the current directory or in the directory specified in your DOS path. Also, if you're using a hard disk system, COMMAND.COM must be in the directory from which you started your computer (typically the root directory of drive C) or in the location specified by the COMSPEC environment variable. If you're using a floppy disk system, COMMAND.COM must be on the disk in drive A (or in the directory found in COMSPEC).

This command can be tested with the IF SUCCESS statement, returning a "FALSE" if COMMAND.COM is not found and "TRUE" if it is found (even if the command to be executed is invalid or fails to execute). The RUN command provides a similar function and provides better testing for DOS external commands and other programs.

Be sure that you have enough memory to execute the specified command and PROCOMM PLUS at the same time. If you use the DOS command to execute a program or command that requires user input, be sure that anyone using the script file is aware of this, since processing will halt until the required input is provided.

Unlike the RUN command, the DOS command will execute both standard DOS batch files and internal DOS commands; RUN only accepts .COM and .EXE programs.

If NOCLEAR isn't specified, the screen and cursor position are saved, the cursor is moved to position 0,0 and the screen is cleared. After the DOS function has finished, the previous screen and cursor position are restored.

For more information, see Chapter 4 later in this manual.

See also

RUN, HOOK, METAKEY, MEMFREE and SHELL.

DOSVER

Returns the current DOS version into a string variable.

DOSVER *strvar*

Comments

Both the major and minor version numbers are included.

Example

```
STRING OS_ver
STRING OS_maj
STRING OS_min
DOSVER OS_ver
  SUBSTR os_maj os_ver 0 1
  SUBSTR os_min os_ver 2 2
```

Loads the DOS version into the string variable OS_VER. The SUBSTR command extracts the major and minor version numbers into OS_MAJ and OS_MIN, respectively.

DSCROLL

Scrolls an area of the screen down by the specified number of lines. This is similar to the BIOS scroll function.

DSCROLL *integer row column row column attribute*

integer	An integer value ranging between 0 and the number of rows on screen; this parameter determines the number of lines to scroll down within the defined area. If the value is 0, then the entire scroll area is blanked.
row column *row column*	Defines the upper left and lower right corners of the scroll area.

Comments

DSCROLL works identically to the **SCROLL** command (moving the screen area down instead of up).

See also

SCROLL.

EBOL

Erases text from the current cursor position to the beginning of the line.

EBOL

See also

EBOS, EEOL and EEOS.

EBOS

Erases text from the current cursor position to the beginning of the screen.

EBOS

See also

EBOL, EEOL and EEOS.

EEOL

Erases text from the current cursor position to the end of the line.

EEOL

See also

EBOL, EBOS and EEOS.

EEOS

Erases text from the current cursor position to the end of the screen.

EEOS

See also

EBOL, EBOS and EEOL.

ELSE

Executes an alternate set of commands when the conditions specified in the associated **IF** or **ELSEIF** commands evaluate as "FALSE". See the **IF** command for further information.

ELSE

Example

```
integer normcolor
if mono
   normcolor=7
else
   normcolor=31
endif
```

Sets the integer variable NORMCOLOR to 7 if the computer is using a monochrome adaptor; if the adaptor is anything else, NORMCOLOR becomes 31.

$ELSE

Allows conditional compilation by testing for the existence of a DEFINEd macro name. For more information, see the **DEFINE** command.

$ELSE

ELSEIF

Performs multiple tests of conditional expressions. **ELSEIF** can be used instead of nested **IF** ... **ENDIF** command blocks.

Conditions can include any numeric expression involving logical or numeric operators and operands. If the expression evaluates to a non-zero value, the the condition was satisfied (and is considered "TRUE"), while a value of 0 indicates that the test was "FALSE".

See the IF command for further information.

ELSEIF *condition*

Example

```
atget 5 5 31 1 choice
if choice == 1
   call menuitem1
elseif choice == 2
   call menuitem2
else
   call selecterr
endif
```

The integer variable CHOICE determines whether to CALL either MENUITEM1 or MENUITEM2. If CHOICE is not equal to 1 or 2, the procedure SELECTERR is CALLed instead.

$ELSEIF

Allows conditional compilation by testing for the existence of a DEFINEd macro name. For more information, see the DEFINE command.

$ELSEIF

EMULATE

Changes the active emulation to the terminal type specified.

EMULATE *terminal*

terminal — Valid terminal types are: TTY, VT52, VT100, VT102, VT220, VT320, ANSI, IBMPC, WYSE75, ATT605, ATT4410, TV922, H19, IBM3101, IBM3161, DGD100, DGD200, DGD210, ADDS60, ADDS90, ADM3, ADM5, ADM31, ESPRIT3, IBM3270, TV910, TV912, TV920, TV925, TV950, TV955, WYSE50 and WYSE100.

Example

```
emulate VT102
```

Changes emulation to the VT102 terminal type.

Comments

The EMULATE command will clear the screen—if this isn't desirable, use the **VIDSAVE** and **VIDREST** commands to save the old screen and restore it after you've switched emulations.

See also

VIDSAVE, VIDREST and SET EMULATION.

ENDCASE

Concludes the **CASE** command within the **SWITCH** command phrase. See the **SWITCH** command for further information.

ENDCASE

Example

```
integer resp
atget 5 5 31 1 resp
switch resp
   case 1
      call menu1
   endcase
   case 2
      call menu2
   endcase
   default
      call selecterr
   endcase
endswitch
```

The integer variable RESP determines whether to **CALL** either MENU1 or MENU2. If RESP is not equal to 1 or 2, the procedure SELECTERR is **CALLed** instead.

ENDCOMMENT

Marks the end of a comment block; commands which follow are no longer ignored.

ENDCOMMENT

See also

COMMENT.

ENDFOR

Concludes the FOR command block. See the FOR command for further information.

ENDFOR

ENDIF

Concludes the IF command block. See the IF command for further information.

ENDIF

$ENDIF

Concludes the IF command block, allowing conditional compilation by testing for the existence of a DEFINEd macro name. For more information, see the DEFINE command.

$ENDIF

ENDPROC

Terminates a procedure block. See the PROC command for further information.

ENDPROC

Comments

ENDPROC always includes a RETURN to the CALLing procedure; the RETURN statement isn't needed.

ENDSWITCH

Terminates the **SWITCH** command block. See the **SWITCH** command for further information.

ENDSWITCH

ENDWHILE

Terminates the **WHILE** command block. See the **WHILE** command for further information.

ENDWHILE

EOF

Tests whether or not the end of a file opened with the **FOPEN** command has been reached.

EOF *index intvar*

Example

```
define true 1
integer testeof=0
fopen 0 "test.fil"
fgets 0 s1
eof 0 testeof
if testeof
   message "EOF reached!"
endif
```

Tests the file corresponding to index number 1 for end-of-file and places the result in the integer variable TEST.

Comments

The EOF condition becomes TRUE after the first operation that attempts to read past the end of a file. It will remain TRUE until an **FSEEK**, **FCLEAR** or **REWIND** is performed.

The integer variable will be initialized to 0 for "FALSE" and 1 for "TRUE".

The conditional form of this command is "IF/ELSEIF/WHILE EOF INDEX".

See also

FSEEK, REWIND, IF EOF and FCLEAR.

EQ

Tests two numeric values for equality.

EQ *number number intvar*

Example

```
integer price,total
eq price total n1
```

Tests the the values PRICE and TOTAL, placing the result in the predefined numeric variable N1.

Comments

The integer variable will be initialized to 0 for "FALSE" and any non-zero value for "TRUE".

The conditional form of this command is "IF/ELSEIF/WHILE EQ NUMBER NUMBER", while the operator form is "NUMBER==NUMBER".

See also

NEQ.

ERRORMSG

Displays an error message centered in a box on the local screen.

ERRORMSG *string*

string — A string of up to 80 characters.

Comments

An error sound accompanies the message. The message can be erased by pressing a key (or waiting two seconds for the message to erase automatically).

See also

USERMSG and STATMSG.

EXECUTE

Executes another script (either a source program or a compiled script); this process is called "chaining". The currently executing script terminates without return.

EXECUTE *filespec*

filespec	Any valid script filename. The file extensions .ASX and .ASP are not required.

Examples

```
execute "CALL_tom"
```

Executes the script file CALL_TOM.ASX.

```
string next_file
get S0
switch S0
   case "A"
      assign next_file "CHOICEA.ASX"
   endcase
   case "B"
      assign next_file "CHOICEB.ASX"
   endcase
   default
      assign next_file "DEFAULT.ASX"
   endcase
endswitch
execute next_file
```

Assigns a filename to NEXT_FILE based on the value of S0 and executes that file.

Comments

If the extension is ommitted, EXECUTE will first search for a compiled script (with the .ASX extension). If a compiled script that matches the filespec can't be found, EXECUTE will attempt to compile a matching source file (with the .ASP extension). If an .ASP file is specified, ASPECT always attempts compilation.

Only predefined variables (S0-S9, N0-N9) can be used to pass values to an EXECUTEd script; user-defined variables will **not** retain their values.

The script will terminate if the specified filespec doesn't exist or can't be compiled.

EXIT

Terminates the executing script file and returns the user to Terminal Mode.

EXIT

Examples

```
transmit "Goodbye"
hangup
exit
```

Transmits closing message "Goodbye" and disconnects. The user is then returned to Terminal mode.

Comments

CONNECT, TERMINAL and EXIT are all synonyms; all three commands perform the same function.

See also

BYE, CONNECT, QUIT and TERMINAL.

EXITFOR

Continues execution with the command after an **ENDFOR** command. For further information, see the **FOR** command.

EXITFOR

EXITSWITCH

Transfers control from within a **CASE** or **DEFAULT** command block to the line following the **ENDSWITCH** command. **EXITSWITCH** is similar to the "C" language BREAK command. For more information, see the **SWITCH** command.

EXITSWITCH

EXITWHILE

Continues execution with the command after an **ENDWHILE** command. For further information, see the **WHILE** command.

EXITWHILE

FATSAY

Displays formatted text on the local screen at the specified location and with the specified color attributes. **FATSAY** is similar to "printf" in the "C" programming language.

FATSAY *row column attribute formatstr* [*param*]...[*param*]

row - the row at which the text will be displayed. This value is zero-based (0 is the top row of your display, 23 means the 24th row).

column - the column at which the first character of the specified text will be displayed. This value is also zero-based.

attribute - the video attribute of the displayed text. For a full listing of the available attributes, please refer to the "**ASPECT** Conventions" section at the beginning of this chapter.

formatstr - the format string, which is either a quoted string or a string variable. It may contain not only text but special reserved characters (called "format specifiers") that indicate how to display optional parameters as part of the text. Each format specifier corresponds to a variable or constant (*param*) that appears at the end of the command. For details on the *formatstr* see below.

param - an optional variable or constant whose value may be converted to a specified format and displayed as part of the text. This variable or constant may be any valid type; the method by which it's displayed is determined by its corresponding format specifier in the format string. If there are multiple *parameters*, the first *param* corresponds with the first format specifier in the format string, the second variable with the second format specifier and so on. Only those *params* with corresponding format specifiers will be displayed; for example, if there are three *params* and only two format specifiers, the values of only the first two *params* will be displayed.

Structure of the Format String

The format string consists of zero or more format specifiers interspersed with text. At its simplest, FATSAY could be used to simply display a string (like the **ATSAY** command). For example, the statement:

```
fatsay 0 0 31 "This is literal text."
```

displays "This is literal text." (**without** the quotation marks) in high-intensity white on a blue background. The display begins at the first row and first column.

However, the format string may also contain an assortment of format specifiers that correspond with optional variables or constants (*params*) following the format string. Format specifiers and their corresponding *params* are processed from left to right. A format specifier is constructed as follows:

%[flag][width][.precision]type

where:

% - indicates that this is the beginning of the format specifier (rather than normal text). Characters following this special marker describe the format for displaying the corresponding *param*. If the % sign is followed by a character that has no special meaning as a format specifier (if it is not followed by a valid *type*), then that character and any subsequent characters until the next % are simply displayed as normal text. To display a percent sign, use "%%".

flag - valid flags are "-", "+", " " (a space), and "#". These flags are optional.

"-" left justifies the converted *param* within the number of places indicated in the *width* field. Without this optional flag, the displayed text is right justified.

"+" prefixes the converted *param* with a plus or minus sign (if it represents a signed type). Without this flag, the text will be displayed with a sign only if it represents a negative number.

" " - a space immediately after the % marker prefixes the converted *param* with a space if it represents a signed positive value (the space is ignored if both the blank and "+" flags appear.)

"#" - prefixes the converted *param* with the characters "0x" (if it represents a hexadecimal value) or with "0" (if it represents an octal value). In addition the # flag can be used to force a decimal point for all float conversions (even if there is no fractional portion). This flag is ignored if used with any format *types* other than *x, X, o, f, e, E, g,* or *G*. For example:

```
integer num1 = 16
float num2 = 16
fatsay 0 0 31 "%+d" num1          ; outputs "+16"
fatsay 1 0 31 "% d" num1          ; outputs " 16"
fatsay 2 0 31 "%#x" num1          ; outputs "0x10"
fatsay 3 0 31 "%#f" num1          ; outputs "16."
```

width - this optional value specifies the minimum number of characters that will be generated for the conversion from the corresponding *param*. If the width value is greater than the number of characters in the converted text, it will be padded on the left with spaces (unless this has been altered by one of the *flags*). A 0 immediately preceding the width field pads the converted text with leading zeros after any sign or prefix (as long as no other padding has been specified). Note that since the width field specifies the *minimum* number of generated characters, it will never cause a value to be truncated.

```
integer num1 = 16
fatsay 0 0 31 "%d" num1           ; outputs "16"
fatsay 1 0 31 "%05d" num1         ; outputs "00016"
fatsay 2 0 31 "%-5d" num1         ; outputs "16  "
```

precision - this optional value must always be preceded by a period (.); it specifies the minimum number of digits to display when converting an integer *param* to displayed text, the number of decimal places to display on a float conversion (using the *e, E* , or *f types*), the maximum number of siginificant digits to display on a *g* or G conversion or the maximum number of characters to display on a string conversion (using the *s type*). The default precision for floating point values may be specified using the **SET DECIMAL** statement. The *width* and/or *precision* fields may be an asterisk (*), which means that the value is supplied from the *param* list following the format string. In this case, the extra *param* would be an integer type preceding the *param* to be converted. Since *precision* specifies the maximum number of digits, it may cause truncation of strings or rounding of floats. For example:

```
string string1 = "This is a string"
float num1 = 1234.567
fatsay 0 0 31 "%7.4s" string1          ; outputs "   This"
fatsay 2 0 31 "%-*.*f" 8 1 num1        ; outputs "1234.5  "
```

type - this specifier determines the data type required of the corresponding *param* (if any) and how the FATSAY command will alter the *param*'s value before converting it to displayed text. **Please note that the following *type* specifiers are case sensitive; lower case *type*s must be entered in lower case and upper case specifiers must be entered in upper case!**

d or *i- param* is converted to signed decimal notation. The *d* or *i* may be preceded by an *l* (lower case) to specify a long integer type.

u - param is output as an unsigned decimal number. The *u* may be preceded by an *l* to specify a long unsigned integer type.

o - param is output in octal notation. The *o* may be preceded by an *l* to specify a long octal type.

x - param is output in hexadecimal notation using the characters 0-9 and a-f.

X - *param* is output in hexadecimal notation using the characters 0-9 and A-F.

f - param is output in the form *[-]dddd.dddd*, where *dddd* is one or more decimal digits. The number of digits displayed after the decimal point is determined by the *precision* specification (see above); the default precision is 2. A minus sign is displayed if the value is less than zero, but a plus sign is displayed only if called for with the "+" *flag*.

e - param is output in the form *[-]d.dddd* **e** *sign ddd*, where *d* is a decimal digit, *dddd* is one or more decimal digits, *ddd* is three decimal digits, and the *sign* is "+" or "-" (scientific notation).

E - same as *e* above, except that this type uses a capital **E** instead of lower case **e** to prefix the exponent.

g - param is output in *f* or *e* format, as appropriate. If the *e* conversion would yield an exponent greater than -4 or less than the specified precision, the *param* value is output in *f* format instead. The generated text has no trailing zeros in any fraction and includes a decimal point only if there are fractional digits or if you specify the "#" *flag*.

G - same as *g* above, except the *E* output format is used instead of the *e* format.

c - integer *param* is output as a single ASCII character.

s - string *param* is displayed up to the first null character or until the *precision* value is reached.

```
integer num1 = 20, num2 = 30
string string1 = "This is a string"
float float1 = 1234.567
fatsay 0 0 31 "Print %d and %d" num1 num2     ; outputs "Print 20 and 30"
fatsay 0 0 31 "To %d and %#X" num1 num2       ; outputs "To 20 and 0X1E"
fatsay 0 0 31 "%7.4s" string1                 ; outputs "  This"
fatsay 2 0 31 "%-8.1f" num1                   ; outputs "1234.5  "
```

See also

STRFMT and FSTRFMT.

FCLEAR

Clears all end-of-file and error flags associated with the indicated file index number.

FCLEAR *index*

Examples

```
fclear 0
```

Clears EOF and error flags for the file corresponding to index 0.

See also

FOPEN and EOF.

FCLOSE

Closes the file corresponding to the provided index.

FCLOSE *index*

Example

fclose 0

Closes the file corresponding to index 0.

Comments

The I/O buffer contents for a file will automatically be written before it's closed (if it was opened in WRITE or APPEND mode).

This command can be tested with the **IF SUCCESS** statement, returning "FALSE" if an error occurred while closing the file.

See also

FOPEN and FFLUSH.

FETCH

Returns the current value(s) of any **SET** command parameter.

FETCH *param value*

param — The desired **SET** command parameter.

value — The current value of the specified **SET** command parameter(s). Note that this value **must** be returned to the proper data type; the variable may be in *intvar*, *strvar* or *longvar* format. The definition for **SET** includes all possible values for each command parameter.

Example

```
long speed
fetch baudrate speed
```

Places the current session baud rate into the long variable SPEED.

Comments

The following terminal emulation types are returned by the **FETCH EMULATION** statement:

TTY	0	DGD210	17
VT52	1	ADDS60	18
VT100	2	ADDS90	19
VT102	3	ADM3	20
VT220	4	ADM5	21
VT320	5	ADM31	22
ANSI	6	ESPRIT3	23
IBMPC	7	IBM3270	24
WYSE75	8	TV910	25
ATT605	9	TV912	26
ATT4410	10	TV920	27
TV922	11	TV925	28
H19	12	TV950	29
IBM3101	13	TV955	30
IBM3161	14	WYSE50	31
DGD100	15	WYSE100	32
DGD200	16		

The following file transfer protocol types are returned by the **FETCH DLXPROTOX/PROTOCOL/ULXPROTOX** statements:

EXTPROTO1	0	YMODEMG	9
EXTPROTO2	1	ZMODEM	10
EXTPROTO3	2	1KXMODEM	11
CISB	3	1KXMODEMG	12
KERMIT	4	ASCII	13
MODEM7	5	IMODEM	14
SEALINK	6	RASCII	15
TELINK	7	WXMODEM	16
YMODEM	8	XMODEM	17

In general, the FETCHed value for parameters with OFF | ON syntax is "0" for OFF, "1" for ON; for parameters with NO | YES syntax, FETCHed values are typically "0" for NO, "1" for YES.

See also

SET.

FFLUSH

Writes the current I/O buffer contents to the file with the specified index.

FFLUSH *index*

Example

```
fflush 0
```

Writes the I/O buffer contents to the file corresponding to index 0.

Comments

If the file was opened in READ mode, the buffer contents are cleared.

In WRITE or APPEND mode, buffers are automatically flushed when they're full or when the file is closed.

This command can be tested with the **IF SUCCESS** statement, returning "FALSE" if an error occurred while writing the buffer contents to the file.

See also

FOPEN, FCLEAR and FCLOSE.

FGETC

Reads a character from the file with the specified index into a variable.

FGETC *index intvar*

Example

```
integer num
fopen 0 "test.fil"
fgetc 0 Num
```

Reads a character from the file corresponding to index 0 and places it in the integer variable NUM.

Comments

To make sure a character is available, first check the file status using the **IF EOF** statement or the **EOF** command.

This command can be tested with the IF SUCCESS statement, returning "FALSE" if an error occurred while reading the file.

See also

FSEEK, FTELL, FGETS, FCLOSE, FPUTC, EOF, FOPEN and FREAD.

FGETS

Reads a string from the file corresponding to a given index into a variable.

FGETS *index strvar*

Example

```
string info
fopen 0 "test.fil" "rt"
fgets 0 info
```

Reads a string from the file corresponding to index 0 and places it in the string variable INFO.

Comments

The reading will terminate when FGETS encounters the end-of-file, a line feed character or when 80 characters have been read.

The string will include the ending line feed (unless you SET FGETS_CRLF OFF). The IF EOF statement or EOF command can test the file before reading it, and the IF SUCCESS statement will return "FALSE" if an error occurred while reading the file.

See also

FSEEK, FTELL, FGETC, FCLOSE, FPUTS, EOF, FREAD, FOPEN and SET FGETS_CRLF.

FIND

Tests for an occurrence of the specified text within a variable.

FIND *string string* [*intvar*]

string — The second string contains the characters being searched.

[*intvar*]	An optional integer variable that's assigned the string index where the desired text was found. The index is zero-based (the first character begins at position 0).

Examples

```
string resp1

message "Enter the password:"
mget resp1
```

Receives information and masks its appearance with asterisks (*).

```
find resp1 "password"
```

Tests for the string "PASSWORD" in variable RESP1.

```
if not found
   message "Invalid password"
goto secure_breach
else
   call welcome
endif
```

Comments

Case-sensitivity can be controlled with the **SET FINDCASE** command.

This command can be tested with the **IF FOUND** statement; **FOUND** is set to 1 if the text occurs and 0 if it does not occur.

See also

SET FINDCASE.

FINDFIRST

Locates a disk file using a specification you provide.

FINDFIRST *filespec* [*string*]

string

An optional string specifying the attributes to be used for the search. If this string is omitted, only "normal" files with no attributes (or just a read-only and/or archive attribute set) will be included. Specifying Hidden ("H"), System ("S") or Directory ("D") will search those files as well as the normal files. If only a Volume label attribute ("V") is used, the volume label is the only item searched; the Volume attribute is ignored when searching with any other attribute (except when searching the root directory of a drive). If a volume label is found, the system variables $FILENAME and $FNAME will both contain up to 11 characters of the volume name, and $FEXT will be set to null.

Example

```
FINDFIRST "*.TXT"
```

Finds the first file in the current directory with an extension of .TXT.

```
string file1="ABC.*"
findfirst file1
if found
fatsay 1 1 7 "Got %s" $fname
endif
```

Finds the first file in the current directory matching the wildcard specification ABC.*. A message is displayed if the file is found; note that the system variable $FNAME is used in the FATSAY statement to display the filename without its extension.

Comments

Both wildcards and a full DOS path are accepted as part of filespec; FINDFIRST defaults to the current directory if a path isn't provided.

The success of the operation can be tested with the **IF FOUND** statement.

The file's name, extension, name and extension, size, date stamp, time stamp and attributes are stored in the system variables $FNAME, $FEXT, $FILENAME, $FSIZE, $FDATE, $FTIME and $FATTR respectively.

See also

FINDNEXT.

FINDNEXT

Locates additional disk files with the specification provided in a previous FINDFIRST command.

FINDNEXT

Example

```
string spec="*.txt"
integer another=0
integer arow=1
findfirst spec
if found
   another=1
   fatsay 1 1 7 "%s" $fname
endif
while another
   ++arow
   findnext
   if found
   fatsay arow 1 7 "%s" $fname
   else
      exitwhile
   endif
endwhile
```

Finds all files in the current directory with the extension *.TXT, displaying each one without their extension. The FINDNEXT command searches the current directory for another filename matching the same specification.

Comments

The file specification for this command is taken from the previous FINDFIRST command. The success of the operation can be tested with the IF FOUND statement.

As with FINDFIRST, the name, extension, name and extension, size, date stamp, time stamp and attributes are stored in the system variables $FNAME, $FEXT, $FILENAME, $FSIZE, $FDATE, $FTIME and $FATTR respectively.

See also

FINDFIRST.

FLOAT

Defines a global or local float variable.

FLOAT *name*[=*expression*][,*name*[=*expression*]]...

[=expression]	An initializer expression. It can contain operators, constants or previously-defined variables.

Examples

float item	Defines the float variable ITEM.
float item,cost = 2.32	Defines the float variables ITEM and COST. COST is initialized to "2.32".

Comments

Up to 128 *global variables* of each type can be defined (a maximum of 512 total). Global variables are defined anywhere within a source program **outside** of any procedure (typically, this is at the top of a source program, before any procedures appear).

A global variable can be referenced within any procedure; if a procedure changes a global variable, its previous value is not saved.

Globals cannot be passed to another script by chaining—only the predefined N*x* and S*x* variables can pass information to another script.

Local variables are defined with the same commands as globals; however, they can only be referenced within the procedure where they're defined (unlike globals, which are defined outside of any procedure), and only 128 total local variables (of any type) can be used in a source program.

The same local variable name can be used in more than one procedure, but each occurrence is a completely different case and has no effect on any other occurences. Local variables will accept initializer expressions (just like global variables).

Global and local variables must be defined before they are referenced; several variable definitions may occur on the same line if each definition is separated with a comma.

See also

FLOATPARM, LONG, STRING and INTEGER.

FLOATPARM

Defines a float parameter variable.

FLOATPARM *name*[, *name*]...

Examples

```
proc main
float x=1.1,y=2.2,z=3.3
   call proc2 with x y z
endproc

proc2
   floatparm a, b, c
endproc
```

Defines three float values and passes them to a second procedure. Within PROC2, A=1.1, B=2.2 and C=3.3.

Comments

Any procedure—except MAIN—can be defined with up to 10 *parameter variables*. Parameter variables are similar to local variables in that they may only be referenced within the procedure in which they're defined; unlike local variables, though, parameter values are automatically initialized when the procedure is called.

The **CALL** command allows values to be passed to the procedure being called; these values are used to initialize the parameters upon entering the procedure. Parameter variables must be defined at the top of a procedure (before any commands or local variables are processed), and the order in which they're defined determines the order in which they're read by the **CALLed** procedure. Additional checking on passed parameters is performed by **ASPCOMP** during compilation—any inconsistencies will result in error messages.

Several parameters of the same type may be defined on the same source line (if each definition is separated from the next by a comma).

See also

LONGPARM, INTPARM and STRPARM.

FLOOR

Computes the largest integer value (with no fractional amount) *less than* or *equal* to a floating point number.

FLOOR *float numvar*

See also

CEIL.

FOPEN

Opens a file in the indicated mode(s) and assigns it to a file index.

FOPEN *index filespec modes*

filespec	A valid filename.
modes	A string containing the desired access modes. Valid modes are READ ("r"), WRITE ("w") and APPEND ("a"). Valid attributes are TEXT ("t"), BINARY ("b") and READ/WRITE ("+").

Examples

`fopen 0 "test.txt" "wb"`	Opens the file TEST.TXT in BINARY WRITE mode and assigns it an index number of 0.
`fopen 1 "data.fil" "rt+"`	Opens the file DATA.FIL in TEXT READ/WRITE mode and assigns it an index number of 1.

Comments

WRITE mode always overwrites (or erases) the contents of an existing file, while APPEND will only allow write operations at the end of an existing file.

TEXT and BINARY determine the translation mode for new lines. In TEXT mode (the default), the carriage return/line feed combination is translated to a single line feed during the read (and the reverse when writing—a line feed is written as a carriage return/line feed pair). In BINARY mode, no translation is performed.

The "+" character can be appended to any mode to allow both READ and WRITE operations on the same file; before switching between READ and WRITE, however, ASPECT requires an **FSEEK** or **REWIND** command.

This command can be tested with the **IF SUCCESS** statement, returning "FALSE" if the file cannot be opened.

The file pointer position is always initially set at the start of the file (no matter what mode or modes you select).

A maximum of 6 files may be opened at once, with index numbers 0 through 5.

See also

FCLOSE, FTELL, FPUTC, FPUTS, FSEEK, REWIND and FWRITE.

FOR

Repeats a command or series of commands a specific number of times.

FOR *numvar*[=*expression*] UPTO | DOWNTO *number*
.
.
.
ENDFOR

[=expression]	Optional initialization expression for the loop variable.
UPTO | DOWNTO	Indicates whether the variable is to be incremented (by UPTO) or decremented (by DOWNTO) after each iteration. The value is inclusive; if UPTO is set to 25 and the variable is initialized to 1, the loop will be performed 25 times.
ENDFOR	Required terminator for the FOR command block.

Example

```
for row=1 upto 25
   call bat
endfor
```

Performs the BAT procedure and increments the value of ROW by 1 until ROW=25. Since ROW is initialized to 1, the loop will be performed 25 times.

Comments

The LOOPFOR and EXITFOR commands allow branching to the next iteration test or command following the ENDFOR.

See also

WHILE, LOOPFOR and EXITFOR.

FPUTC

Writes an ASCII character value to the indicated file.

FPUTC *index character*

character The character can be any ASCII value.

Comments

This command allows the addition of a carriage return/line feed pair to the end of any string. For example, if a file was opened with the command **FOPEN 0 "filename" "wt"**, a string written with **FPUTS** could be terminated with a line feed with the statement **FPUTC 0 10.**

This command can be tested with the **IF SUCCESS** statement, returning "FALSE" if the data cannot be written to the file.

See also

FOPEN, FPUTS and FWRITE.

FPUTS

Writes a string to the file corresponding to the specified index.

FPUTS *index string*

Comments

FPUTS should only be used on strings terminating in one or more nulls; otherwise, use the **FWRITE** command.

This command can be tested with the **IF SUCCESS** statement, returning "FALSE" if the data cannot be written to the file.

See also

FOPEN, FPUTC and FWRITE.

FREAD

Reads a block of data from a file into a string variable and returns the actual number of bytes read in a third integer variable.

FREAD *index strvar length intvar*

strvar	String variable that will hold the results of the read.
length	The size in bytes of the block to be read (an integer value between 1 and 80).
intvar	Return value is the actual number of bytes read, which may be less than requested if an error occurs or end-of-file is reached.

Example

fread 0 S1 40 N3	Reads 40 bytes from the file indexed as 0 into S1 and puts the "return value" into N3.

Comments

The status of this command can be checked by comparing the number of bytes read with the number requested; alternately, you can test for the end-of-file condition.

To test for errors while reading a file, use the **IF SUCCESS** and **IF FAILURE** statements.

See also

FGETC, FGETS and FOPEN.

FSEEK

Repositions the file pointer for the file corresponding to the specified index.

FSEEK *index long integer*

long	Offset of the new position relative to the origin of the seek; the value can be positive or negative.

integer	The origin of the seek operation. Possible values are 0 (beginning of the file), 1 (current pointer position) or 2 (end-of-file).

Example

fseek 3 100 1	Positions the pointer within the file corresponding to index 3. The pointer will move forward 100 bytes from the current location.

Comments

Attempting to reposition the file pointer before the beginning of the file will result in an error.

When using this command with a TEXT file, seek with an offset of 0 relative to any of the origin values *or* seek with an offset from a previous **FTELL** command relative to the beginning of the file.

This command can be tested with the **IF SUCCESS** statement, returning "FALSE" if seek errors occur.

See also

FTELL, FOPEN and REWIND.

FSTRFMT

Similar to **STRFMT** and **FATSAY**, this command writes a formatted string to the file referenced by the provided index number.

FSTRFMT *index formatstr [param]...[param]*

Example

integer inum **fstrfmt 0 "ITEM no. %d" inum**	Writes the string to the file indexed as 0 with value substitution (as indicated by the format specifiers).

Comments

For more information on the format specifiers available for FSTRFMT, please refer to the definition for the **FATSAY** command.

This command can be tested with the **IF SUCCESS** statement, returning "FALSE" if the data cannot be written to the file.

See also

STRFMT and FATSAY.

FTELL

Returns the current file pointer position of the file corresponding to the specified file index into a long variable.

FTELL *index longvar*

Example

```
long loc
ftell 2 loc
```

Returns the pointer position of the file corresponding to index 2 into the long variable LOC.

Comments

The long variable contains the number of bytes past the beginning of the file.

If the file is opened in TEXT mode, carriage return/line feed translations may cause the physical offset to be incorrectly reported; however, FTELL can be used together with **FSEEK** to store and return to a particular file location.

See also

REWIND, FOPEN and FSEEK.

FTOA

Converts a float to an ASCII string and stores it in a string variable.

FTOA *float strvar*

See also

ATOF.

FWRITE

Writes a block of data to a file.

FWRITE *index string length*

string	A string or string variable that contains the block to be written.
length	The size in bytes (an integer value between 1 and 80) of the block to be written.

Example

```
string block1
fwrite 0 block1 30
```

Writes 30 characters from BLOCK1 to the output file specified by index 0.

Comments

This command can be tested with the **IF SUCCESS** statement, which returns "FALSE" if the string cannot be written.

See also

FOPEN, FSEEK, FREAD, FSTRFMT, FPUTC and FPUTS.

GE

Performs relational testing ("greater than or equal to") on two numeric values.

GE *number number intvar*

Comments

The integer variable will be initialized to 0 for "FALSE" and 1 for "TRUE".

The conditional form of this command is "IF/ELSEIF/WHILE GE NUMBER NUMBER", and the symbol ">=" acts as the operator form.

See also

LT, GT and GE.

GET

Receives and stores a text string or number up to 80 characters in length, entered by the user at the keyboard.

GET *strvar* | *numvar* [*length*]

length	A numeric value which determines the maximum number of characters to be accepted. If this optional parameter is omitted, the default value of 80 characters is used. A beep will sound if the user attempts to enter more than the specified number of characters.

Example

get s3 1	Stores a single keystroke entered by the user in the predefined string variable S3.

Comments

When responding to a GET command, the user must enter a carriage return to signal the completion of the input. The carriage return is not included in the text string stored in a string variable.

GET accepts only ASCII characters; function keys and non-displayable characters are not allowed.

The cursor position will remain at the location where the user pressed Enter or Esc.

GET can be exited with the Esc key (the statement **IF FAILURE** returns "TRUE").

For more information, see the **ATGET** command.

See also

ATGET, KEYGET, MATGET, MGET and RGET.

GETCUR

Returns the cursor's current row and column values into the first and second numeric variables, respectively.

GETCUR *intvar intvar*

Example

```
getcur rows cols
```

Places the current cursor row and column values in the numeric variables ROWS and COLS.

Comments

These values are also available directly through the system variables $ROW and $COL.

The LOCATE command can restore a cursor position saved with GETCUR.

See also

LOCATE, $ROW and $COL system variables.

GETDIR

Returns the current working directory path of the specified drive into a string variable.

GETDIR *disk strvar*

disk — An integer number designating the drive; 0 indicates the current drive, 1 specifies drive A, 2 for drive B and so on.

Example

```
string path
getdir 0 path
```

Places the working path and directory for the current drive into the string variable PATH.

Comments

You can test for possible drive access errors with the statements **IF SUCCESS** or **IF FAILURE**.

See also

FINDFIRST, FINDNEXT, DISKFREE and CHDIR.

GETENV

Returns the contents of an environment variable definition into a string variable.

GETENV *string strvar*

string	An environment variable currently defined through DOS.

Example

```
string env
getenv "pcplus" env
```

Places the current value for the environment variable PCPLUS into the string variable ENV.

Comments

GETENV returns a null string if the environment variable doesn't exist.

The environment variable is forced uppercase (even if supplied in lowercase).

See also

PUTENV.

GETFATTR

Returns the attributes of a specified file into a string variable.

GETFATTR *filespec strvar*

filespec	A fully-qualified single filename; wildcards are not allowed.

Example

getfattr "RBBS.LOG" attr

Places the current attributes for the file RBBS.LOG into the string variable ATTR.

Comments

Attributes can include "R" (Read-Only), "H" (Hidden), "S" (System), "A" (Archive), "D" (Directory) and "V" (Volume).

The statements **IF SUCCESS** or **IF FAILURE** can test for the successful return of the attribute information.

GETFDATE

Returns the date stamp of a specified file into a string variable.

GETFDATE *filespec strvar*

string

A fully-qualified single filename; wildcards are not allowed.

Example

getfdate "MLC.DAT" bdate

Places the date stamp (in the format xx-xx-xx) of the file MLC.DAT into the string variable BDATE.

GETFILE

Receives (or "downloads") a file from a remote computer using the indicated transfer protocol.

GETFILE *protocol* [*filespec*]

protocol

The following protocols do **not** require a filespec:

EXTPROTO1
EXTPROTO2
EXTPROTO3
YMODEM
YMODEMG
SEALINK
KERMIT
TELINK
MODEM7
ZMODEM
CISB

The following protocols do require a filespec:

1KXMODEM *filespec*
1KXMODEMG *filespec*
IMODEM *filespec*
XMODEM *filespec*
WXMODEM *filespec*
ASCII *filespec*
RASCII *filespec*

Examples

```
waitfor "Begin transfer!"
getfile zmodem
```

Uses GETFILE with ZMODEM.

```
waitfor "Kermit-32>"
message "Enter file to transfer"
get S1
transmit "SEND"
transmit S1
transmit "^M"
getfile kermit
```

Uses GETFILE with KERMIT.

Comments

Note that many of the protocols require either a valid DOS filespec or a variable containing a filespec. A path may be specified; if omitted, PROCOMM PLUS uses the default download directory specified in Setup. For the other protocols, a filespec should *not* be specified, since the remote system supplies the filename.

GETFILE can be tested with the **IF SUCCESS** and **IF FAILURE** statements.

Pre-existing files can pose a problem if you're using the ZMODEM transfer protocol. If a file you're receiving already exists and *crash recovery* is not enabled, the file will be skipped—however, the **IF SUCCESS** statement will still return a value of "TRUE". To avoid this, use the **ISFILE** command to determine if the file already exists on your system before the transfer—then delete it before using GETFILE. For more information on crash recovery, see "ZMODEM Protocol Options" in Chapter 8 of your USER MANUAL.

See the appendix "File Transfer Protocols" in your USER MANUAL for more information on each protocol. For the numeric values returned by a **FETCH PROTOCOL** statement, please refer to the definition for **FETCH** in this chapter.

See also

SENDFILE.

GETFSIZE

Returns the size of a specified file into a long variable.

GETFSIZE *filespec longvar*

filespec	A fully-qualified single filename; wildcards are not allowed.

Example

```
long size
getfsize "pcplus.dir" size
```

Places the size of the file PCPLUS.DIR (in bytes) into the long variable SIZE.

Comments

GETFSIZE can be tested with the **IF SUCCESS** and **IF FAILURE** statements.

GETFTIME

Returns the time stamp of a specified file into a string variable.

GETFTIME *filespec strvar*

filespec	A fully-qualified single filename; wildcards are not allowed.

Example

```
string ft
getftime "ROB.TXT" ft
```

Places the time stamp (in military 24-hour format) of the file ROB.TXT into the string variable FT.

GETVATTR

Returns the display attribute from the specified screen coordinates into an integer variable.

GETVATTR *row column intvar*

Example

```
integer color
getvattr 5 8 color
```

Places the attribute for the character at row 5, column 8 into the numeric variable COLOR.

See also

GETVCHAR and PUTVATTR.

GETVCHAR

Returns the decimal value of a character from the specified screen coordinates into an integer variable.

GETVCHAR *row column intvar*

Example

```
integer letter
getvchar 9 1 letter
```

Places the decimal value of the character at row 9, column 1 into the numeric variable LETTER.

See also

GETVATTR and PUTVCHAR.

GOSUB

Provides an unconditional branch to the specified procedure with return.

GOSUB *name* [WITH *param*[*param*]...[*param*]]

Comments

GOSUB is a synonym for the **CALL** command. It has been retained for backward compatibility with previous versions of PROCOMM PLUS—however, since GOSUB may be removed in future versions of ASPECT, it's recommended that you use **CALL** instead.

See also

CALL.

GOTO

Performs an unconditional branch to the specified label within the current procedure.

GOTO *name*

name

A label within the current procedure. Labels provide a point to which the GOTO command directs program control.

Example

```
if not waitfor
   goto error_exit
endif
.
.
.
error_exit:
   message "Abnormal termination"
   hangup
   quit
```

Comments

When PROCOMM PLUS encounters a GOTO command, it jumps to the location where the label is defined, resuming execution with the command immediately following the label.

A label can only be referenced within the procedure where it's defined (for non-local branching, use the **SETJMP** and **LONGJMP** commands). Labels can't be placed between **CASE** statements in a **SWITCH** command block, or between procedure blocks.

See also

SETJMP, LONGJMP and CALL.

GT

Performs relational testing ("greater than") on two numeric values.

GT *number number intvar*

Comments

The integer variable will be initialized to 0 for "FALSE" and any non-zero value for "TRUE".

The conditional form of this command is "IF/ELSEIF/WHILE GT NUMBER NUMBER", and the symbol ">" acts as the operator form.

See also

GT and LE.

HANGUP

Disconnects your computer from the telephone line by dropping your modem's Data Terminal Ready (DTR) line. The command then tests the Carrier Detect (CD) and, if the modem is still connected, sends the hang-up string defined in Setup.

HANGUP

Example

hangup	Disconnects the telephone line.

Comments

The default hang-up string (which should work for Hayes and compatible modems) is ~~~+++~~~ATH0^M. Use the **IF CONNECTED** statement to determine if the disconnect was successful.

See also

SET MODEM HANGUP.

HELP

Displays the PROCOMM PLUS Command Menu help screen, allowing access to On-line Help.

HELP

HOME

Moves the cursor to the first row and column on the screen.

HOME

See also

CURDN, CURLF, CURRT, CURUP and RCA.

HOOK

Executes an external program, passing the segment and offset of the current PROCOMM PLUS Setup structure as an argument.

HOOK *string*

Example

hook "FASTXFER"	Executes the external program FASTXFER.

Comments

HOOK allows you to execute external programs specifically written to utilize and/or change information on the current settings in PROCOMM PLUS, including the current COM port and baud rate.

HOOK can also utilize and/or change the values in the predefined variables (S0 to S9 and N0 to N9), allowing an information channel between HOOKed programs and currently-executing scripts.

See also

METAKEY, RUN and EXECUTE.

HOST

Places PROCOMM PLUS in Host mode.

HOST

Comments

By creating a script containing this command, you can automatically switch to Host mode when you start PROCOMM PLUS (or you can start PROCOMM PLUS in Host mode at a particular time).

You can test the HOST command for **SUCCESS** or **FAILURE**; **FAILURE** is set if a privileged user aborted Host mode or if the Esc key was pressed to abort Host.

Several system variables can provide information on the last caller that successfully logged on to Host; see the section titled "System Variables" at the end of this chapter.

IF

Executes the commands on the following line if the *condition* is true.

IF *condition*
.
.
.
[ELSEIF *condition*]
.
.
.
[ELSE]
.
.
.
ENDIF

ELSE	Commands on the lines following this command are executed when the condition tested by the IF command is "FALSE".
ELSEIF	Branches to another IF command block.
ENDIF	Concludes the IF command phrase.

A condition consists of a command word and its parameters or a numeric expression; you may also include the optional **NOT** and **ZERO** parameters. If a command or expression is evaluated as a non-zero value, it was satisfied (or "TRUE"). A zero value indicates that the conditional test was not satisfied (or "FALSE").

[NOT | ZERO] {expression | math | EOF index | NULL strvar}

NOT - May be employed with any condition. The effect of the NOT is to reverse the value of the condition. For example, when CONNECTED is "TRUE", then NOT CONNECTED is "FALSE". The conditions NOT SUCCESS and FAILURE are exactly the same.

ZERO - Tests whether a numeric variable has a value of zero. ZERO and **NOT** are functionally equivalent.

math - Can be one of the following: **ADD, AND, ANDL, COMP, DEC, DIV, EQ, GE, GT, INC, INIT, LE, LT, MOD, MUL, NEG, NEQ, NOT, OR, ORL, SHL, SHR, SUB, XOR** or **ZERO**. For most of these commands, the required parameters should be included except for the "result" variable; DEC, INC and INIT require **all** parameters in the condition. For more information on math commands, please refer to the Operator Appendix and their definitions elsewhere in this chapter.

condition - commonly-used conditions for the IF command include the following system variables and commands: **CONNECTED, COMDATA, FROMDDIR, EOF *index*, FAILURE, FOUND, HITKEY, MONO, NULL *string*, SUCCESS** and **WAITFOR.**

The *CONNECTED* condition is true if CD (Carrier Detect) is high. CD is normally high only when you're connected to a remote system. If your modem forces CD high (usually a dip switch setting) the CONNECTED condition will always be "TRUE".

The *EOF* condition tests for the end-of-file on the file corresponding to the provided index number See the definition for **EOF** for more information.

The *FAILURE* condition is considered "TRUE" if the last tested command was not successfully completed.

The *FOUND* condition is used to test the result of the last FIND command executed. It is considered "TRUE" if the "target" was found in the specified string variable. Please see the FIND command for further information.

The *HITKEY* condition is considered "TRUE" if a key has been pressed and remains in the input buffer.

If you execute a script from a Dialing Directory entry, the *FROMDDIR* condition will be "TRUE" anywhere within that script. The primary use for FROMDDIR is to allow a script to be executed either independently or "attached" to a Dialing Directory entry.

The *MONO* condition is evaluated as "TRUE" if a monochrome monitor is connected.

The *NULL* condition tests a string for no contents. See the definition for **NULL** for more information.

The *SUCCESS* condition is evaluated as "TRUE" if the last testable command was successfully completed.

The *WAITFOR* condition checks the result of the last **WAITFOR** command. If the "target" specified in the **WAITFOR** command is received, the WAITFOR condition is "TRUE"; if the **WAITFOR** command times out before receiving the "target", the condition is "FALSE". For more information, please refer to the definition of the **WAITFOR** command.

Examples

```
run "someprog"
if success
   .
   .
   .
endif
```

Executes if the program SOMEPROG ran successfully.

```
loop:
while not hitkey
endwhile
```

Execution loops endlessly until a key is pressed.

```
if not fromddir
   dial "5"
endif
```

Does not execute if the script file was started via a Dialing Directory link.

```
init n1 10
while (n1!=0)
   .
   .
   .
   --n1
endwhile
```

This segment will loop until N1 = 0.

```
if mono
   clear 7
endif
```

Executes the **CLEAR** command if a monochrome monitor is connected.

```
if total > 25
   call printdata
endif
```

Branches to a subroutine if the value of TOTAL is greater than or equal to 25.

Comments

Each IF command must end with an **ENDIF** command and can optionally be used with an **ELSE** or **ELSEIF** command. The command following the **ELSE** is executed if the specified condition evaluates as "FALSE".

There is no practical limit on the number of nested IF commands within a script.

See also

ELSE, ELSEIF, ENDIF and SWITCH.

$IFDEF

Allows conditional compilation by testing for the existence of a DEFINEd macro name. For more information, see the **DEFINE** command.

$IFDEF

INC

Increments a numeric variable by one.

INC *numvar*

Comments

It is possible to increment a number beyond the range of valid integers; PROCOMM PLUS does no error checking on this value.

The operator form of "INC N0" is "N0++".

See also

DEC and ADD.

INCLUDE

Merges commands from ASPECT source files during compilation.

INCLUDE *"filespec"*

Example

include "color.h"	Merges the commands found in the file COLOR.H with the current script. Note that this must be a quoted string.

Comments

Each INCLUDE may contain further INCLUDE commands. You can nest INCLUDE commands up to 10 levels deep.

INCLUDE commands can occur within or outside of a procedure block.

INCLUDE can be used to set up source "header" files—much like the "C" programming language—which contain a set of **DEFINE** commands; you can easily INCLUDE these header files in different ASPECT scripts without having to duplicate the work.

Another use of INCLUDE is the creation of a "library"—procedures commonly found in many of your scripts. Since **ASPCOMP** will not generate output for procedures that aren't called, your library can be as large as you like without adding to the size of the compiled .ASX script.

The filespec **must** be a quoted string, but it does not require the .ASP extension.

See also

DEFINE.

INIT

Initializes a numeric variable.

INIT *numvar number*

Examples

init N0 330 **init N1 N0**	Initializes N0 to 330 and then also initializes N1 to 330. The *operator* form of the first statement would be N0=330.

Comments

It is possible to initialize a variable outside of the valid range allowed by the numeric variable; ASPECT does no error checking on this value. Values outside of the valid range will be truncated to fit the variable limits.

The operator form of the statement "INIT N0 330" is "N0=330".

See also

ASSIGN and STRCPY.

INPORT

Reads data from the specified I/O port.

INPORT *port intvar*

Examples

inport 0x3f8 char	Reads a byte from the port with address 0X3F8 into the integer variable CHAR.

Comments

The first integer identifies the port address (in decimal or hex format). The value is treated as unsigned, ranging from 0 to 65535. Port values greater than 32767 must be entered as hexadecimal.

See also

OUTPORT.

INSCHAR

Inserts a character at the current cursor position. Any text from the current cursor to the end of the line will be shifted one column to the right.

INSCHAR

See also

DELCHAR, DELLINE and INSLINE.

INSLINE

Inserts a new line at the current cursor position. Any text on the current line will be scrolled up to the previous line.

INSLINE

See also

DELCHAR, DELLINE and INSCHAR.

INTEGER

Defines a global or local integer variable.

INTEGER *name[=expression][,name[=expression]]...*

[=expression]	An initializer expression. For numeric types, it can specify operators, constants or previously-declared numeric variables.

Examples

integer item	Defines the integer global variable ITEM.

Comments

For more information on using *global* and *local* variables, please refer to the definition for **FLOAT**.

See also

INTPARM, LONG, FLOAT and STRING.

INTPARM

Defines a integer parameter variable.

INTPARM *name[,name]...[,name]*

Examples

intparm count	Defines the integer parameter COUNT.

Comments

Any procedure—except MAIN—can be defined with up to 10 *parameter variables*. Parameter variables are similar to local variables in that they may only be referenced within the procedure in which they're defined; unlike local variables, though, the value of parameters are automatically initialized when the procedure is called.

For more information on using parameter variables, please refer to the definition for **FLOATPARM**.

See also

LONGPARM, FLOATPARM and STRPARM.

ISFILE

Determines if a file exists in the current (or specified) directory.

ISFILE *filespec*

filespec — Any valid DOS path and/or filename, including extension.

Examples

```
isfile "MARK.DOC"
if success
   message "Doc file exists"
else
   message "Doc file not found"
endif
message "Enter filename"
get S0
isfile S0
if not success
   message "File does not exist"
endif
```

Variable S0 stores the filename.

Comments

You can include a path specification with the filespec. Use the IF SUCCESS and IF FAILURE statements to test the results of the ISFILE command.

ITOA

Converts an integer to an ASCII string and stores it in a string variable.

ITOA *integer strvar*

See also

FTOA, LTOA, ATOI and STRFMT.

KERMSERVE

Issues a KERMIT server command.

KERMSERVE {SENDFILE *filespec* | GETFILE *filespec* | FINISH | LOGOUT}

SENDFILE filespec	Sends a file to a remote system. *filespec* is any valid DOS filename, including the extension.
GETFILE filespec	Receives a file from a remote system. *filespec* is any filename that's valid on the remote system.
FINISH	Logs out of KERMIT server mode.
LOGOUT	Logs out of KERMIT server mode *and* logs off Host.

Example

```
message "File to send?"
get sfil
kermserve sendfile sfil
kermserve finish
```

Stores the filename in variable SFIL, transfers the file and issues the FINISH SERVER command.

Comments

This command phrase will only work if the host is in the KERMIT server mode.

KERMSERVE can be tested with the **IF SUCCESS** and **IF FAILURE** statements.

KEY2ASCII

Converts an integer to the ASCII character it represents and places the result into a string variable.

KEY2ASCII *integer strvar*

Example

```
string charcd
integer numval
key2ascii numval charcd
```

Reads the value of the numeric integer NUMVAL and places the corresponding ASCII character in the string variable CHARCD.

Comments

If the integer value is greater than 255, it's converted to a 4-digit hexadecimal string.

See also

TERMKEY, SET KEYS and KEYGET.

KEYGET

Receives a key pressed by the user and (if requested) stores it in the specified integer variable.

KEYGET [*intvar*]

Comments

This command lets you get a single keystroke without requiring that the user terminate a series of keystrokes by pressing Enter (which is necessary with GET). The value placed in the numeric variable depends upon the key pressed.

To place the ASCII character corresponding to a key value into a string variable, use the **KEY2ASCII** command.

The Alt-Z key sequence will always call On-line Help, and will not be returned as a key code; it will remain in effect until the user presses a key other than Alt-Z.

See also

KEY2ASCII, GET, MATGET, TERMKEY, SET KEYS, MGET and RGET.

KFLUSH

Clears accumulated keystrokes from the keyboard buffer. Any unprocessed keystrokes that have been entered will be lost.

KFLUSH

See also

KEYGET, HITKEY system variable and RFLUSH.

KLOAD

Loads a different Keyboard Map file.

KLOAD *filespec*

Comments

The .KBD file extension is optional.

If the directory does not exist (either in the current directory or in the directory specified with the "SET PCPLUS=" environment variable), KLOAD will create an empty directory with that name.

LE

Performs relational testing ("less than or equal to") on two numeric values.

LE *number number intvar*

Example

```
integer price, subtotal
price=117
subtotal=15
le price subtotal n1
```

Tests the the values PRICE and SUBTOTAL, placing the result in the predefined numeric variable N1. Since 117 is **not** less than or equal to 15, N1 will be initialized to 0.

Comments

The integer variable will be initialized to 0 for "FALSE" and 1 for "TRUE".

The conditional form of this command is "IF/ELSEIF/WHILE LE NUMBER NUMBER", and the symbol "<=" acts as the operator form.

See also

LT, GT and GE.

LINEFEED

Moves the cursor down one line. If the current line is the last line of the screen, this command may scroll the screen (depending on the current emulation and the **SET SCROLL** option).

LINEFEED

See also

CURDN.

LOCATE

Positions the screen cursor to the location specified by *row* and *column*.

LOCATE *row column*

Example

```
clear
locate 10 20
message "Enter choice:"
locate 10 44
get S8
```

Clears the screen and locates the cursor at row 10, column 20.

Displays the message.

Positions the cursor at row 10, column 44 (at the right of the prompt) for user input.

Comments

Characters received from the remote system are displayed at the current cursor position.

Note that some commands (like **BOX**) write directly to the screen (without affecting the current cursor location); other commands (like **TERMWRT**) update the cursor location when executed.

See also

GETCUR, $ROW and $COL system variables.

LOG

Controls session logging during script file execution. The log file is a continuous record of all characters received and transmitted. Note that a SET DISPLAY OFF command in a script will disable the log file.

LOG {OPEN [*filespec*] | CLOSE | SUSPEND | RESUME}

OPEN	Opens the log file and begins logging data. This parameter can be tested for successful completion with IF SUCCESS | FAILURE.
[filespec]	The optional filename lets you specify the log filename. If this parameter is not included in the command, PROCOMM PLUS uses the default filename specified using the Setup facility (see "File/Path Options" in Chapter 8 of your USER MANUAL).
CLOSE	Turns off file logging and closes the log file.
SUSPEND	Temporarily halts logging of text without closing the log file.
RESUME	Continues logging after a SUSPEND command.

Examples

log open	Begins logging to default log file.
log open "newlog.txt"	Begins logging to file NEWLOG.TXT.

```
LOG SUSPEND
.
.
.
LOG RESUME
```

Comments

If the log file already exists, the new data is added to the end of the existing file.

See also

SET DISPLAY and SNAPSHOT.

LONG

Defines a global or local long variable.

LONG *name*[=*expression*][,*name*[=*expression*]]...

[=expression]	An initializer expression. It can specify operators, constants or previously-declared numeric variables.

Examples

long location	Defines the long variable LOCATION.

Comments

For more information on using *global* and *local* variables, please refer to the definition for FLOAT.

See also

LONGPARM, INTEGER, FLOAT and STRING.

LONGJMP

Returns to a location within a script previously marked with a SETJMP command.

LONGJMP *index integer*

index	One of three integer index values (0,1 or 2); the value identifies the SETJMP location for this LONGJMP.
integer	An integer value to return to the SETJMP command; this allows further processing by statements following the SETJMP.

Example

longjmp 0 2	Returns to the SETJMP location referenced by the index number 0. The SETJMP command's second parameter will receive the integer value 2.

Comments

The marked location is the command following the **SETJMP**. Up to three **SETJMP** locations can be active at once.

The marked location remains active until a return from the function where it was set—or until another routine uses a **SETJMP** with the same index.

See also

GOTO, RETURN and SETJMP.

LONGPARM

Defines a long parameter variable.

LONGPARM *name*[,*name*]...[,*name*]

Examples

longparm memory	Defines the long parameter MEMORY.

Comments

Any procedure—except MAIN—can be defined with up to 10 *parameter variables*. Parameter variables are similar to local variables in that they may only be referenced within the procedure in which they're defined; unlike local variables, though, the value of parameters are automatically initialized when the procedure is called.

For more information on using parameter variables, please refer to the definition for **FLOATPARM.**

See also

INTPARM, FLOATPARM and STRPARM.

LOOPFOR

Increments or decrements the control variable in a **FOR** command block. If another iteration should be performed, execution continues with the command following the **FOR** command. See the **FOR** command for further information.

LOOPFOR

LOOPWHILE

Increments or decrements the control variable in a **WHILE** command block. If another iteration should be performed, execution continues with the command following the **WHILE** command. See the **WHILE** command for further information.

LOOPWHILE

LT

Performs relational testing ("less than") on two numeric values.

LT *number number intvar*

Example

```
integer price,subtotal
price=10
subtotal=15
lt price subtotal n1
```

Tests the the values PRICE and SUBTOTAL, placing the result in the predefined numeric variable N1. Since 10 is less than 15, N1 will be initialized to a non-zero value.

Comments

The integer variable will be initialized to 0 for "FALSE" and any non-zero value for "TRUE".

The conditional form of this command is "IF/ELSEIF/WHILE LT NUMBER NUMBER", and the symbol "<" acts as the operator form.

See also

GT and LE.

LTOA

Converts a long to an ASCII string and stores it in a string variable.

LTOA *long strvar*

See also

ATOL.

MATGET

Relocates the cursor to a specified row and column, gets a text string or number entered by the user at the keyboard and stores it in the specified variable. The input is masked by echoing asterisks (*) on the screen.

MATGET *row column attribute length strvar | numvar*

length	A numeric value between 1 and 80 that determines the maximum number of characters that will be accepted. A beep will sound if the user attempts to enter more than the specified number of characters.
*strvar	numvar*	The variable where the text string or number is stored.

Comments

This command is similar to **ATGET**; however, MATGET masks the input data.

The cursor position will remain at the location where the user pressed Enter or Esc.

MATGET can be exited with the Esc key (**FAILURE** is set "TRUE").

See also

ATGET, GET, KEYGET, and MGET.

MDIAL

Calls the specified telephone number ("manual dial").

MDIAL *"[ldcode]number[ldcode]" | strvar [message]*

number	The telephone number of the computer you want to call.
ldcode	A dialing code (see "Adding or Changing a Dialing Code" in Chapter 6 of your USER MANUAL).
strvar	A variable, which you can set to a valid telephone number and use in place of a quoted string.
[message]	An optional string displayed in the Directory window while PROCOMM PLUS is dialing the number. This string typically contains the name of the BBS or on-line service you're calling.

Examples

mdial "555-1234"	Dials the telephone number 555-1234.
mdial "A555-1234"	Dials dialing code A and then the telephone number 555-1234.
if not fromddir ** mdial "555-1234" "WORK"** **endif**	Calls the telephone number 555-1234 (unless the script file is executed via the Dialing Directory). The message "WORK" appears in the directory window while the number is dialed.

Comments

Use the **IF NOT FROMDDIR** statement to prevent the MDIAL command from redialing if the script file being executed is linked to a Dialing Directory entry (see the last example above).

MDIAL accepts only a single telephone number; for multiple Dialing Directory entry numbers, use the **DIAL** command.

See also

DIAL and REDIAL.

MEMFREE

Returns the amount of free memory (or RAM) available for running other programs into a long variable.

MEMFREE *longvar*

Example

```
long mem
memfree mem
fatsay 10 20 30 "%ld bytes" mem
```

Returns the current free RAM available into the long variable MEMORY.

See also

DISKFREE.

MEMPEEK

Returns the value of a single byte at the specified memory address.

MEMPEEK *segment offset intvar*

Example

```
mempeek 0 0x0417 byte
```

Returns the value of the byte at segment 0, offset 0x0417 into the integer variable BYTE.

Comments

Both segment and offset can be entered in decimal or hex form; they're treated as unsigned values.

See also

MEMPOKE and MEMREAD.

MEMPOKE

Sets the value of a single byte at the specified memory address.

MEMPOKE *segment offset character*

Example

mempoke 0x40 0x1E byte — Sets the value of the byte at segment 0x40, offset 0x1E to the value specified in BYTE.

Comments

Both segment and offset can be entered in decimal or hex form; they're treated as unsigned values.

See also

MEMPEEK and MEMWRITE.

MEMREAD

Returns the values beginning at the specified memory address into a string variable.

MEMREAD *segment offset strvar length*

Example

memread 0xF000 0xFFF5 adate 8 — Returns the value of the first 8 bytes beginning at segment 0xF000, offset 0xFFF5 into the string variable ADATE.

Comments

Both segment and offset can be entered in decimal or hex form; they're treated as unsigned values.

The contents of the string can be examined with the **STRPEEK** command.

See also

MEMPEEK, MEMWRITE and STRPEEK.

MEMWRITE

Sets the values beginning at the specified memory address from the contents of a string.

MEMWRITE *segment offset string length*

Example

memwrite 0x40 0x1E outbytes 16 Sets the value of the first 16 bytes beginning at segment 0x40, offset 0x1E from the string OUTBYTES.

Comments

Both segment and offset can be entered in decimal or hex form; they're treated as unsigned values.

The contents of a string variable can be set using **STRSET** and **STRPOKE**.

See also

MEMREAD.

MESSAGE

Displays text on the local screen, **without** sending it to the remote. The message is displayed at the current cursor position in the current colors. MESSAGE appends a CR/LF to each text string (unless you **SET MSG_CRLF OFF**).

MESSAGE *string*

string A series of characters to display. The string may contain control characters (such as CR and LF) by using the translation conventions described under "Translating Control Codes" in your USER MANUAL.

Examples

```
message "+--------------------------+"
message "| Enter your choice: |"
MESSAGE "+--------------------------+"
locate 2 20
get S0 1
assign S9 "This is line one^M^JThis is the second"
message S9
```

Comments

^M sends a carriage return, while ^J is a line feed.

Use the MESSAGE command for prompts, informational messages and building menus.

See also

SET MSG_CRLF, ATSAY, FATSAY and LOCATE.

METAKEY

Executes the specified Meta key.

METAKEY *integer*

integer	Any integer value from 0 to 9.

Examples

`metakey 5`	Sends Meta key assigned to Alt-5.
`init n8 2` `metakey n8`	Stores a Meta key number in N8 and then sends the Meta key sequence assigned to Alt-2.

Comments

Use the MLOAD command to load individual keyboard Meta key files.

METAKEY can only be used with Meta key entries that don't execute other scripts.

See also

MLOAD.

MGET

Receives a text string or number entered by the user at the keyboard and stores it in the specified variable. MGET prevents anyone from reading the entry on the screen by "masking" the entry with asterisks (*).

MGET *strvar | numvar* [*length*]

*strvar	numvar*	The variable into which the string or number received from the user is placed.
length	A numeric value which determines the maximum number of characters that will be accepted. If this optional parameter is omitted, the default value of 80 characters is used. A beep will sound if the user attempts to enter more than the specified number of characters.

Example

```
message "Enter the password"
mget S9 8
find S9 "secret"
IF NOT FOUND
   message "You're not an authorized user!"
   quit
endif
```

Comments

MGET is useful for security-related information; for example, user passwords.

The cursor position will remain at the location where the user pressed Enter or Esc.

MGET can be exited with the Esc key (the statement **IF FAILURE** returns "TRUE").

See also

ATGET, GET, KEYGET, and MATGET.

MKDIR

Creates a new directory using a path and/or directory name you provide.

MKDIR *filespec*

Example

mkdir "temp" — Creates the directory TEMP (within the current working directory).

Comments

If a path is not provided, PROCOMM PLUS creates the directory within the current working directory.

The result of a MKDIR command can be tested with the **IF SUCCESS** and **IF FAILURE** statements.

See also

RMDIR, CHDIR and GETDIR.

MLOAD

Loads the specified keyboard Meta key file.

MLOAD *filespec*

filespec — Any valid DOS filename.

Example

mload "SYSTEM1.KEY" — Loads a new Meta key file.

Comments

If the file doesn't exist, it will be created. The *.KEY* extension is not required.

This command can be tested with the **IF SUCCESS** statement, returning "FALSE" if the specified Meta key file can't be read (or created) in either the current directory or in the directory specified with the "SET PCPLUS=" command in the AUTOEXEC.BAT file.

MOD

Returns the remainder after division (called the "modulus") into a numeric variable.

MOD *number number numvar*

Example

```
integer leftover
mod 10 7 leftover
```

Divides 10 by 7 and places the remainder in the numeric variable LEFTOVER. The operator form would be *LEFTOVER=10%7*.

Comments

For floats, integers and longs, the remainder has the same sign as the number being divided (the dividend); its absolute value is always less than the absolute value of the divisor.

See also

DIV, CEIL and FLOOR.

MSPAUSE

Pauses script execution for the specified number of milliseconds.

MSPAUSE *integer*

Example

```
mspause 100
```

Pauses the execution of the script for 100 milliseconds.

Comments

Unlike the **PAUSE** command, MSPAUSE can't be aborted with the Esc key.

See also

PAUSE.

MUL

Multiplies the first two numbers and stores the result in the last numeric variable.

MUL *number number numvar*

Comments

It is possible to multiply two numbers and have the result outside the range of the specified numeric variable; PROCOMM PLUS does no error checking on this value.

The statement "MUL N0 N1 N2" has the equivalent operator form "N2=N0*N1".

See also

DIV.

NEG

Negates the value of the specified number and places the result in a numeric variable.

NEG *number numvar*

Comments

The *command* form "NEG N0 N1" is equivalent to the *operator* form "N1=-N0".

See also

SUB and COMP.

NEQ

Tests two numeric values for non-equality.

NEQ *number number intvar*

Example

```
integer first, second
neq first second n1
```

Tests the values FIRST and SECOND, placing the result in the predefined numeric variable N1.

Comments

The integer variable will be initialized to 0 for "FALSE" and any non-zero value for "TRUE".

The conditional form of this command is "IF/ELSEIF/WHILE NEQ NUMBER NUMBER". The statement "NEQ N0 N1 N2" has the equivalent operator form "N2=N0!=N1".

See also

EQ.

NORMON

Turns on the normal attribute for Terminal mode display.

NORMON

See also

REVON, ULINEON, DIMON and BLANKON.

NOT

Performs a logical NOT operation on a number. The numeric variable is assigned 1 if the number is 0, and 0 if the number is non-zero.

NOT *number intvar*

Comments

NOT and **ZERO** function identically.

The *command* form "NOT N0 N2" is equivalent to the *operator* form "N2=!N0".

See also

ZERO.

NULL

Tests a string variable for the null condition (no contents).

NULL *string intvar*

Example

```
null s1 result
```

Tests the contents of the predefined string variable S1 for null. The result is placed in the integer variable RESULT.

```
while null s1
  atget 10 20 15 10 s1
endwhile
```

Forces a response to the ATGET statement.

Comments

The integer variable will be initialized to 0 (meaning "FALSE") or 1 (meaning "TRUE", or no contents) after the test.

The conditional form of this command is "IF NULL STRVAR".

OR

This command performs a bitwise comparison of two numbers and places the result in the specified numeric variable. For each two bits compared, the resulting variable is assigned the value 1 or 0 in the same corresponding bit position. A 1 is assigned if either or both bits are 1, while a 0 is assigned if both of the bits are 0.

OR *number number numvar*

Comments

OR cannot be used with floating point numbers.

The *command* form "OR N0 N1 N2" is equivalent to the *operator* form "N2=N0|N1".

See also

XOR and AND.

ORL

Performs a logical OR operation on two numbers and places the result in the specified numeric variable. The result is 1 if either or both numbers are non-zero, while the result is 0 if both of the numbers are zero.

ORL *number number intvar*

Comments

ORL cannot be used with floating point numbers.

The *command* form "ORL N0 N1 N2" is equivalent to the *operator* form "N2=N0||N1".

See also

ANDL.

OUTPORT

Writes data to the specified I/O port.

OUTPORT *port character*

Examples

```
integer char=65
outport 0x3f8 char
```

Writes a byte to the port with address 0x3F8 from the integer variable CHAR.

Comments

Port identifies the port address (in decimal or hex format); the value is treated as unsigned.

IMPORTANT: OUTPORT SHOULD ONLY BE USED IF YOU'RE EXPERIENCED WITH MANIPULATING THE HARDWARE SIGNALS AND PORTS ON YOUR SYSTEM. IMPROPER USE OF THE OUTPORT COMMAND COULD DAMAGE YOUR HARDWARE!

See also

INPORT.

PARMREST

Restores the PROCOMM PLUS settings found in PCPLUS.PRM.

PARMREST

See also

PARMSAVE, SET and FETCH.

PARMSAVE

Saves the current PROCOMM PLUS settings to PCPLUS.PRM.

PARMSAVE

See also

PARMREST, SET and FETCH.

PAUSE

Halts script-file execution for the specified number of seconds. Characters received during a pause are not displayed until after the pause has completed.

PAUSE *integer*

integer	A positive integer value indicating the number of seconds to pause execution.

Examples

```
transmit "Kermit send file.ext"
pause 3
kermit receive
```

Begins a transfer and provides a pause to allow the remote system to start.

Comments

Unlike the **MSPAUSE** command, PAUSE can be aborted with the Esc key.

This command can be tested with the **IF SUCCESS** and **IF FAILURE** statements; FAILURE is set if PAUSE was terminated by the user pressing the Esc key.

See also

SUSPEND UNTIL and MSPAUSE.

PRINTER

Controls print logging. PROCOMM PLUS writes the log to the DOS print device specified in Setup General Options.

PRINTER ON | OFF

ON	Begins logging the session to the printer.
OFF	Ends logging the session to the printer.

Example

```
printer on
clear
.
.
.
printer off
```

Begins printing, executes other commands and then ends printing.

Comments

You can use the DOS MODE command to redirect printer output; alternately, change the name of your print device in Setup (or with the **SET PRTNAME** statement).

Instead of using this command, note that you can save time by using the LOG command and printing the data later.

See also

LOG and SET PRTNAME.

PROC

Marks the beginning of a procedure block.

PROC *name*

name — A unique name or "MAIN" (every ASPECT program begins execution at a procedure called "MAIN").

Example

Declares the procedure MAIN, manually dials a number, executes other commands and closes the procedure.

```
proc main
  clear
  mdial "345-6789" "WEEKLY"
.
.
.
endproc
```

Comments

Each procedure **must** be given a unique name, and every ASPECT script **must** have a procedure called "MAIN" (which indicates the starting point of script execution).

Most ASPECT commands occur within the body of a procedure—exceptions include **DEFINE, UNDEF, INCLUDE, INTEGER, LONG, FLOAT,** and **STRING.**

Any procedure except MAIN may declare a parameter list; the parameters are variables of any type, and are local to the procedure. The parameters are initialized with values passed via the **CALL** which invoked the procedure.

See also

CALL, INTPARM, STRPARM, LONGPARM, RETURN, SETJMP, LONGJMP, FLOATPARM and ENDPROC.

PUSHBACK

Moves the last character read from the receive data buffer back to the beginning of the buffer; in other words, the last character read from the buffer becomes the next character read from the buffer.

PUSHBACK

Comments

PUSHBACK can be called multiple times to push back any number of characters previously read from the buffer.

See also

COMGETC, COMGETCD, COMPUTC and SET RXDATA.

PUTENV

Adds or changes an environment variable definition.

PUTENV *string*

Example

putenv "pcplus=d:\pc"	Sets the PCPLUS environment variable to the \PC directory on drive D.

Comments

Only 1 variable can be added or changed at a time.

The variable is active until the command is called again (at which time the previous variable definition is purged). Any addition or change will be lost after you exit PROCOMM PLUS.

The word SET is not required in the string, and the environment variable is forced uppercase (even if supplied in lowercase).

This command can be tested with the **IF SUCCESS** and **IF FAILURE** statements to determine if the environment modification was successful.

See also

GETENV.

PUTVATTR

Updates the local screen with a display attribute at the specified row and column.

PUTVATTR *row column attribute*

Example

```
putvattr 4 20 7
```

Sets the attribute at row 4, column 20 to 7.

See also

GETVATTR and PUTVCHAR.

PUTVCHAR

Updates the local screen with a character at the specified row and column.

PUTVCHAR *row column character*

Example

```
putvchar row1 col1 205
```

Displays the character (decimal 205) at the location specified by the integer variables ROW1 and COL1.

See also

GETVCHAR and PUTVATTR.

QUIT

Terminates the executing script file, disconnects, and then exits PROCOMM PLUS.

QUIT

Example

```
transmit "disconnecting"
quit
```

Quits PROCOMM PLUS after sending a message and disconnecting.

Comments

Use QUIT only when you wish to terminate both the script **and** PROCOMM PLUS. To exit without disconnecting, use the **BYE** command.

See also

BYE, CONNECT, EXIT and TERMINAL.

RCA

Interprets the next two received characters as a specific row and column where the cursor should be moved.

RCA

Comments

The row and column values are based from the value 32; for example, the values 35 and 40 refer to row 3, column 8 (subtract 32 from the values received).

See also

CURDN, CURLF, CURRT and HOME.

RDWRITE

Empties the Terminal mode Redisplay buffer into the specified file.

RDWRITE *filespec*

Comments

Data is appended to the end of a file if it already exists.

This command can be tested with the **IF SUCCESS** statement, returning "TRUE" if the data was written to the file.

See also

RDFLUSH and SET REDISPLAY.

REDIAL

Redials numbers in the dialing queue.

REDIAL [*strvar*]

strvar	A variable in which PROCOMM PLUS returns the number of the dialing directory entry with which a connection has been made.

Comments

If you use REDIAL with a queue, it will attempt to redial beginning with the first number in the queue.

See also

MDIAL and DIAL.

RENAME

Renames an existing file.

RENAME *filespec filespec*

Example

rename "PC.LOG" "bak.log"	Renames PC.LOG to BAK.LOG in the current directory.
rename "C:\tmp" "C:\hw\old"	Renames the file TMP in the root directory of drive C to OLD in the HW directory (effectively moving it).

Comments

If no path is supplied, RENAME searches the current directory for the source file.

Files can be renamed to a different directory on the same drive—a functional equivalent to moving them from one directory to another.

The result of a RENAME command can be tested with the **IF SUCCESS** or **IF FAILURE** statements.

See also

DELETE.

RETURN

Exits the current procedure and resumes processing at the statement following a CALL command.

RETURN

Comments

A RETURN statement placed in the "MAIN" procedure terminates the script.

The ENDPROC command includes an implied RETURN.

Any SETJMP commands invoked in the current procedure will no longer be active upon a RETURN from that procedure.

See also

PROC, ENDPROC, SETJMP, LONGJMP and CALL.

REVON

Turns on the reverse attribute for Terminal mode display.

REVON

See also

BLINKON, BOLDON, ULINEON and NORMON.

REWIND

Repositions the file pointer corresponding to the specified index back to the beginning of the file.

REWIND *index*

Example

rewind 0	Rewinds the pointer for the file indexed as 0. All end-of-file and error flags are cleared.

See also

FSEEK, FOPEN and FCLEAR.

RFLUSH

Clears the receive data buffer.

RFLUSH

Comments

Any characters that have been received but not processed or displayed will be lost when this command is issued. It's generally used to clear the receive data buffer in preparation for some task.

See also

KFLUSH and RDFLUSH.

RGET

Receives and stores text strings sent by a remote system.

RGET *strvar* [*length* [*integer*]]

strvar	The variable where the string is stored. The carriage return is not stored as a part of the user's input.
length	An integer value ranging from 0 to 80, specifies the maximum number of characters to receive before continuing processing. If this parameter is not included, the maximum of 80 characters is used.
integer	This integer determines the maximum number of seconds to wait for the string to be received from the remote system before timing out. If this parameter is not included, the default of 30 seconds is used.

Example

```
transmit "ATS0=1^M"
while not connected
endwhile
transmit "ENTER PASSWORD:"
rget S9 8 45
if not success
      transmit "Goodbye!"
      hangup
endif
```

Waits a maximum of 45 seconds to receive a text string of 8 characters and then tests the outcome.

Disconnects if the characters are not received.

Comments

The RGET command completes when a carriage return is received, when the specified number of characters are received or when the specified/default time expires. Execution continues with the next statement in the script.

This command can be tested with the **IF FAILURE** command, which returns "TRUE" when RGET times out.

RGET can be exited with the Esc key (the statement **IF FAILURE** is returns "TRUE").

To RGET a result message from a modem, you need to do two RGETs: one to strip the carriage return sent by the modem before its message, and one to capture the modem message itself (since RGET ends as soon as it encounters a carriage return).

See also

GET, COMGETC and COMGETCD.

RMDIR

Removes an existing directory using a specified path.

RMDIR *filespec*

Comments

If no path is supplied, RMDIR removes a subdirectory in the current working directory.

The result of a RMDIR command can be tested with the **IF SUCCESS** or **IF FAILURE** statements.

Only empty directories can be removed.

See also

CHDIR, MKDIR and GETDIR.

RSTRCMP

Compares the contents of two strings up to the specified length.

RSTRCMP *string string* [*length*]

Example

rstrcmp pass input 30	Compares the first 30 characters of the two strings PASS and INPUT.

Comments

The strings can contain *raw* data, displayable or not.

The result of this command can be tested with the **IF SUCCESS** or **IF FAILURE** statements.

See also

STRCMP.

RUN

Executes any external program from within a PROCOMM PLUS script file, except for DOS internal commands and batch files.

RUN *string* [WAIT | NOCLEAR]

string	Any executable program or command, (except an internal DOS command or a batch file) as it would appear on the DOS command line (optionally including a DOS path).
WAIT	After the program is completed, PROCOMM PLUS waits for a key press before returning to the script file.

NOCLEAR	Prevents PROCOMM PLUS from saving and clearing the screen before executing the command (and restoring it after execution is complete).

Examples

run "filesort"	Executes a program with the start-up command FILESORT.
string prg	Executes the program and then tests for a successful completion.

```
assign prg "filesort"
run prg
if failure
   message "filesort returned error"
else
   message"filesort executed successfully"
endif
```

Comments

If the path to the executable command is not specified with it, the program or the DOS command must be either in the current directory or in the directory specified in your DOS path.

Arguments can be passed to the external program by separating the program name and the arguments from each other with spaces.

This command can be tested with the IF SUCCESS statement, returning "FALSE" if the program could not be executed or exited with any return code other than 0.

Be sure that you have enough memory to execute the specified command or program and PROCOMM PLUS at the same time. If you use the RUN command to execute a program or command that requires user input, make sure that the user running the script file is aware of this, since processing will halt until the required input is provided.

For more information, see "Problems When Running an External Program" in your USER MANUAL and Chapter 4 of this manual.

See also

METAKEY, DOS and SHELL.

SCROLL

Scrolls an area of the screen up by the specified number of lines. This is similar to the BIOS scroll function.

SCROLL *integer row column row column attribute*

integer	An integer value ranging from 0 to the number of screen rows which determines the number of lines to scroll up within the defined area. If the value is 0, then the entire scroll area is blanked.
row column *row column*	Defines the scroll area. Since no error checking is performed, make sure that the first row and column pair are less than or equal to the second pair.

Examples

scroll 10 0 0 24 10 79	Scrolls the area from row 0 column 0 (upper-left corner) to row 24 column 10 (lower-right corner) up 10 lines and clears the area to a red background with a white foreground.
scroll 24 0 0 23 79 7	Clears the screen above the Status line to black on a white background.

See also

BOX and DSCROLL.

SENDFILE

Sends (or "uploads") a file to a remote computer using the indicated transfer protocol.

SENDFILE *protocol filespec*

protocol

The following protocols do **not** require a filespec:

EXTPROTO1
EXTPROTO2
EXTPROTO3
CISB

The following protocols require a filespec:

KERMIT *filespec*
XMODEM *filespec*
ZMODEM *filespec*
WXMODEM *filespec*
1KXMODEM *filespec*
1KXMODEMG *filespec*
IMODEM *filespec*
TELINK *filespec*
MODEM7 *filespec*
ASCII *filespec*
RASCII *filespec*
YMODEM *filespec*
YMODEMG *filespec*
SEALINK *filespec*

filespec

A valid DOS filename or a variable containing a valid DOS filename.

Examples

```
waitfor "Begin your transfer"
sendfile zmodem "FILE.EXT"
```

Sends a file with ZMODEM.

```
string ufile
message "Enter filename:"
```

Prompts user for filename.

```
get ufile
transmit "RECEIVE"
transmit ufile
transmit "^M"
sendfile kermit ufile
```

Sends the filename and then sends the file to a Kermit server.

Comments

To perform an upload, you must first initiate the receive on the remote; begin your transfer only after the remote indicates that it is ready. All transfers may be tested for successful completion with the commands IF SUCCESS or IF FAILURE.

Pre-existing files can pose a problem if you're using the ZMODEM transfer protocol. If a file you're sending already exists and *crash recovery* is not enabled, the file will be skipped—however, the IF SUCCESS statement will still return a value of "TRUE". To avoid this, enable crash recovery or delete the file on the remote system before using SENDFILE.

For more information on protocols, see the appendix entitled "File-Transfer Protocols" in your USER MANUAL.

See also

GETFILE.

SET

Changes system parameters which control various PROCOMM PLUS and ASPECT operations. SET commands can customize the settings in the Setup facility and the Line Settings screen for your applications.

SET *parameter value*

parameter	One or two keywords identifying the parameter to be changed.
value	The new setting for this parameter. Value can be a keyword, integer, long or string (depending on the parameter requirements).

For details about the following parameters, please refer to Chapter 8 of your USER MANUAL.

ABORTDL KEEP | DELETE

Determines whether PROCOMM PLUS will KEEP or DELETE incomplete and aborted downloads. The default is KEEP, and FETCH returns a 0 for KEEP and a 1 for DELETE.

ALARM OFF | ON

Sets the alarm sound OFF or ON.

ANSI8BIT OFF | ON

Sets 8-bit ANSI Escape sequences OFF or ON. The default is OFF (7-bit sequences).

ASCII 8STRIP OFF | ON

Strips the 8th bit from each character sent or received during an ASCII file transfer.

ASCII BLANKEX OFF | ON

Controls expansion of blank lines during ASCII uploads.

***ASCII CHARPACE** integer*

Sets the character pacing in milliseconds (valid values range from 0 to 999).

ASCII DN_CR CR | STRIP | CR_LF

Controls translation of incoming carriage returns during ASCII downloads. FETCH returns a 0 for CR, a 1 for STRIP and a 2 for CR_LF.

ASCII DN_LF LF | STRIP | CR_LF

Controls translation of incoming line-feeds during ASCII downloads. FETCH returns a 0 for LF, a 1 for STRIP and a 2 for CR_LF.

***ASCII DN_TO** integer*

Sets the timeout delay for ASCII downloads; the value can range from 0 to 30000 seconds between received characters.

ASCII ECHO OFF | ON

Controls local echo during ASCII uploads.

***ASCII LINEPACE** integer*

Sets line pacing timing in 1/10 seconds (valid values range from 0 to 99).

***ASCII PACECHAR** character*

Sets the pace character used. Specify this as an ASCII decimal value (valid values range from 0 to 255).

ASCII TABEX OFF | ON

Converts the TAB character to eight spaces during ASCII uploads.

ASCII UP_CR CR | STRIP | CR_LF

Controls translation of outgoing carriage-returns in ASCII uploads. FETCH returns a 0 for CR, a 1 for STRIP and a 2 for CR_LF.

ASCII UP_LF LF | STRIP | CR_LF

Controls translation of outgoing line-feeds in ASCII uploads. FETCH returns a 0 for LF, a 1 for STRIP and a 2 for CR_LF.

ASPDEBUG OFF | ON

Controls whether or not ASPCOMP will display an offset location as a part of run-time error messages. The default is OFF. When ASPDEBUG is set to ON, the message "ASPECT file aborted" is displayed when a script is aborted with the Esc key.

ATIME integer

Sets amount of time alarm sounds (valid values range from 0 to 9999).

BACKSPACE NONDEST | DEST

Controls destructive nature of received backspace characters. FETCH returns a 0 for NONDEST and a 1 for DEST.

BAUD long

Sets the baud rate. Possible values are 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600 and 115200. The SET BAUD statement is a synonym for SET BAUDRATE.

BAUDRATE long

Sets the baud rate. Possible values are 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600 and 115200. The SET BAUDRATE statement is a synonym for SET BAUD.

BLOCKCUR OFF | ON

Specifies a square block cursor or a line cursor.

BREAK integer

Sets the length (in milliseconds) of a break sent by the BREAK command (valid values range from 0 to 32767).

CALLOG OFF | ON

Determines whether PROCOMM PLUS will keep a log of all completed calls. This command must be set *before* a connection is made for the log to be updated correctly.

CDINXFER NO | YES

Determines whether or not carrier detect will be monitored during file transfer operations.

CHATMODE CHARACTER | BLOCK

Specifies BLOCK or CHARACTER transmission of data in Terminal mode Chat. FETCH returns a 0 for CHARACTER and a 1 for BLOCK.

CLIPCHAR character

Specifies the Clipboard filename separator character.

CR CR | CR_LF

Controls incoming CR translation. FETCH returns a 0 for CR and a 1 for CR_LF.

DATABITS integer

Sets the data bits used; possible values are 7 or 8.

DECIMAL length

Sets the number of digits to the right of the decimal point to print for floating point numbers. This command affects the floating point precision used in ATGET, ATSAY, FTOA and all commands that use a format string (FATSAY, FSTRFMT and STRFMT). Valid values range from 0 to 80, and the default is 2. The value doesn't change if you chain to another script; however, any precision occurring in a format string overrides the default precision, and the ATGET command only uses the default precision when displaying a default floating point value. The last digit in the displayed value is rounded (for example, if the default precision is 2, a value of 123.456789 would become 123.46). For more information, see the descriptions for each of the above commands in this manual.

DIALENTRY integer

Loads the information from the specified Dialing Directory entry into the "$D_" family of system variables; possible values range from 1 to 200.

DISPLAY OFF | ON

Controls whether characters received from the remote system are displayed. Use DISPLAY OFF, for example, if you don't want to see characters displayed in menu-driven script files. Note that no characters will be written to an open logfile if DISPLAY is OFF.

DLDIR string

Sets the default download drive and directory; maximum length for DLDIR is 64 characters.

DLXPROTO1 string

Specifies the program to use as External Download Protocol 1; maximum length is 15 characters.

DLXPROTO2 string

Specifies the program to use as External Download Protocol 2; maximum length is 15 characters.

DLXPROTO3 string

Specifies the program to use as External Download Protocol 3; maximum length is 15 characters.

DROPDTR NO | YES

Specifies whether PROCOMM PLUS will drop DTR when you press the Alt-H terminal sequence. This doesn't affect the **SET MODEM DROPDTR** command, which controls how PROCOMM PLUS hangs up from the Dialing Directory.

DUPLEX FULL | HALF

Sets the duplex mode. **FETCH** returns a 0 for FULL and a 1 for HALF.

EDITOR string

Defines a program name to invoke with the Alt-A terminal sequence; maximum length for the editor string is 64 characters.

EMULATION terminal

Defines the active terminal type, clears the screen and updates the status line. The **FETCH** form of this SET command returns a numeric variable value corresponding to the terminal type. SET EMULATION is functionally equivalent to the **EMULATE** command. For the complete list of terminal values, please refer to the definition for the **FETCH** command.

ENQ OFF | ON | CISB

Controls response to ENQ (ASCII 5). **FETCH** returns a 0 for OFF, a 1 for ON and a 2 for CISB.

EXITCDHIGH IGNORE | HANGUP | ASK

Determines the action PROCOMM PLUS takes if Carrier Detect is high when you exit the program. ASK displays a prompt asking whether or not to hang up, IGNORE continues without prompting and HANGUP automatically disconnects without prompting. **FETCH** returns a 0 for IGNORE, a 1 for HANGUP and a 2 for ASK.

EXPLODE OFF | ON

Enables and disables exploding windows.

EXTPROTO1 string ASPECT | PROGRAM | HOOK

Specifies the name and type for External Download Protocol 1; maximum length for the name is 8 characters. **FETCH** returns a 0 for ASPECT, a 1 for PROGRAM and a 2 for HOOK.

EXTPROTO2** string **ASPECT | PROGRAM | HOOK

Specifies the name and type for External Download Protocol 2; maximum length for the name is 8 characters. **FETCH** returns a 0 for ASPECT, a 1 for PROGRAM and a 2 for HOOK.

EXTPROTO3** string **ASPECT | PROGRAM | HOOK

Specifies the name and type for External Download Protocol 3; maximum length for the name is 8 characters. **FETCH** returns a 0 for ASPECT, a 1 for PROGRAM and a 2 for HOOK.

FASTKBD OFF | ON

Sets the keyboard typomatic and repeat rates on enhanced keyboards between normal and fast mode.

FGETS_CRLF OFF | ON

Controls the function of the **FGETS** command. When set OFF, this command discards the line feed character at the end of a string retrieved with **FGETS**. For files opened in binary mode, it also discards the carriage return before the line feed (and the end-of-file character if it's the last string read from the file). The default is ON.

FINDCASE OFF | ON

Sets case-sensitivity OFF or ON for the **FIND** command. The default is OFF, and the value doesn't change if you chain to another script. For more information, see the **FIND** command description in this chapter.

FNLOOKUP OFF | ON

Determines whether PROCOMM PLUS will "search" the current screen for a valid filename when downloading with certain protocols (such as XMODEM).

HARDFLOW OFF | ON

Enables and disables RTS/CTS hardware flow control.

HOST AUTOBAUD OFF | ON

Sets Host mode Auto baud detect ON or OFF.

HOST CONNECTION MODEM | DIRECT

Selects either the MODEM or DIRECT (wired) connection mode for Host operations. **FETCH** returns a 0 for MODEM and a 1 for DIRECT.

***HOST DLDIR** string*

Specifies the download directory for Host mode operations; maximum length for the download path is 64 characters.

HOST GOODBYE RECYCLE | HANGUP | EXIT

Determines the GOODBYE action taken when exiting Host mode. FETCH returns a 0 for RECYCLE, a 1 for HANGUP and a 2 for EXIT.

HOST MESSAGE string

Contains the Host Welcome Message (maximum length is 50 characters).

HOST NEWUSERLVL 0 | 1

Determines whether new users will be assigned a "0" (limited privilege level, no file transfer capability) or "1" (normal access) at logon.

HOST SHELLBOOT OFF | ON

Determines whether a loss of carrier will reboot Host when the remote user has shelled to DOS.

HOST SYSTYPE CLOSED | OPEN

Sets the Host system between OPEN (allow new users) and CLOSED (don't allow new users). FETCH returns a 0 for CLOSED and a 1 for OPEN.

HOST TIMEOUT integer

Determines the Host Inactivity Timeout value; possible values range from 0 to 999 minutes.

HOST ULDIR string

Specifies the upload directory for Host mode operations; maximum length for the upload path is 50 characters.

INSMODE OFF | ON

Sets insert mode OFF or ON for user-defined terminal emulation.

KERMIT 8QUOTE character

Selects the 8th bit quote character; valid values range from 33 to 126.

KERMIT BLOCKCHECK 1 | 2 | 3

Selects the block check type, where 1 is a 1 byte checksum, 2 is a 2 byte checksum and 3 is a 3 byte CRC.

KERMIT CQUOTE character

Sets the Kermit control quote character; valid values range from 32 to 127.

KERMIT EOLCHAR character

Selects the Kermit end-of-line character; valid values range from 0 to 127.

KERMIT FILETYPE TEXT | BINARY

Selects the Kermit transfer file type. FETCH returns a 0 for TEXT and a 1 for BINARY.

KERMIT HANDSHAKE character

Selects the Kermit handshake character; valid values range from zero to 31.

KERMIT PACKSIZE integer

Sets the Kermit maximum packet size; valid values range from 20 to 1024.

KERMIT PADCHAR character

Selects the pad character; valid values range from 0 to 127.

KERMIT PADNUM integer

Sets the number of pad characters; valid values range from 0 to 127.

KERMIT STARTCHAR integer

Specifies the Kermit block start character; valid values range from 0 to 127.

KEYS OFF | ON

Controls how keystrokes are processed when a script is executed. When KEYS are OFF (the default), PROCOMM PLUS will check for keystrokes before each command is executed. If the key is anything other than the Esc key, it will be processed normally (as a command or a character sent to the remote system). If the Esc key is pressed, the user is asked whether or not to terminate the script. If KEYS are ON, however, the script is expected to check for and process each keystroke (the TERMKEY command can be used to process the command or pass keystrokes to the remote system). For example, if you press PgDn while a script file is being executed and KEYS are OFF, the window displaying the file-transfer protocols appears, just as in Terminal mode. If KEYS are ON, however, the PgDn keystroke is held in the keyboard buffer until it's processed by a script command (such as KEYGET). The value of KEYS doesn't change if you chain to another script. The SET KEYS command replaces the SET MENUMODE command used in earlier versions of PROCOMM PLUS.

LOGFILE string

Specifies the default name for the Log file invoked with the Alt-F1 terminal sequence; maximum length for the Log file is 64 characters.

MODEM AUTOANSOFF string

Contains the modem string to disable auto answer operation; maximum length for this field is 24 characters.

MODEM AUTOANSON string

Contains the modem string to enable auto answer operation; maximum length for this field is 24 characters.

MODEM AUTOBAUD OFF | ON

Sets auto baud detection ON or OFF.

MODEM CALLPAUSE integer

Sets the amount of seconds PROCOMM PLUS will pause between calls; valid values range from 0 to 999.

MODEM CDHIGHINIT NO | YES

Controls whether PROCOMM PLUS will send the modem initialization string if Carrier Detect is high when you start the program.

MODEM CNCT300 string

Contains the message sent by the modem for a 300 baud connection; maximum length for this field is 15 characters.

MODEM CNCT1200 string

Contains the message sent by the modem for a 1200 baud connection; maximum length for this field is 15 characters.

MODEM CNCT2400 string

Contains the message sent by the modem for a 2400 baud connection; maximum length for this field is 15 characters.

MODEM CNCT4800 string

Contains the message sent by the modem for a 4800 baud connection; maximum length for this field is 15 characters.

MODEM CNCT9600 string

Contains the message sent by the modem for a 9600 baud connection; maximum length for this field is 15 characters.

MODEM CNCT19200 string

Contains the message sent by the modem for a 19200 baud connection; maximum length for this field is 15 characters.

MODEM CNCT38400 string

Contains the message sent by the modem for a 38400 baud connection; maximum length for this field is 15 characters.

MODEM DIALCMND string

Contains the modem dialing command string; maximum length for this field is 24 characters.

MODEM DIALSUFFIX string

Contains the modem dialing command suffix string; maximum length for this field is 24 characters.

MODEM DROPDTR NO | YES

Specifies whether PROCOMM PLUS will drop DTR to hang up.

MODEM HANGUP string

Contains the hangup command string; maximum length for this field is 24 characters.

MODEM INIT string

Contains the modem initialization string; maximum length for this field is 46 characters.

MODEM MAXDIAL integer

Specifies the maximum number of redial attempts; valid values range from 0 to 999.

MODEM NOCNCT1 string

Contains the first message sent by the modem for no connection connection; maximum length for this field is 15 characters.

MODEM NOCNCT2 string

Contains the second message sent by the modem for no connection connection; maximum length for this field is 15 characters.

MODEM NOCNCT3 string

Contains the third message sent by the modem for no connection connection; maximum length for this field is 15 characters.

MODEM NOCNCT4 string

Contains the fourth message sent by the modem for no connection connection; maximum length for this field is 15 characters.

MODEM SENDCR NO | YES

Specifies whether PROCOMM PLUS will send a carriage return to the modem after hanging up.

MODEM WAITCNCT integer

Determines the maximum amount of seconds PROCOMM PLUS will wait for a connection during a call; valid values range from 0 to 999.

MOUSEX *integer*

Sets the horizontal mouse sensitivity; sensitivity decreases as the value increases. Valid values range from 1 to 999.

MOUSEY *integer*

Sets the vertical mouse sensitivity; sensitivity decreases as the value increases. Valid values range from 1 to 999.

MSG_CRLF *OFF | ON*

Controls the function of the **MESSAGE** command. When set OFF, automatic carriage returns and line feeds used in the **MESSAGE** command are suppressed. The default is ON.

PARITY *NONE | ODD | EVEN | MARK | SPACE*

Controls the parity setting for this session. **FETCH** returns a 0 for NONE, a 1 for ODD, a 2 for EVEN, a 3 for MARK and a 4 for SPACE.

PAUSECHAR *integer*

Identifies the character PROCOMM PLUS translates into a half-second pause within modem commands, strings sent with the **TRANSMIT** command or character strings (like the Host mode welcome message). The default character is a tilde (~), and valid values range from 32 to 126.

PORT *COM1 | COM2 | COM3 | COM4 | COM5 | COM6 | COM7 | COM8* [*base* [*IRQ*]]

Selects the serial port. Optional base and IRQ address parameters can be included; they're treated as unsigned values.

PROTECT *OFF | ON*

Sets a display field as protected or normal for user-defined emulations.

PROTOCOL *protocol*

Selects a default protocol for uploading and downloading; the value must be a valid protocol name.

PRTNAME *string*

Defines the device to be used for print operations. A maximum of 4 characters is allowed in the device name.

PULLDNKEY *integer*

Selects the key to invoke the pull-down menu system; valid values range from ASCII 32 to 126.

PULLDNMENU *OFF | ON*

Sets the pull-down menu system OFF or ON.

RANGECHK OFF|ON

Sets script *value range checking* OFF or ON. If ON, an out-of-range value will issue an error message during execution, allowing the user to end the script (if the script is not aborted, the value may cause unpredictable results). A setting of OFF forces PROCOMM PLUS to ignore the out-of-range value and continue. The default is ON, and the value doesn't change if you chain to another script.

REDISPLAY integer

Controls the size of the Terminal mode Redisplay buffer (in kilobytes). Legal values range from 1 to 63.

RELAXED OFF|ON

When set to ON, this parameter increases the amount of time PROCOMM PLUS will wait for a block before timing out during a file transfer.

REMOTECMD OFF|ON

Determines whether PROCOMM PLUS will accept script commands sent by a remote PC.

RXDATA OFF|ON

Specifies whether PROCOMM PLUS or a script will process characters received at the active com port. The default, OFF, indicates that PROCOMM PLUS will process received characters; if RXDATA is ON, your script must handle incoming data (with commands like COMGETC and TERMWRT). If RXDATA is set ON, any active WHEN statements will only be checked when COMGETC or COMGETCD are called.

SCREEN 25X80|EXTRAX80|USERMODE|EXTRAXUSER

Toggles the current video mode (unlike VIDSTART, which sets the startup mode); functionally, it's similar to the Ctrl-- Terminal mode sequence. FETCH returns a 1 for 25X80, a 2 for EXTRAX80, a 3 for USERMODE and a 4 for EXTRAXUSER.

SCROLL OFF|ON

Determines scroll setting. If ON, characters or messages on the last row and column of the screen will scroll; if OFF, this field forces PROCOMM PLUS to overwrite characters on the last row with new data.

SNAPSHOT string

Specifies the default filename for the Screen Snapshot (invoked with the Alt-G Terminal mode sequence); maximum length for this field is 64 characters.

SNOW OFF|ON

Removes interference (or "snow") on older CGA monitors during direct screen write operations.

SOFTFLOW OFF|ON

Enables or disables XON/OFF (or "software") flow control.

SOUND OFF|ON

Controls sound effects.

STATLINE OFF|ON

Turns the status line at the bottom of the Terminal mode screen ON or OFF. When STATLINE is OFF, PROCOMM PLUS can use the entire screen for terminal display.

STOPBITS 1|2

Sets the stop bits used for the current session.

SWITCHCASE OFF|ON

Sets case-sensitivity OFF or ON when the SWITCH command is using string values. The default is OFF, and the value doesn't change if you chain to another script. For more information, see the SWITCH command description in this chapter.

TERMBOLD attribute

Selects the color used to display high-intensity characters in Terminal mode.

TERMDIM integer

Selects the color used to display low-intensity characters in Terminal mode.

TERMNORM integer

Selects the color used to display normal characters in Terminal mode.

TERMREV integer

Selects the color used to display reverse characters in Terminal mode.

TERMULINE integer

Selects the color used to display underlined characters in Terminal mode.

TERMWIDTH 80|132

Controls whether PROCOMM PLUS will use an 80-column screen or a 132-column screen (regardless of the actual columns your hardware

can display). For more information, see "Terminal Options" in Chapter 8 of your USER MANUAL.

TRANSLATE OFF | ON | STRIP

Controls the use of the translate table. The STRIP option allows the removal of the 8th bit from all incoming data. FETCH returns a 0 for OFF, a 1 for ON and a 2 for STRIP.

***TXPACE** integer*

Determines character pacing in milliseconds for all outgoing character strings.

ULINEFONT OFF | ON

When set ON, this parameter enables "true" underlining for EGA/VGA color monitors.

***ULXPROTO1** string*

Specifies the program to use as External Upload Protocol 1; maximum length for this field is 15 characters.

***ULXPROTO2** string*

Specifies the program to use as External Upload Protocol 2; maximum length for this field is 15 characters.

***ULXPROTO3** string*

Specifies the program to use as External Upload Protocol 3; maximum length for this field is 15 characters.

***USERVID** integer*

Sets the value sent to your graphics adaptor to select the "USER MODE" video mode. Values range from 0 to 255. For more information on video modes, see "Display/Sound Options" in Chapter 8 of your USER MANUAL.

VGALINES 28 | 43 | 50

Specifies the number of extra lines PROCOMM PLUS will display on EGA and VGA monitors; note that EGA supports only 43 lines.

VIDSTART DOSMODE | 25X80 | EXTRAX80 | USERMODE | EXTRAXUSER

Sets the video mode PROCOMM PLUS will use each time you start the program. FETCH returns a 0 for DOSMODE, a 1 for 25X80, a 2 for EXTRAX80, a 3 for USERMODE and a 4 for EXTRAXUSER. For more information, see "Display/Sound Options" in Chapter 8 of your USER MANUAL.

***VIEWUTIL** string*

Defines a program name to invoke with the Alt-V Terminal mode sequence; maximum length for this field is 64 characters.

WAITCASE OFF|ON

Sets case-sensitivity OFF or ON for the **WAITFOR** command. The default is OFF, and the value doesn't change if you chain to another script. For more information, see the **WAITFOR** command description in this manual.

WHENCASE OFF|ON

Sets case-sensitivity OFF or ON for the **WHEN** command. The default is OFF, and the value doesn't change if you chain to another script. For more information, see the **WHEN** command description in this manual.

WRAP OFF|ON

Controls line wrap in Terminal mode.

WRITEPROT OFF|ON

Sets a display field as write-protected or normal for user-defined emulation.

XFERKEY REG|CTRL

Selects the keys to use to initiate a file transfer. REG uses the default keys PgUp and PgDn, while CTRL uses Ctrl–PgUp and Ctrl–PgDn.

ZMODEM AUTODLOAD OFF|ON

Enables and disables ZMODEM auto downloading.

ZMODEM ERRDETECT CRC32|CRC16

Selects ZMODEM 16-bit or 32-bit CRC error detection. **FETCH** returns a 0 for CRC32 and a 1 for CRC16.

ZMODEM RECVCRASH PROTECT|NEGOTIATE|ON|OVERWRITE

Determines ZMODEM crash recovery action for received files. **FETCH** returns a 0 for PROTECT, a 1 for NEGOTIATE, a 2 for ON and a 3 for OVERWRITE.

ZMODEM SENDCRASH PROTECT|NEGOTIATE|ON|OVERWRITE

Determines ZMODEM crash recovery action for files being sent. **FETCH** returns a 0 for PROTECT, a 1 for NEGOTIATE, a 2 for ON and a 3 for OVERWRITE.

ZMODEM TIMESTAMP OFF|ON

Sets ZMODEM time/date stamping ON or OFF.

ZMODEM TXMETHOD STREAMING|2KWINDOW|4KWINDOW

Selects ZMODEM window/streaming transmission method. **FETCH** returns a 0 for STREAMING, a 2 for 2KWINDOW and a 4 for 4KWINDOW.

Comments

Values shown as *"character"* should use the ASCII decimal value of the desired character. For example, to use XON (ASCII 17) as the KERMIT handshake character, issue the command **SET KERMIT HANDSHAKE 17.**

See also

FETCH.

SETFATTR

Sets a file's attributes.

SETFATTR *filespec string*

Example

setfattr "pass.dat" "hr"	Sets the file PASS.DAT with the attributes Hidden and Read-Only.

Comments

Attributes can include "R" (Read-Only), "H" (Hidden), "S" (System) and "A" (Archive).

This command can be tested with the **IF SUCCESS** and **IF FAILURE** statements.

See also

GETFATTR.

SETFDATE

Sets a file's date stamp.

SETFDATE *filespec string*

Example

setfdate "A1.DAT" "01/01/91"	Sets the date stamp for the file A1.DAT at 01/01/91.

Comments

This command can be tested with the IF SUCCESS and IF FAILURE statements.

See also

GETFDATE and SETFTIME.

SETFTIME

Sets a file's time stamp.

SETFTIME *filespec string*

Example

setftime "DAN.TXT" "00:33"	Sets the creation time for the file DAN.TXT at 00:33 (military time).

Comments

SETFTIME only sets the hour and minute of a file's time stamp.

This command can be tested with the IF SUCCESS and IF FAILURE statements.

See also

SETFDATE and GETFTIME.

SETJMP

Marks a location within a script that you can immediately "jump" to with the LONGJMP command.

SETJMP *index intvar*

index	One of three integer index values (0,1 or 2); the value identifies the location to the LONGJMP command.
intvar	An integer variable initialized to 0 at execution. This variable will contain a new value when the LONGJMP is executed, allowing further processing (dependent on the script's execution at the point of the SETJMP).

Example

```
setjmp 0 test
if test
   switch test
      case 1
        clear
      endcase
      default
        exit
      endcase
   endswitch
endif
```

Creates a SETJMP location at the IF command. The location is referenced by the index number 0, and the integer variable TEST will receive the value passed by the corresponding LONGJMP command.

Comments

The marked location is the command following the SETJMP. Up to three SETJMP locations can be active at once.

The marked location remains active until a RETURN from the function where it was set—or until another routine uses a SETJMP with the same index.

See also

LONGJMP.

SHELL

Suspends PROCOMM PLUS and drops to the DOS command prompt. You may run any program that can be normally run from DOS at this time. To return to PROCOMM PLUS, type *exit* at the DOS command line.

SHELL

Comments

IMPORTANT: Use caution while in SHELL mode! Programs using the COM port may be hazardous to the communications link.

Any program or DOS command you run from SHELL must be either in the current directory or in the directory specified in your DOS path. Also, if you're using a hard disk system, COMMAND.COM must be in the directory from which you started your computer (typically the root directory of drive C). If you're using a floppy-disk system, COMMAND.COM must be on the disk in drive A.

Be sure that you have enough memory to run any SHELL command or program **and** PROCOMM PLUS at the same time.

SHELL performs the same function as the DOS Gateway Terminal mode command.

See also

METAKEY, DOS and RUN.

SHL

This command performs a left shift operation on the bits of one number and places the result in the specified numeric variable. The number of shifts performed is designated by the second number. For each shift performed, each bit in the number is shifted one bit position to the left. The most significant bit is discarded, and a 0 is stored in the least significant bit position. The result of each individual shift operation is the same as multiplying the number by two.

SHL *number number numvar*

Comments

This command cannot be used with floating point numbers. The *command* form "SHL N0 N1 N2" is equivalent to the *operator* form "N2=N0<<N1".

See also

AND, XOR, COMP and SHR.

SHR

This command performs a right shift operation on the bits of one number and places the result in the specified numeric variable. The number of shifts performed is designated by the second number. For each shift performed, each bit in the number is shifted one bit position to the right. The least significant bit is discarded, and a 0 is stored in the most significant bit position. The result of each individual shift operation is the same as dividing the number by two.

SHR *number number numvar*

Comments

This command cannot be used with floating point numbers. The *command* form "SHR N0 N1 N2" is equivalent to the *operator* form "N2=N0>>N1".

See also

AND, XOR, COMP and SHL.

SNAPSHOT

Copies the contents of the current screen to the disk file PCPLUS.SCR (or the filename specified as the default for screen snapshot files; see "File/Path Options" in Chapter 8 of your USER MANUAL. If the file exists, it appends the new screen to the existing file. If the file does not exist, it opens the file and then writes the information.

SNAPSHOT

Example

snapshot	Copies the screen contents to the PCPLUS.SCR file.

See also

LOG and PRINTER.

SOUND

Produces a sound of the specified frequency and duration.

SOUND *integer integer*

integer	Frequency of the sound in Hertz. The frequency can range from 21 to 32767 Hz.
integer	Length of time for sound, in hundredths of a second. Duration can range from 0 to 32767.

Example

```
sound 440 500
```

Sounds an "A" note for 5 seconds.

STATMSG

Displays a message centered on the Status line.

STATMSG *string*

string	A string of up to 80 characters.

Example

```
string umsg
statmsg "Enter password"
get umsg 8
statrest
```

Displays the message "Enter password" and restores the Status line.

Comments

The message remains onscreen until another command clears or resets it. If the Status line is SET ON, any PROCOMM PLUS function key will also erase the message; if the Status line is SET OFF, the message will be erased only when it's overwritten by incoming data, scrolls off of the screen or is cleared with another command (like CLEAR or ATSAY).

See also

ERRORMSG and USERMSG.

STATREST

Updates and redisplays the Status line (if the Status line is currently SET ON). The command has no effect if the Status line is SET OFF.

STATREST

See also

SET STATLINE and STATMSG.

STRCAT

Concatenates one string variable or quoted string to another string variable.

STRCAT *strvar string* [*length*]

strvar	The variable to which the second string is concatenated. The result string must be less than or equal 80 characters in length.
string	The variable concatenated to *strvar*.
length	An optional integer specifying the maximum length to be copied from the second string.

Example

```
string front="STR"
string back="CAT"
strcat front back
```

Adds the string variable BACK to the end of FRONT. The value of FRONT is now "STRCAT".

See also

STRUPTDT and STRFMT.

STRCMP

Compares two strings and sets IF SUCCESS to "TRUE" if the strings are identical.

STRCMP *string string* [*length*]

length — An optional integer specifying the maximum number of characters to be compared.

Comments

STRCMP is case-sensitive.

See also

FIND, RSTRCMP, STRUPR and STRLWR.

STRCPY

Assigns a string variable or quoted string to another string variable.

STRCPY *strvar string* [*length*]

string — The second string can be either a string variable or a quoted string and is the string to be copied.

length — An optional integer specifying the maximum number of characters to copy from the second string.

Examples

strcpy lname "Henry M." 5 — Assigns the value HENRY to the variable LNAME.

Comments

The command word **ASSIGN** is a synonym for STRCPY.

Strings can also be assigned to each other with the assignment operator "=" (for example, S0=S3).

See also

STRFMT, SUBSTR, STRUPDT and ASSIGN.

STRFMT

Creates a formatted string using a template and modifies it with string or numeric variables. This is similar to "sprintf" in the C programming language.

STRFMT *strvar formatstr* [*param*]...[*param*]

strvar	Variable where the formatted string is stored.
formatstr	For a complete description of the valid format specifiers, please refer to the definition for **FATSAY** in this chapter.

Comments

Values are taken sequentially (left-to-right) from the first parameter listed to the last for format types found in the *formatstr*. The maximum number of parameters is 10.

Example

```
init N1 29
strfmt S2 "I'm %d years old" N1
message S2
```

Prints "I'm 29 years old".

See also

FATSAY and FSTRFMT.

STRING

Defines a global or local string variable.

STRING *name*[=*string*][, *name*[=*string*]]...

[=*string*]	An initializer expression; it can be a quoted string or a previously-declared string variable.

Examples

`string name = "jody"`	Defines the string variable NAME and initializes it to "jody".

Comments

For more information on using *global* and *local* variables, please refer to the definition for **FLOAT**.

See also

STRPARM, LONG, FLOAT AND INTEGER.

STRLEN

Returns the length of a string or string variable's contents into an integer variable.

STRLEN *string intvar*

Example

```
string bbs
strlen bbs len
```

Returns the length of the string variable BBS into the integer variable LEN.

STRLWR

Converts the contents of a string variable to all lowercase characters.

STRLWR *strvar*

Example

```
string name
strlwr name
```

Converts the contents of the string variable NAME to all lowercase.

See also

STRUPR.

STRPARM

Defines a string parameter variable.

STRPARM *name*[, *name*]...[,*name*]

Example

strparm bbsname — Defines the string parameter BBSNAME.

Comments

Any procedure—except MAIN—can be defined with up to 10 *parameter variables*. Parameter variables are similar to local variables in that they may only be referenced within the procedure in which they're defined; unlike local variables, though, the value of parameters are automatically initialized when the procedure is called.

For more information on using parameter variables, please refer to the definition for **FLOATPARM**.

See also

STRING, INTPARM, FLOATPARM and LONGPARM.

STRPEEK

Returns the ASCII value of a single character within a string.

STRPEEK *string strindex intvar*

Example

```
integer char
string origin
strpeek origin 7 char
```

Returns the value of the byte at position 7 in the string variable ORIGIN into the integer variable CHAR.

Comments

The character position is zero-based from the beginning of the string (the first position in the string variable is position 0).

See also

SUBSTR and STRPOKE.

STRPOKE

Sets the ASCII value of a single character within a string.

STRPOKE *strvar strindex character*

Example

```
integer newchar=65
string origin,text
strpoke text 7 newchar
```

Sets the value of the character at position 7 in the string variable ORIGIN with the value of the integer variable NEWCHAR.

Comments

The character position is zero-based from the beginning of the string (the first position in the string variable is position 0).

See also

STRPEEK and STRUPDT.

STRSET

Initializes the contents of a string variable up to the designated length.

STRSET *strvar character length*

Example

```
string name
strset name 32 40
```

Initializes the value of the string variable NAME with the value 32 for a length of 40 bytes.

STRUPDT

Overwrites a string with another string beginning at a specified index.

STRUPDT *strvar string strindex length*

Example

```
string output, newdata
strupdt output ndata 3 10
```

Sets the values starting at position 3 in the string variable OUTPUT with the value of the string variable NDATA. A total of 10 characters will be overwritten.

Comments

This command is similar in function to STRPOKE (except that the characters replaced are taken from a string instead of an integer).

See also

STRPOKE and SUBSTR.

STRUPR

Converts the contents of a string variable to all uppercase characters.

STRUPR *strvar*

Example

```
string name
strupr name
```

Converts the contents of the string variable NAME to all uppercase.

See also

STRLWR.

SUB

Subtracts the second number from the first and then stores the result in the last numeric variable.

SUB *number number numvar*

Comments

It is possible to subtract one number from another number and have the result beyond the range of numeric variables; PROCOMM PLUS does no error checking on this value.

The *command* form "SUB N1 N2 N3" is equivalent to the *operator* form "N3=N1-N2".

See also

DEC and DIV.

SUBSTR

Copies the indicated number of characters from a string, beginning at a specified position.

SUBSTR *strvar string strindex length*

strvar	The resultant string.
string	The source for the substring extraction.
strindex	The character position in the source string that starts the resultant string. The position is counted from zero; for example, in the string "Tom" the character *T* is position 0 and the character *o* is position 1.
length	This is the number of characters in the source string that will be extracted. If the length is greater than the remaining length of the source string, SUBSTR will copy the remaining characters from the source string.

Examples

```
string phone
string data
substr phone data 1 7
```

Takes a substring from DATA starting at position 1 for 7 characters in length, and stores it in PHONE.

```
assign s1 "PROCOMM"
substr s2 s1 3 3
```

Takes a substring from S1 starting at position 3 for 3 characters in length, and stores it in S2. Since we begin counting positions at 0, S2 would now be the string "COM".

See also

STRPEEK, STRUPDT, STRCPY and FIND.

SUSPEND UNTIL

Waits until the given time to continue execution.

SUSPEND UNTIL *integer integer*

integer integer	The first integer (the hour) is in 24 hour format, with a range of 0 to 23. The second integer (the minute) has a range of 0 to 59.

Example

SUSPEND UNTIL 13 30	Waits until 1:30 PM to continue.

Comments

SUSPEND UNTIL can be exited with the Esc key (**IF FAILURE** is set "TRUE"). All other keys are ignored.

See also

WAITQUIET, PAUSE and WAITFOR.

SWITCH

Provides multiple decision points by comparing a string or integer to one or more values. SWITCH requires the use of **CASE, ENDCASE** and **ENDSWITCH** commands; the use of the **DEFAULT** command is optional.

SWITCH *string | integer*
  CASE *string | integer*
  .
  .
  .
ENDCASE
  [DEFAULT
  .
  .
  .
ENDCASE]
ENDSWITCH

*string	integer*	The source item to be compared to the string or integer following each **CASE** statement.

CASE string | integer	Compares the variable following the SWITCH command to the "target" of the **CASE** command. If matched, processing continues with each command on subsequent lines until the **ENDCASE** command is encountered. If a match is not found, the processing of subsequent **CASE** or **DEFAULT** commands continues. Items in the **CASE** statement must match the data types used in the SWITCH statement (for example, you can't compare a string to an integer).
ENDCASE	Concludes processing of commands following **CASE** and **DEFAULT** commands and passes control to the command on the line following the **ENDSWITCH** command.
DEFAULT	An optional parameter, which handles cases where no match is found in any preceding **CASE**. This command causes unconditional processing of each command on the lines following until the **ENDCASE** command is encountered. Processing then passes to the command on the line following the **ENDSWITCH** command.

Examples

```
get_choice:
message "Enter your choice"
get S5
switch S5
   case "A"
      gosub choicea
      endcase
   case "B"
      gosub choiceb
   endcase
   case $NULL
      gosub null_case
   endcase
   default
      message "Invalid selection"
      pause 3
      goto get_choice
   endcase
endswitch
```

Gets choice A, B or C.
Branches to a subroutine based on the choice.

Comments

When a match occurs between the source item and the value following a **CASE** command, command processing continues on the line following the **CASE** command until the **ENDCASE** command is encountered. Processing then continues with the command on the line after the **ENDSWITCH** command, which concludes the entire phrase.

SWITCH supports multiple **CASE** commands within a single **CASE-ENDCASE** command block; a single command block can be executed by matching the target item with any one of several **CASE** commands. Since the end of a block is always an **ENDCASE** command, at least one **ENDCASE** must occur whenever **CASE** or **DEFAULT** is used.

If a **CASE** command is matched, the statements following it are executed until an **ENDCASE** or **EXITSWITCH** is encountered. If another **CASE** is encountered, it's skipped and execution will fall through to the next set of commands following that **CASE** command. The **ENDCASE** command is an implied **EXITSWITCH**; once an **ENDCASE** is encountered, the only commands that can follow it are another **CASE**, a **DEFAULT** or an **ENDSWITCH**.

SWITCH command blocks can be nested if required.

The **DEFAULT** command is optional, but there must be only one occurrence of it within a **SWITCH** command block. If a **DEFAULT**

isn't used (and none of the **CASE** commands were matched), no action takes place at all.

CASE and **DEFAULT** statements can occur in any order, but if a match is made with more than one **CASE** command, only the statements following the first matched **CASE** command will be executed.

If you're using strings with SWITCH, you can control the case-sensitivity of a match with the **SET SWITCHCASE** statement.

You can use the SWITCH command phrase to allow a user to make a choice, and then perform different tasks depending upon the option selected. A menu system can be created using the SWITCH command phrase together with the **BOX**, **ATSAY** and **ATGET** commands.

See also

CASE, DEFAULT, ENDCASE, SET SWITCHCASE, ENDSWITCH and EXITSWITCH.

TERMINAL

Exits the script file and returns to Terminal mode (without disconnecting).

TERMINAL

Comments

The commands **EXIT** and **CONNECT** are synonyms for TERMINAL; all three perform the same function.

See also

BYE, CONNECT, EXIT and QUIT.

TERMKEY

Processes the keycode value represented by an integer and performs the corresponding program function.

TERMKEY *integer*

Example

termkey 0x2000	Calls the Dialing Directory (0x2000 is the keycode for Alt-D).

Comments

Typically, TERMKEY is used with the SET KEYS ON statement to "intercept" keystrokes. If the keycode value corresponds to a program function (such as the Dialing Directory), that function will be performed. If the value corresponds to a "special" key (for instance, the function keys F1 through F10), PROCOMM PLUS sends the Keyboard Map codes for the current emulation for that key. If the value is a standard key (like a letter of the alphabet), the keycode will be sent to the remote unchanged.

TERMKEY treats the keycode value as unsigned.

See also

HITKEY, SET KEYS, KEYGET and KEY2ASCII.

TERMRESET

Resets the terminal mode display. The screen is cleared, the cursor is moved to row 0, col 0 and the emulation is re-initialized.

TERMRESET

TERMWRT

Displays the ASCII character corresponding to the keycode value at the current cursor position and moves the cursor to the next available location.

TERMWRT *integer*

Example

```
integer char=65
termwrt char
```

Displays the character contained in the integer CHAR ("A") at the current cursor position. The cursor is moved to the next available position.

Comments

The keycode value is treated as unsigned, and range checking is suppressed for this command. Unlike WRITEC, no conversion is done on the value.

See also

WRITEC, KEYGET, HITKEY, SET KEYS and MESSAGE.

TIME

Places the current time into a string variable.

TIME *strvar* 0|1

0|1

0 = Regular AM/PM format—
HH:MM:SSAM or HH:MM:SSPM

1 = 24-hour military format—
HH:MM:SS

Example

```
string current
time current 1
```

Gets the system time and places it in string variable CURRENT in 24-hour military format.

See also

$TIME0, $TIME1 and DATE.

TRANSMIT

Sends a character string to the remote system or to the modem.

TRANSMIT *string*

string — May contain non-printing characters using standard translation conventions (see "Translating Control Codes" in the USER MANUAL appendix titled "Technical Information").

Examples

```
transmit "ATDT1 314 875-0503^M"
pause 10
```

Transmits dial command and phone number to modem number and then pauses 10 seconds.

```
string id
if connected
.
.
.
endif
message "Enter your ID"
get id
transmit id
transmit "^M"
```

Tests for connection; performs addtional processing if connected.

Receives user id from local user. Send to remote system ... with a carriage return.

```
transmit "Dave said '"Hello'""
```

Sends "Dave said "Hello"" to the remote system. If you want to send a quotation mark (") within the character string, you must precede it with the single back quote character (‘). This back tic character (ASCII 96) is

on the key that has the tilde (~) above it.

Comments

Note that TRANSMIT does *not automatically* add a carriage return at the end of a character string. To send one, use the ^M control code (as illustrated in the example).

TYPE

Displays a file to the local screen with paging support.

TYPE *filespec*

filespec	A valid filename or path and filename.

Comments

TYPE should only be used with ASCII text files. It's equivalent to the Terminal mode View File command.

ULINEON

Turns on the underline attribute for Terminal mode display.

ULINEON

See also

REVON, BOLDON, DIMON and NORMON.

UNDEF

Removes the current definition of a previously-**DEFINEd** macro.

UNDEF *name*

Examples

UNDEF MYNAME	Removes the macro definition for the name MYNAME.

Comments

UNDEF is paired with a **DEFINE** command to create a "region" in a script where an identifier has a special meaning. UNDEF must also be used prior to redefining a macro that was previously **DEFINE**d.

See also

DEFINE and $IFDEF.

USERMSG

Displays a message centered in a box on the local screen.

USERMSG *string*

string	A string of up to 80 characters.

See also

A short bell sound accompanies the message. The message will be erased when the user presses a key (or, if no action is taken, in two seconds).

See also

ERRORMSG and STATMSG.

VIDREST

Restores the screen with video buffer data from a specified memory buffer.

VIDREST *index*

index	Specifies the screen memory buffer containing the data to restore—values can range from 0 to 2.

Example

```
vidsave 0
emulate ansi
vidrest 0
```

Saves the current contents of the video buffer into screen memory buffer 0, switches to ANSI emulation and restores the screen from that same buffer.

Comments

The video buffer can only be restored once; to perform multiple restores, you must first VIDSAVE the data again or perform the restore from another active buffer index.

See also

VIDSAVE.

VIDSAVE

Saves the current video buffer data to memory.

VIDSAVE *index*

index	Specifies the screen memory buffer for this VIDSAVE—values can range from 0 to 2.

Example

vidsave 0	Saves the current contents of the video buffer into screen memory buffer 0.

Comments

Your script can use the IF SUCCESS or IF FAILURE statements to test whether or not the screen buffer was actually saved.

If VIDSAVE is called with an already-active index, the previous contents of the video buffer will be lost.

See also

VIDREST.

WAITFOR

Causes a pause in processing until a target string is received from the remote system.

WAITFOR *string* [*integer* | FOREVER]

string	A string of text and numeric characters up to 80 characters in length. You can also include control characters in the target. Case-sensitivity can be specified with the **SET WAITCASE** statement.
integer	An integer value, which determines how many seconds to wait for the target before timing out and continuing execution. If no delay nor FOREVER is specified, the default of 30 seconds will be used.
FOREVER	If this is specified for the delay, PROCOMM PLUS will wait indefinitely for the target; it will not time out.

Examples

waitfor "first name:" 45	Waits 45 seconds for prompt.

if waitfor
** transmit "TOM^M"**
else
** call error**
endif
	Tests and proceeds based on the result.

waitfor "^M^JBUSY"	Uses control characters; waits for Carriage Return, Line Feed and "Busy".

Comments

An exact match is required, but the match is not case-sensitive (unless you specify **SET WAITCASE ON**); for example, "ABC" matches "AbC", but not "BCA". WAITFOR times out and allows processing to continue if the target string is not received in the specified time. The statement IF WAITFOR on a line following the WAITFOR command returns a "TRUE" if the target string was received; a "FALSE" is returned if a timeout occurred or the Esc key was pressed.

WAITFOR can be exited with the Esc key.

See also

WAITQUIET, SET WAITCASE, PAUSE and SUSPEND UNTIL.

WAITQUIET

Waits up to a specified time until the receive data line has been inactive for a number of seconds.

WAITQUIET [*integer*[*integer* | FOREVER]]

integer	The amount of time (in seconds) that the line must be inactive. The default for this optional field is 15 seconds.
integer | *FOREVER*	The amount of time (in seconds) that PROCOMM PLUS will wait to satisfy the first parameter (FOREVER forces the script to pause indefinitely until the line is quiet the required number of seconds). The default is for this optional field 30 seconds.

Example

waitquiet 20 forever	Pauses execution of the script indefinitely until the receive data line is inactive for 20 seconds.
waitquiet	If none of the optional parameters are used, WAITQUIET pauses execution of the script for up to 30 seconds while waiting for the receive data line to be inactive for 15 seconds.

Comments

WAITQUIET can be exited with the Esc key.

The **IF SUCCESS** and **IF FAILURE** statements allow you to test a WAITQUIET operation.

See also

WAITFOR, PAUSE and SUSPEND UNTIL.

WHEN

Forces an automatic response or procedure CALL when a particular target string is encountered.

Once a WHEN command is entered, it remains in effect until either a CWHEN command is encountered or the script terminates.

WHEN {*index string* | DISCONNECT} {TRANSMIT *string* | CALL *name*}

index	The identifying index number for this WHEN statement. Up to 3 WHEN commands can be in effect, with indexes ranging from 0 to 2.
string	The string used to trigger the action. Control characters can be included.
DISCONNECT	Indicates an action to take if carrier is lost or not present. DISCONNECT is automatically cleared as soon as it's triggered.
TRANSMIT string	Transmits the characters contained in the response string. Control characters can be included.
CALL name	CALLS the specified procedure, which cannot require any parameters.

Example

```
when 0 "more?" transmit "Y^M"
```

Sends a "Y" followed by a carriage return each time the prompt "MORE?" is received.

Comments

Up to 3 WHEN commands can be active at one time.

The SET WHENCASE command allows you to specify whether the comparison is case-sensitive.

The trigger string should be null-terminated. Only up to the first 40 characters of the trigger string are recognized.

See also

WAITFOR, TRANSMIT, CALL and SET WHENCASE.

WHILE

Repeats a series of commands until the condition becomes "FALSE".

WHILE *condition*

Example

```
while num < 100
.
.
.
endwhile
```

Performs a block of commands WHILE the integer variable NUM remains less than 100.

Comments

The valid conditional expressions for WHILE are the same as those available with the **IF** command, with the addition of the FOREVER option. The commands **LOOPWHILE** and **EXITWHILE** may also be used in a WHILE block.

FOREVER creates an infinite loop within a WHILE construct.

LOOPWHILE branches directly to the WHILE conditional test from anywhere within the command block.

EXITWHILE exits the WHILE loop and branches to the command following the **ENDWHILE** command.

See also

EXITWHILE, LOOPWHILE, ENDWHILE and IF.

WRITEC

Performs emulation conversion and displays or acts upon the provided character value. The cursor position is updated where appropriate.

WRITEC *integer*

Example

```
integer char=65
writec char
```

Converts the integer value in CHAR based on the present emulation and displays it at the current cursor position.

Comments

WRITEC interprets the keycode value represented by the integer based on the current terminal emulation. If the keycode corresponds to an emulation function, that function is performed; otherwise, WRITEC behaves exactly like TERMWRT. It displays the ASCII character corresponding to the keycode value and moves the cursor to the next available position.

The keycode value is treated as unsigned, and range checking is suppressed.

See also

TERMWRT, KEYGET, HITKEY, SET KEYS, KEY2ASCII and MESSAGE.

XOR

This command performs a bitwise comparison of two numbers and places the result in the specified numeric variable. For each two bits compared, the resulting variable is assigned the value 1 or 0 in the same corresponding bit position. A 1 is assigned if 1 bit is 1 and the other is 0. A zero is assigned if both bits are the same.

XOR *number number numvar*

Comments

XOR cannot be used with floating point numbers.

The *command* form "XOR N0 N1 N2" is equivalent to the *operator* form "N2=N0^N1".

See also

OR and AND.

ZERO

Compares a number with zero. The integer variable is assigned 1 if the number is zero, and 0 if the number is non-zero.

ZERO *number intvar*

Comments

ZERO and NOT function identically.

See also

NOT.

System Variables

For a general discussion of system variables and their uses, please refer to the beginning of this chapter.

$COL	The current column cursor position.
$DATE	The current system date (in the format mm/dd/yy).
$FATTR	The attributes of the last file returned by a **FINDFIRST** or **FINDNEXT** command.
$FDATE	The creation date of the last file returned by a **FINDFIRST** or **FINDNEXT** command (in the format mm/dd/yy).
$FEXT	The extension (up to three characters following the period in a filename) of the last file returned by a **FINDFIRST** or **FINDNEXT** command.
$FILENAME	The full filename (including extension) of the last file returned by a **FINDFIRST** or **FINDNEXT** command.
$FNAME	The base filename (excluding extension) of the last file returned by a **FINDFIRST** or FINDNEXT command.

$FSIZE	The size (in bytes) of the last file returned by a **FINDFIRST** or **FINDNEXT** command.
$FTIME	The creation time for the last file returned by a **FINDFIRST** or **FINDNEXT** command (in the format HH:MM).
$NULL	Always a "null" (or empty) character; this system variable is usually used to test another string variable for the null condition. An empty string ("") acts as a synonym for $NULL.
$ROW	The current row cursor position.
$SCRNCOLS	The number of rows in your current video mode.
$SCRNROWS	The number of rows in your current video mode.
$TERMBOLD	The color used for high-intensity characters in Terminal mode.
$TERMDIM	The color used for low-intensity characters in Terminal mode.
$TERMNORM	The color used for normal characters in Terminal mode.
$TERMREV	The color used for reverse characters in Terminal mode.
$TERMULINE	The color used for underlined characters in Terminal mode.
$TIME0	The current system time (in the format HH:MM:SSAM or HH:MM:SSPM).
$TIME1	The current system time (in 24-hour military format HH:MM:SS).

The following group of system variables allows you to test a session or operation for a specific condition. Unless otherwise indicated, each has two possible values: a 0 (indicating FALSE) and a 1 (indicating TRUE).

COMDATA	Returns the current status of available characters in the receive data buffer. If RXDATA is set OFF, available characters will be processed between script commands; therefore, it's possible that the status of COMDATA may change from "TRUE" to "FALSE" by the time your script actually uses it. The only method to ensure that COMDATA will remain "TRUE" is to **SET RXDATA ON**. There is no way to guarantee that COMDATA will remain "FALSE", since data can be received at any time.
CONNECTED	Indicates whether or not the Carrier Detect (CD) signal is high (indicating that PROCOMM PLUS is currently connected to another system). If your modem forces CD high, CONNECTED will always be "TRUE".
FAILURE	Indicates if the last testable ASPECT operation (like writing to a file or creating a directory) did **not** complete successfully.
FOREVER	This unique system variable will **always** return a value of 1, allowing you to easily create "infinite" **WHILE** loops (as in the statement **WHILE FOREVER**).
FOUND	Denotes that a matching filename has been found by a **FINDFIRST** or **FINDNEXT** statement. Additionally, FOUND can be used to indicate a match by a **FIND** statement.
FROMDDIR	Contains the Dialing Directory entry number which invoked the currently-executing script; otherwise, the value is 0.
HITKEY	Contains the keycode of the next available key; otherwise, the value is 0.
MONO	Indicates whether or not the local computer is using a monochrome monitor.
SUCCESS	Indicates if the last testable ASPECT operation completed successfully.
WAITFOR	Denotes whether or not the last WAITFOR statement received the specified string (and therefore completed) before timing out.

The following group of system variables allows you to read the information from a Dialing Directory entry. Any system variable with the "$D_" convention will contain the corresponding value from the last directory entry with a successful connection (otherwise, the contents are undefined). If you access the Dialing

Directory again, the contents of these variables will be changed. To access the information from a particular entry, use the **SET DIALENTRY** statement before using these variables.

$D_BAUD	The baud rate for this entry. Valid values are 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600 and 115200. A long value.
$D_DATABIT	The number of data bits. An integer value.
$D_DUPLEX	The duplex setting for this entry (either 0 for Full or 1 for Half). An integer value.
$D_ENTRY	The entry number (as displayed within the Dialing Directory). An integer value.
$D_KBDFILE	The keyboard file. A string value.
$D_LDATE	The last call date for this entry. A string value.
$D_METAKEY	The Meta key file for this entry. A string value.
$D_MODE	The connection type for this entry. An integer value; 0 indicates a modem connection, while 1 indicates a direct connection.
$D_NAME	The name identifying this entry. A string value.
$D_NOTE	The *.NOT filename for this entry (as used by the Jot function). A string value.
$D_PARITY	The parity for this entry (0 for None, 1 for Odd, 2 for Even, 3 for Mark and 4 for Space). An integer value.
$D_PHONE	The phone number for this entry. A string value.
$D_PORT	The COM port used by this entry (values range from 1 to 8). An integer value.
$D_PROTO	The protocol used by this entry. Values can range from 0 to 17 (for a listing of protocol keywords and their values, refer to the definition for **FETCH** in this chapter). An integer value.
$D_PWORD	The password for this entry. A string value.
$D_SCRIPT	The script name for this entry; a string value.
$D_STOPBIT	The stop bit setting (either 1 or 2). An integer value.

$D_TERM	The terminal emulation type used by this entry. Values can range from 0 to 32 (for a listing of emulation keywords and their values, refer to the definition for **FETCH** in this chapter). An integer value.
$D_TOTAL	The total number of successful connections for this entry. An integer value.

The following group of system variables are used with Host mode; any system variable with the "$H_" convention will provide information on the last caller who successfully logged on to your Host system.

$H_BAUD	This long variable contains the baud rate at which the last user connected to Host.
$H_ELAPSED	Total elapsed time of the last user's session. This is a string variable.
$H_LEVEL	The user's access level. Possible values are 0, 1, and 2; if the logon was unsuccessful (Host didn't allow the user into the system), this variable is set to -1. An integer value.
$H_NAME	The last user's name (in "firstname lastname" format). This is a string variable.
$H_OFFLINE	The time the last user disconnected (in 24-hour format). This is a string variable.
$H_ONLINE	The time the last user connected (in 24-hour format). This is a string variable.

CHAPTER

3

Compiling ASPECT

Overview

This chapter discusses the compile process (which we introduced in Chapter 1) in more detail.

For those users who are "upgrading" scripts from earlier revisions of PROCOMM PLUS, Chapter 3 will provide a wealth of valuable information. An example 1.1B source program will be **CONVERTed** and compiled—and then we'll demonstrate changes that you can make to your upgraded scripts to make them more efficient, easier to understand and easier to maintain.

ASPCOMP Technical Notes

The Compiling Process

For those interested in the inner workings of ASPCOMP, let's take a moment to describe both *passes* make by the compiler in detail:

- ***Pass 1—Checking for Errors.*** The compiler makes a first pass through your source program, specifically checking for errors *before* the compiled code is generated. Pass 1 includes a thorough check of the source program's syntax. Errors are reported as they're discovered, and warning messages may be displayed concerning potential problems (depending on the **/W** option setting). If the number of errors specified by the **/E** option are found in pass 1, ASPCOMP will abort the compile process (without performing pass 2, where the output file is created).
- ***Pass 2—Compiling the Script.*** If the code is validated by the first pass, the compiler begins the "tokenizing" process. Commands words and keywords are converted to tokens, and specific file offsets are generated for each conditional statement—these offsets greatly enhance the speed at which procedure CALLs and label jumps are performed within your compiled script. For security, constants are converted to shorter groups of encrypted characters.
- ***Setting the Exit Code.*** As ASPCOMP terminates, it generates an "exit code" that can be tested in a DOS batch file or parent process. If the script compiled successfully, ASPCOMP returns a "0"; if an error occurred during compilation, a "-1" is returned.

Using a Symbol Map and Line Reference

ASPCOMP includes two features that can provide you with valuable information as you're writing and debugging scripts. As we mentioned in Chapter 1, the **/M** option generates a source file summary, while the **L** switch (used in conjunction with **/M**) creates an additional line reference table; both sets of data are written to the same file, which carries an extension of "*.MAP*". Let's examine the output you'll receive when you specify these options during a compilation.

Note that a .MAP file is only created if the compile completes successfully!

The Line Reference Table (invoked by the **L** switch) consists of two columns labelled "Line" and "Offset". Each *line* of the source program that generates compiled code carries a corresponding beginning *offset* value that ASPECT displays as part of its run-time error messages (if you **SET ASPDEBUG ON**)—this allows you to trace problems that occur within your compiled script back to the line (or lines) of source code causing the error. Comment blocks are ignored.

The Output File Summary (invoked by the **/M** option) includes similar information for each procedure, including:

- its size (in bytes).
- its beginning offset.
- the number of labels (with their names).
- the number of parameters (with their names).
- the number of local variables.
- the number of times the procedure was **CALL**ed.
- the number of times it **CALL**ed other procedures (with their names).

The summary also displays totals for the entire script.

Upgrading Scripts from PROCOMM PLUS 1.x

Using CONVERT

The **CONVERT** utility is designed to make the switch from PROCOMM PLUS 1.*x* standalone script files to compiled "*.ASX*" scripts as easy as possible. **CONVERT** can modify most of your present ASPECT scripts with the "basics" required for a successful compile.

If the script you wish to convert is in the current directory, you can run **CONVERT** from the DOS prompt:

- Type *CONVERT [options] scriptname* and press Enter.

The options are:

/C	Suppresses the **CONVERT** header and comments normally placed in the output file.
/En	Determines the maximum errors to allow for the conversion; if more than *n* errors are found by **CONVERT**, it will abort automatically. The default is 20 errors.

/T — Provides *terse* error messages. Specifying this option will substitute one-sentence error messages for the default detailed text (with probable solutions). If you're familiar with CONVERT, this option can save time.

As an example of the changes CONVERT will make to your existing script files, we'll convert the 1.1B script "HOSTXFER.ASP" and compile it.

HOSTXFER.ASP automates a session with a remote system running PROCOMM PLUS Host; it logs on, prompts the user for the names of files to send and get and logs off. The complete 1.1B script file looks like this:

```
;*
;* HOSTXFER.ASP (C) 1989 DATASTORM TECHNOLOGIES, INC.
;*
;* YOU'LL NOTE SOME PROMPTS FROM THE HOST ARE SENT TWICE.  THIS
;* ENSURES THAT THE TIMING BETWEEN SYSTEMS IS CORRECT.
;*
;*
;* STEP 1 - SET USER VARIABLES AND DIAL
;*
ASSIGN S0 "JOHN PUBLIC"        ;* INSERT YOUR NAME HERE
ASSIGN S1 "JOHN"               ;* AND YOUR PASSWORD HERE
WHEN "-MORE-" "^M"
SET TXPACE 50
IF NOT LINKED
   DIAL "1"                    ;* SET THIS TO THE PROPER DIR ENTRY
ENDIF
;*
;* STEP 2 - LOGS YOU INTO THE PROCOMM PLUS HOST COMPUTER.
;*
PAUSE 3
GETNAME0:
TRANSMIT "^M"
WAITFOR "NAME:"
IF NOT WAITFOR
   GOTO GETNAME0
ENDIF
TRANSMIT S0
TRANSMIT "^M"
PAUSE 1
TRANSMIT "Y"
PAUSE 3
SENDPASS:
```

```
TRANSMIT S1
TRANSMIT "^M"
WAITFOR "VERIFY:" 5
IF WAITFOR
   PAUSE 1                    ;* RESENDS YOUR PASSWORD
   GOTO SENDPASS              ;* IF YOU'RE A NEW USER
ENDIF
TRANSMIT "^M"
;*
;* STEP 3 - PROMPTS FOR THE FILENAME TO UPLOAD
;*
GETNAME1:
BOX 0 0 4 67 15
ATSAY 1 3 7 "ENTER THE FILE TO 'UPLOAD' (SEND TO HOST), OR"
ATSAY 2 5 7 "HIT JUST <ENTER> TO SKIP THIS STEP:"
ATSAY 3 3 7 ">"
ATSAY 3 64 7 "<"
ATGET 3 4 15 60 S2
SWITCH S2
   CASE "_NULL"
      CLEAR
      GOTO GETNAME2
   ENDCASE
ENDSWITCH
ISFILE S2
IF FAILURE
   SCROLL 0 2 2 3 49 7
   ATSAY 2 3 15 "FILE NOT FOUND!"
   ALARM 1
   PAUSE 1
   GOTO GETNAME1
ENDIF
CLEAR
;*
;* STEP 4 - PERFORMS THE UPLOAD USING YMODEM BATCH
;*
PAUSE 1
TRANSMIT "U"                  ; SELECT UPLOAD FUNCTION
PAUSE 1
TRANSMIT "Y"                  ; SELECT YMODEM BATCH
WAITFOR "NAME?"
PAUSE 1
TRANSMIT S2                   ; SEND FILENAME
TRANSMIT "^M"
PAUSE 1
```

```
TRANSMIT "^M"                    ; ENTER FOR DESCRIPTION
PAUSE 3
SENDFILE BYMODEM S2              ; SEND THE FILE!
PAUSE 3
;*
;* STEP 5 - PROMPTS FOR THE FILENAME TO DOWNLOAD
;*
GETNAME2:
BOX 0 0 4 67 15
ATSAY 1 3 7 "ENTER THE FILE TO 'DOWNLOAD' (RECEIVE FROM HOST) OR"
ATSAY 2 5 7 "HIT JUST <ENTER> TO SKIP THIS STEP:"
ATSAY 3 3 7 ">"
ATSAY 3 64 7 "<"
ATGET 3 4 15 60 S2
SWITCH S2
   CASE "_NULL"
      CLEAR
      GOTO TRYAGAIN
   ENDCASE
ENDSWITCH
CLEAR
;*
;* STEP 6 - TELL HOST TO SEND FILE, ENABLE RECEIVE
;*
TRANSMIT "^M"
PAUSE 1
TRANSMIT "D"
PAUSE 1
TRANSMIT "Y"
PAUSE 1
TRANSMIT S2
TRANSMIT "^M"
WAITFOR "PROCEDURE" 10
IF WAITFOR
   GETFILE BYMODEM
   PAUSE 3
ELSE
   MESSAGE "FILE NOT FOUND!"
   ALARM 2
   GOTO GETNAME2
ENDIF
;*
;* STEP 7 - ASKS WHETHER TO TRANSFER MORE FILES
;*
TRYAGAIN:
```

```
BOX 0 0 2 34 15
ATSAY 1 3 7 "TRANSFER MORE FILES? (Y/N)"
ATSAY 1 30 15 "NO"
CURON
LOCATE 1 30
KEYGET S3
SWITCH S3
   CASE "Y"
      GOTO GETNAME1
   ENDCASE
ENDSWITCH
CLEAR
;*
;* STEP 8 - LOGS YOU OFF!
;*
BIYA:
TRANSMIT "^M"
WAITFOR "CHOICE?"
IF NOT WAITFOR
   GOTO BIYA
ENDIF
TRANSMIT "G"
```

To convert HOSTXFER.ASP:

- Copy it into the same directory as **CONVERT**.
- Type *CONVERT /C HOSTXFER* and press Enter.

Note that we used the **/C** option. By default, **CONVERT** adds a commented "header" to the output source program; since the identifying information wasn't required for this example, we ommitted it with the **/C** option.

CONVERT will make the following changes to the 1.1B **ASPECT** script file:

- The **PROC MAIN** and **ENDPROC** statments are added to the beginning and end of the script, respectively. **Remember, every ASPECT source program must contain a procedure named MAIN!**
- The **WHEN** statement in Step 1 becomes

  ```
  WHEN 0 "-MORE-" TRANSMIT "^M"
  ```

  Since up to 3 **WHEN** commands can now be active, **CONVERT** supplies the index "0". The **TRANSMIT** parameter (which was not required in earlier versions of **ASPECT**) is also added.

- The CASE "_NULL" conditions used by the SWITCH commands in Steps 3 and 5 are replaced with CASE $NULL. The new system variable $NULL can appear anywhere in your script where a string constant can appear.

- The KEYGET in Step 7 is replaced by the statements

```
KEYGET AUX_INT_1
KEY2ASCII AUX_INT_1 S3
```

  The global integer variable AUX_INT_1 (declared at the top of the script) holds the value for the key pressed by the user. Since KEYGET now supports only integer variables, the KEY2ASCII command must be used to convert the key's integer value into the string variable S3.

Now that CONVERT has performed the basics necessary to compile HOSTXFER.ASP, it can be used with PROCOMM PLUS—however, since CONVERT cannot rewrite a script, HOSTXFER.ASP isn't as efficient, clear or easily-modified as it could be!

Writing Scripts for Efficiency

There are a number of general rules you can follow to make your compiled scripts clearer and more efficient:

- ***Use a Structured Approach.*** Previous versions of ASPECT favored programs written to "flow" from beginning to end (a style usually called "unstructured"). Compiled ASPECT is a structured, procedure-oriented language.

- ***Avoid GOTOs.*** Instead of performing a GOTO, CALL a procedure. Procedures can contain more than one process; the current function of such a "multi-purpose" procedure can be determined through the value of a flag or passed parameter.

- ***Use Full Variable Names.*** Although the S*x* and N*x* predefined variables still exist (and will still work), a script becomes much easier to read, understand and maintain if you define your own names for variables. The predefined variables must be used, however, if you need to preserve the value of a variable when chaining to another script with the EXECUTE command.

- ***Avoid Multiple Exits.*** Ideally, only a single "exit" point (where the user ends program execution) should be included within a script. Multiple exits make a source program harder to read and can cause errors.

- ***Use Local Variables.*** Global variables are indeed easy-to-use, but they can also create problems. By localizing variables, you avoid accidental changes to the same variable in other areas of the program—and the time it takes to track down such bugs!

- *Use Staggered Code Indentation.* Logical indentation in a program can dramatically increase its clarity, allowing you to easily note the beginning and ending points for procedures and conditional blocks. Since ASPCOMP ignores indentation and comments in your source programs, it's recommended that you both comment and indent your code heavily.

In the last section, we used CONVERT to upgrade the sample script HOSTXFER.ASP. Let's take HOSTXFER.ASP one step further:

- First, let's move each routine into a separate procedure; for our example, we'll use the LABEL names as the new procedure names.
- To make the source more readable, we'll use lowercase throughout the script and logically indent each conditional block.
- As mentioned earlier, the predefined variables S0 and S1 could have been left intact; for clarity, however, we'll change them to user-defined string variables.
- The MAIN procedure will now hold a structured loop, executing the procedures GETNAME1, GETNAME2, TRYAGAIN and BIYA.
- The procedure TRYAGAIN will now use an integer parameter (passed by reference) called "ACTION"; this value is returned to the MAIN procedure as a decision flag.

Our rewritten HOSTXFR2.ASP looks like this:

```
;*
;* hostxfr2.asp (c) 1990 datastorm technologies, inc.
;* You'll note that some prompts from the host are sent twice.  This
;* ensures that the timing between systems is correct.
;*
define TRUE 1
define FALSE 0
define NAME     "Chip Tawdry"
define PASSWORD  "BigB"

proc main

   integer action                ;Selection variable
   action =  TRUE                ;At least once through the loop.
   ;*
   ;*  INITIALIZATION
   ;*
   when 0 "-more-" transmit "^m"
```

```
   when 1 "any key to" transmit "^M"
   set txpace 100
   call logtohost                    ;Call login routine
   clear
   while (action == TRUE)            ;Until user chooses not to
      call getname1                  ;Call upload routine
      call getname2                  ;Then call download routine
      call tryagain with &action     ;Then call confirm
   endwhile
   call biya                         ;Call logoff routine.
endproc
   ;*
   ;*  LOGS YOU INTO THE PROCOMM PLUS HOST COMPUTER.
   ;*

proc logtohost
   if not connected                  ;if CD low
      if not fromddir                ;And not called by dialer
        dial "1"                     ;dial this entry
      endif
   endif                             ;answer logon prompts
   pause 1
   transmit "^m"
   waitfor "name:"
   transmit NAME
   transmit "^M"
   pause 1
   transmit "y"
   waitfor "password"
   transmit PASSWORD
   transmit "^M"
   waitfor "verify:" 2
   if waitfor
      transmit PASSWORD
      transmit "^M"
   endif
   transmit "^M"
endproc
;*
;*     PROMPTS FOR THE FILENAME TO UPLOAD
;*

proc getname1

   string filewanted                 ;File desired for download
```

```
  while forever
    filewanted = ""                ;Clear old value of filewanted
    box 0 0 4 50 15
    atsay 1 3 7 "Enter the file to UPLOAD (send to host),"
    atsay 2 3 7 "or hit just <enter> to skip this step:"
    atsay 3 18 7 ">"
    atsay 3 31 7 "<"
    atget 3 19 15 12 filewanted
    if null filewanted             ;did we hit just <enter>?
      exitwhile                    ;if we did then get out
    endif
    isfile filewanted              ;check existence of the file
    if failure                     ;If found, then do_send proc
      scroll 0 3 2 3 47 7          ;is CALLed, else retry
      curoff
      atsay 3 18 15 "File not found!"
      alarm 1
      pause 1
      curon
    else
      clear
      call do_send with filewanted   ;send the file
    endif
    clear
  endwhile
  clear
endproc
;*
;*    PROMPTS FOR THE FILENAME TO DOWNLOAD
;*

proc getname2

  string filewanted
  while forever
    box 0 0 4 50 15
    atsay 1 3 7 "Enter a file to DOWNLOAD (receive from host)"
    atsay 2 3 7 "or hit just <enter> to skip this step:"
    atsay 3 18 7 ">"
    atsay 3 31 7 "<"
    atget 3 19 15 12 filewanted
    strcmp filewanted $null        ;If the user inputs a null string
    if failure                     ;no download attempted
      clear
```

```
        call do_receive with filewanted
        clear
      else
        exitwhile
      endif
    endwhile
    clear
endproc
;*
;*    PERFORMS THE UPLOAD USING ZMODEM
;*

proc do_send

    strparm filewanted

    transmit "u"                    ;select upload function
    pause 1
    transmit "Z"                    ;select Zmodem
    pause 1
    transmit filewanted             ;send filename
    transmit "^m"
    sendfile zmodem filewanted      ;send the file!
endproc
;*
;*    RECEIVES THE FILE FROM THE HOST
;*

proc do_receive

    strparm filewanted

    transmit "d"                    ;Select download
    pause 1
    transmit "Z"                    ;Select ZMODEM protocol
    pause 1
    transmit filewanted             ;Tell it the filename
    transmit "^m"
    waitfor "procedure" 5
    if waitfor
       getfile zmodem               ;NO getfile zmodem necessary
     else                           ;as long as zmodem auto-download
       transmit "^M"                ;is enabled!!!
    endif
 endproc
```

```
;*
;*    ASKS WHETHER TO TRANSFER MORE FILES
;*

proc tryagain

   intparm userchoice

   integer userkey
   box 0 0 2 34 15
   atsay 1 3 7 "transfer more files? (Y/n)"
   locate 1 30
   atsay 1 30 15 "no"
   curon
   keyget userkey                    ;Get the keypress
   userchoice = FALSE
   if userkey == 'y' || userkey == 'Y'
      userchoice = TRUE
   endif
   clear
endproc
;*
;*    LOGS YOU OFF!
;*

proc biya

   transmit "^m"
   pause 1
   transmit "g"
endproc
```

To the user, the "flow" of HOSTXFR2.ASP remains the same (it now repeats, however, until the user explicitly chooses to stop). To the programmer, however, HOSTXFR2.ASP has been greatly improved! It's easier to read and understand, the conditional processing it performs is more efficient—and future modifications will be much easier to perform.

In the next chapter, "Common ASPECT Questions", we'll review topics often covered by our Technical Support staff.

CHAPTER

4

Common ASPECT Questions

Overview

A full-featured and powerful language like ASPECT often elicits questions from both novice and professional programmers alike—perhaps you're looking for a more efficient way to perform a function, or your script unexpectedly pauses during execution.

If you're experiencing a problem with an ASPECT script, check this chapter first! The following items have been identified as the most common questions asked by users calling our Technical Support staff.

Additionally, we'll also include tips on programming "shortcuts" that may save you time and keystrokes.

ASPECT Variables

How can I initialize a string variable to a null value?

ASPECT supports the *direct assignment* of a value to each data type. To initialize to null (or empty), use the read-only system variable "$NULL"—for example:

```
ANYVAR = $NULL
```

Here we have assigned the string variable ANYVAR a null value.

How can I make variables global or local?

To automatically make a variable *global*, simply declare it outside of any procedure. Note that globals need not be placed at the top of the source program (they are, however, easier to locate if you do declare them at the top).

Local variables are defined from within a procedure. Remember, local variables only operate within their "home" procedure; don't make the mistake of attempting to use a local from another procedure!

```
LONG TOTAL = 170133
PROC MAIN
  STRING BILLNAME
ENDPROC
```

In this example, the long variable TOTAL is a global variable initialized to 170133 (since it occurs outside of the MAIN procedure) , while we've defined BILLNAME as a local string variable.

How can I change a single character in a string?

The STRPOKE command comes in handy here—try this method:

```
string="FILE0001.TXT"  ;before
strpoke string 7 '2'
string="FILE0002.TXT"  ;after
```

Note the use of the character constant '2' in the STRPOKE statement.

Why can't I compare two strings with the statement "IF string1 == string2"?

Strings **must** be compared with the STRCMP command. The "==" equality operator can only be used with numeric variables or constants.

The SUCCESS or FAILURE system variables can be tested to determine if the strings are identical.

Why is ASPECT returning the wrong value for my integer variable assignments?

Check your value ranges for numeric variables; chances are that your values are larger than 32767 or smaller than -32768 (making them LONG values instead of integer). Declare them as LONG variables.

Is your value a fractional amount? ASPECT assumes that any value containing a decimal be defined as a FLOAT type (and should be assigned as a float variable to retain the fractional amount).

How do passed parameters work? Why are the values I pass to a procedure never modified when I return to the CALLing procedure?

You can instruct ASPECT to pass parameters by *reference* (any modifications made to the variable by the CALLed procedure are permanent) or by *value* (where the CALL sends only the value of the variable). To CALL by reference, prefix the variable name in the CALL statement with an ampersand ("&").

To illustrate passing parameters by reference, let's use the example procedure READ_SEC:

```
PROC READ_SEC
  INTPARM X
  STRING TSTR
    TIME TSTR 0
    SUBSTR TSTR TSTR 6 2
    ATOI TSTR X
ENDPROC
```

You'll note that READ_SEC "extracts" the seconds from the 12-hour format system time and then converts those seconds to an integer; the result is placed in an integer parameter named X. If we CALL READ_SEC with the statement:

```
INTEGER SEC
CALL READ_SEC WITH &SEC
```

the variable parameter name &SEC indicates that the value is to be returned intact from the CALLed procedure.

To pass variable parameters in a CALL without allowing their value to change, simply omit the ampersand prefix.

For more information on passing parameters, please refer to the definitions for CALL and PROC in Chapter 2 of this manual.

Why can't I use variables like "$amount" and "amount$"?

The "$" is reserved for ASPECT system variables, and should never appear in a user-defined variable name. Also, user-defined variables must begin with an alphabetic character (in either UPPER or lower case) or an underscore.

Does ASPECT support operations like "IF VAR < 5 AND VAR > 6..."?

Yes, ASPECT does support the AND operator; however, the example would return the error message "UNEXPECTED COMMAND" in a script. AND is a bitwise ASPECT command—in the above example, you would instead use the logical operator "&&". The example would correctly read:

```
IF VAR < 5 && VAR > 0...
```

File Input/Output

I've opened a file as Read/Write—why isn't it working properly?

Although an FOPEN command can access a file in Read/Write mode, you must perform an FSEEK or REWIND on the file before switching between read and write operations.

In this example, we both read and write to the same file:

```
long pointer
proc main
   string stuff
   fopen 0 "io.dat" "WT+"
   fwrite 0 "1234567890" 10
   ftell 0 pointer
   fseek 0 pointer 0
   fread 0 stuff 10
endproc
```

Note that this example uses a text file; when using the FSEEK command, you must seek with an offset of 0 relative to the origin (or, as we did here, you can seek from the file's beginning with an offset from an FTELL command).

Where possible, it's suggested that a file be opened in a single mode (Read-Only or Write-Only, for example); this can reduce the risk of errors.

Why does the statement "IF EOF 0" always seem to return FALSE as I write to a file?

The EOF condition becomes TRUE after the first operation that attempts to **read** past the end of a file. It will remain TRUE until an FSEEK, FCLEAR or REWIND is performed.

I'm working with a fixed-length file with records 80 characters long. Does ASPECT allow me to directly access each record?

Yes—rather than read each record sequentially, use the **FSEEK** command to speed file access!

To illustrate, this example reads record 12 of that file directly:

```
proc main
   define recordlength 80
   integer total
   long position
   record=12
   total=recordlength+2          ;add CR/LF
   record=record-1               ;start at 0
   position=record*total         ;position of record
   fseek 0 position 0            ;point past start of file
   fread 0 line recordlength     ;read the line
endproc
```

Note that the first record in this file is considered record 0.

Display

Why does VIDREST work fine once, and then fail?

If a screen has been **VIDSAVE**d, it can only be restored **once**. If you need to restore a screen multiple times, simply perform another **VIDSAVE** and re-save the screen after each restore. Since you can **VIDSAVE** up to three screens, you could also save two additional "copies" of the same screen and restore them separately.

Connecting to a Remote System

My logon script works erratically with Telenet and Tymnet. How can I fix it?

Large switching networks like Tymnet and Telenet are sometimes slow when processing your sign-on data. Until you've connected to your destination service (like CompuServe), try slowing your system down slightly by SETting TXPACE to 100 or higher. Alternately, you can insert pause characters in your script's TRANSMIT statements—the default pause character is a tilde (~).

Why doesn't this BBS recognize succeeding carriage returns in my scripts?

It may be necessary to send carriage returns separately instead of one immediately after another; unless the remote system can buffer the incoming data, the second carriage return may be "lost". Instead of sending the string

```
TRANSMIT "^M^M"
```

try sending the string

```
TRANSMIT "^M"
PAUSE 1
TRANSMIT "^M"
```

As we mentioned earlier, it's always a good idea to keep the processing speed of the remote computer in mind when writing scripts.

What changes do I need to make to logon scripts created with Record mode?

First, shorten the WAITFOR strings to the final word (or characters) in each prompt. For example, the statement

```
WAITFOR "^M^JNAME?^M"
```

becomes

```
WAITFOR "NAME?"
```

Next, check to see if you need to add a PAUSE statement at the top of your script:

```
TRANSMIT "^M"
WAITFOR "NAME?"
```

In this example, your system may miss the **WAITFOR** string; to make sure that PROCOMM PLUS is prepared for incoming data, add a **PAUSE** statement:

```
PAUSE 1
TRANSMIT "^M"
WAITFOR "NAME?"
```

Finally, make sure that you trim extra statements from the beginning and end of your script.

Accessing DOS

I'm trying to read the return value from DOS using the DOS command... but ASPECT always returns SUCCESS, even though the program failed!

When you use the **DOS** command, ASPECT always tests the return value of COMMAND.COM, not the DOS program you ran. Unless PROCOMM PLUS can't locate or use COMMAND.COM (COMMAND.COM isn't in your current directory, DOS PATH or defined through the COMSPEC environment variable), a **DOS** statement will always return a SUCCESS condition.

If you require a return value, use the **RUN** command whenever possible; a **RUN** statement will return the SUCCESS or FAILURE of the program, **not** COMMAND.COM.

I'm running PROCOMM PLUS on a floppy system. Why do I keep getting errors when I attempt to run a DOS program?

ASPECT must load a second copy of COMMAND.COM to execute an external program with **DOS** or **SHELL**. The value of your "COMSPEC" environment variable points to the location of COMMAND.COM (you can type SET and press Enter at the DOS command prompt to display the value of COMSPEC).

Problems occur on a floppy-based system if you insert a different disk in your boot drive while PROCOMM PLUS is executing; make sure that COMMAND.COM is available within the path specified by COMSPEC.

Transferring Files

Why do my SENDFILE and GETFILE operations abort or time out?

First, make sure that a file transfer operation works manually (this may point out a problem with your modem setup, an invalid filename or a protocol mismatch). If the transfers work manually, you may have a timing problem—most file transfer protocols require fairly close cooperation between the sending and receiving computers. It's a good idea to "fine-tune" a script performing file transfers by inserting messages and pauses where necessary.

My files transfer fine, but the WAITFOR immediately following the transfer is almost always ignored. What's going on?

Depending on the processing speed of the remote computer, you may discover that the final "handshaking" required by some protocols (for example, YMODEM and ZMODEM) completes earlier on the remote system. Thus the characters that trigger the **WAITFOR** are sent before PROCOMM PLUS is ready for them, and the **WAITFOR** is never satisfied.

A good example of this is a BBS that sends out a prompt immediately after the transfer completes. In this case, a solution might be to send a character or carriage return to force the BBS to resend the prompt (or to simply assume that the prompt has been sent, and proceed anyway).

Debugging

Without a TRACE mode, how can I debug my scripts?

Since all syntax errors will be found by **ASPCOMP** during compilation, run-time errors will require debugging. Use the "/ML" options with **ASPCOMP**—this will create a symbol map, along with additional procedure execution and source line reference information.

Also, make sure you **SET ASPDEBUG ON** to display the offset error messages generated by ASPECT during execution.

With this information, you should be able to debug scripts accurately and quickly. Use the offset to locate the source line where the error occurred. Note that the error offset may refer to a location *between* two source offsets—this means that the error occurred somewhere in the middle of the source line with the lesser offset.

How can I debug timing and sequence problems?

If a particular section of your script is causing errors, a few well-placed **MESSAGE** or **FATSAY** display statements can help pin down the problem. Use them to display the current value of a variable, the name of the current procedure or to indicate the command currently being executed. Don't forget to remove these debugging statements after you've solved the problem!

In the next chapter, "Advanced ASPECT Examples", we'll discuss advanced ASPECT topics and review examples of Host mode "external" processing, file manipulation and remote commands.

CHAPTER

5

Advanced ASPECT Examples

Overview

In this chapter, we'll present two examples of advanced ASPECT script programming.

These sample source programs present procedures commonly found in two popular ASPECT applications—remote commands and external Host mode processing.

These scripts also use several of the "less-familiar" ASPECT command words. Although they're not required in many situations, these commands can save many lines of code (or perform specialized functions too complex for other languages).

Please note that:

- these examples are written for the expert ASPECT programmer—**you should be comfortable with writing in ASPECT before attempting to modify these script procedures.**

- for your convenience, these examples are available as .ASP files on the DATASTORM CompuServe Support forum and the DATASTORM BBS; additionally, some of them may be available on your distribution diskettes.

Remote Commands and String Manipulation

First, we present REMOTE.ASP, which demonstrates the usefulness of remote commands and the power of string manipulation. Although short, REMOTE.ASP allows a remote computer (also using PROCOMM PLUS) to compile and load a script file that you've written on your PC! This provides an alternative to sending several remote commands separately, where each command would require separate compilation before execution.

The file you send can be in source or compiled form; the name of the file to send should be substituted in the **DEFINE REMFILE** statement.

Note *The remote machine must have the Remote Commands Setup option ON for REMOTE.ASP to work!*

```
; REMOTE.ASP
; COPYRIGHT (C) 1990 DATASTORM TECHNOLOGIES, INC.

define REMFILE "$ASP_TMP"          ; filename received by remote

proc main
```

```
string filename,xmitfile,xmitstr
integer extndx
vidsave 0
box 4 18 18 60 14
atsay 5 24 15 "REMOTE SCRIPT EXECUTION FACILITY"
atsay 6 19 14 "----------------------------------"
atsay 8 20 15 "Enter the ASPECT filename:"
atget 8 47 112 12 filename
isfile filename
if failure
   sound 220 10
   atsay 10 20 15 "File not found..."
   pause 3
else
   find filename "." extndx
   substr xmitfile filename extndx 4
   strupr xmitfile
   strfmt xmitfile "%s%s" REMFILE xmitfile
   atsay 10 20 15 "Sending script file to remote..."
   strfmt xmitstr "^Dgetfile xmodem '"%s'"^M" xmitfile
   transmit xmitstr
   atsay 12 20 15 "Pausing for remote compilation..."
   pause 5
   sendfile xmodem filename
   if failure
      sound 220 20
      atsay 14 20 15 "Error sending file to remote"
      pause 3
   else
      pause 3
      atsay 14 20 15 "Sending EXECUTE command sequence..."
      strfmt xmitstr "^Dexecute '"%s'"^M" xmitfile
      transmit xmitstr
      pause 3
      sound 440 20
      atsay 16 20 15 "Operation completed -- Press any key"
      keyget
   endif
endif
vidrest 0
endproc
```

Host Mode External Processing and File I/O

Next, we present a **section** of HOST.ASP, which performs external Host mode processing; note that this is a partial script, and will not function as presented here! The full version of HOST.ASP (with complete comments and description) is available through the same sources listed at the beginning of this chapter. HOST.ASP allows your Host system to selectively call back a user (to reverse the

telephone charges). It also sets up Host with a custom configuration each time you run it.

```
define HOSTUSRFILE    "C:\PCPLUS\PCPLUS.USR"
define HOSTDLDIR      "C:\PCPLUS\HOST_DN\"
define HOSTULDIR      "C:\PCPLUS\HOST_UP\"

define HOSTCDXFER     YES
define HOSTCONTYP     MODEM
define HOSTHFLOW      OFF
define HOSTMAXDIAL    3
define HOSTNEWUSR     0
define HOSTPORT       COM1
define HOSTREMCMD     OFF
define HOSTSFLOW      OFF
define HOSTSHELCD     ON
define HOSTSYSTYP     CLOSED
define HOSTTIMOUT     5
define HOSTUSEDTR     YES
define HOSTWELCOM     ""
define FALSE          0
define TRUE           1

; GLOBAL DATA

string urec,uname,ufirst,ulast,upassword,uaccess,ucomment,ucbnumber
integer tempkey

; Function: MAIN

proc Main

  call Setup                          ; Setup port, modem, and variables

  while forever
       host                           ; Execute host mode
       if not success                 ; If ESC key or Abort command
          exitwhile                   ;  EXIT loop and script
       endif
       if null $H_NAME                ; If no successful login
          loopwhile                   ;  loop to restart host
       endif
       uname = $H_NAME                ; uname = "firstname lastname"
       call ParseUsrRec               ; Find and parse user record
       if success                     ; If found and parsed:
          call CallBackRights         ;  Does user has callback rights
          if success                  ;  If he does:
             call WantsCB             ;     See if he wants callback
             if success               ;     If he does:
                call CallBack         ;        Call him back
             endif
             loopwhile                ; Loop to reenter host mode
          endif
```

```
      endif
      HOSTHANGUP                    ; If not parsed or no rights, hangup
  endwhile

endproc

; Function: Setup
; Purpose: Initialize PROCOMM PLUS 2.0 for use as a BBS

proc Setup
  call SetupPort
  call SetupVars
  call SetupModem
endproc

; Function: SetupPort
; Purpose: Initialize the communications port

proc SetupPort
  set port      HOSTPORT
  set baud      HOSTBAUD
  set parity    NONE
  set databits 8
  set stopbits 1
endproc

; Function: SetupVars
; Purpose: Initialize system variables

proc SetupVars
                                    ; Make sure both mutually exclusive
  fopen 1 HOSTUSRFILE "rt"          ; Try to open user file
  if not success                    ; If unable, abort with message
    BOXMSG "Error opening user file."
    exit
  endif
  fclose 1

  set host autobaud   HOSTAUTOBD ; Setup HOST variables using their
  set host connection HOSTCONTYP ; definitions at the top of file
  set host dldir      HOSTDLDIR
  set host message    HOSTWELCOM
  set host newuserlvl HOSTNEWUSR
  set host systype    HOSTSYSTYP
  set host timeout    HOSTTIMOUT
  set host uldir      HOSTULDIR
  set host shellboot  HOSTSHELCD

  set callog          HOSTCALLOG ; Setup miscellaneous variables using
  set cdinxfer        HOSTCDXFER ; their definitions at the top
  set dropdtr         HOSTUSEDTR
  set hardflow        HOSTHFLOW
  set modem maxdial HOSTMAXDIAL
  set remotecmd       HOSTREMCMD
```

```
  set softflow          HOSTSFLOW

  set host goodbye      EXIT            ; callback mode

  set keys              ON
  set rxdata            ON
  set msg_crlf          OFF
  set switchcase        OFF

  set kermit blockcheck     3           ; Setup protocols to be good hosts
  set kermit filetype       BINARY
  set kermit packsize       1024
  set zmodem errdetect      CRC32
  set zmodem recvcrash      PROTECT
  set zmodem sendcrash      NEGOTIATE
  set zmodem timestamp      OFF
  set zmodem txmethod       STREAMING
endproc

; Function: SetupModem
; Purpose: Initialize the modem for use by host mode

proc SetupModem
  integer savetxpace
  fetch txpace savetxpace
  set txpace 150
  TX "ATZ^M"                        ; reset to defaults
  mspause 500
  TX "ATV1^M~"                      ; use verbal result codes
  mspause 500
  TX "ATQ0^M~"                      ; use verbal result codes
  mspause 500
  TX "ATS7=60^M"                    ; wait 60 seconds for CD
  mspause 500
  set txpace savetxpace
endproc

; Function: CallBack
; Purpose: Hangup and dial the callback number.

proc CallBack
  string title
  if connected
     HOSTPUTS "'r'n'r'n Hangup now! Be sure your modem is set to answer."
     HOSTPUTS "'r'n You will be called back momentarily....'r'n'r'n"
     HOSTHANGUP
     if connected
        return
     endif
  endif
  strfmt title "Calling: %s" uname
  mdial ucbnumber title             ; Dial number in user record
endproc
```

```
; Function: CallBackRights
; Purpose: Check .USR comment field for special callBACK string

proc CallBackRights
  integer idx
  find ucomment "CALLBACK[" idx    ; Search comment for keyword
  if found
    idx = idx + 9                  ; idx -> 1st part of number
    substr ucbnumber ucomment idx 80   ; Copy from number to end of line
    find ucbnumber "]" idx         ; Find terminator
    if found                       ; If found
      strpoke ucbnumber idx 0      ;  strip the rest of line
      SETSUCCESS                   ;  and return success
      return
    endif
  endif
  SETFAILURE
endproc

; Function: WantsCB
; Purpose: Ask user if he wants to be called back.

proc WantsCB
  string response
  HOSTPUTS "'r'n'r'n Would you like to be called back at "
  HOSTPUTS ucbnumber
  HOSTPUTS "?'r'n"
  HOSTPUTS " Your choice (Y=Yes, H=Hangup now, C=cancel)? "

  while forever                    ; Loop until we break out
    HOSTGETC &response             ; Get a character
    if not success                 ; (If connection lost,
      exitwhile                    ;  break with FAILURE set)
    endif
    SETFAILURE                     ; Assume not calling back
    switch response                ; What do you want user?
      case "Y"                     ; If 'Y'
        SETSUCCESS                 ;  change assumption
        exitwhile                  ;  and break
        endcase
      case "H"                     ; If 'H'
        HOSTGOODBYE                ;  Say goodbye and hangup
        exitwhile                  ;  break
        endcase
      case "C"                     ; If 'C' or dropped thru from 'H'
        exitwhile                  ;  break
        endcase
    endswitch
  endwhile
endproc

; Function: _HostGetc
; Purpose: Input a character from the port or local keyboard
```

```
proc _HostGetc
  strparm c
  integer i = -1

  while i == -1
    if hitkey                          ; If a key is pressed
      XKEYGET &i                       ;  get the key
    endif
    if comdata                         ; If data available at port
      comgetc i                        ;  get the next character
    endif
    if not connected                   ; If carrier drops
      SETFAILURE                       ;  set error return code
      return                           ;  and return to caller
    endif
  endwhile
  key2ascii i c
  SETSUCCESS
endproc

; Function: ParseUsrRec
; Purpose: Lookup user in .USR file and parse record into globals

proc ParseUsrRec
  integer i
  string tmp
  find uname " " i                     ; i = index of blank name separator
  strcpy ufirst uname i                ; copy first name
  i++                                  ; i = index of last name
  substr ulast uname i 80              ; extract last name
  strfmt tmp "%s;%s;" ulast ufirst     ; 'tmp' is what we're looking for
  strlen tmp i                         ; i = length of name part
  fopen 1 HOSTUSRFILE "rt"             ; Try to open user file
  if success                           ; If opened
    while not EOF 1                    ;  Loop until end of file
      fgets 1 urec                     ;    Get record
      strcmp urec tmp i                ;    Scan record for user
      if success                       ;    If this is our guy,
        COPYSFLD &upassword urec &i FLD_SEP     ; Copy password
        COPYSFLD &uaccess   urec &i FLD_SEP     ; Copy access level
        COPYSFLD &ucomment  urec &i FLD_SEP     ; Copy comment
        SETSUCCESS                     ; set return code to TRUE
        fclose 1                       ; close the file
        return                         ; exit
      endif
    endwhile
  else
    BOXMSG "Error opening user file."
  endif
  fclose 1                             ; close the file
  SETFAILURE
endproc
```

Appendices

APPENDIX

A

PCEDIT Technical Notes

This appendix summarizes the commands available in **PCEDIT**, along with additional technical information. For details about the Setup fields controlling **PCEDIT**, see Chapter 8.

PCEDIT NOTES

PCEDIT is a powerful ASCII text editor with both ASPECT and word processing functions; it handles lines up to 120 characters long, with a maximum of 5000 lines per file.

Multiple files can be edited at once, and two files can be viewed simultaneously. Use Alt-+ (or Alt-=) and Alt-- to traverse the list of files being edited. Information can easily be copied or moved between files. Two separate input modes are available; one tailored for writing ASPECT files, and one for word processing tasks.

When entering text, the full ASCII character range is supported except for the following values (which have special interpretations): 0 (null), 8 (backspace), 9 (tab), 10 (linefeed), 13 (carriage return), 27 (escape) and 255.

The command line syntax for **PCEDIT** is:

PCEDIT [options] filespec(s)

Filespecs may include the "*" and "?" wildcards, and multiple filespecs can be specified. All option switches are optional. Spaces must be used to separate each option and/or filespec. **PCEDIT** supports the following command line option switches:

/B Force use of black & white (monochrome) screen.

/Ln Go directly to line number n.

/V Start in split screen mode.

/N Suppress logo screen.

/U Use uppercase characters in ASPECT macros.

PCEDIT KEYS

The following keys control **PCEDIT** functions:

Enter	Next line; cursor at first char; add line if EOF or insert on
Cur Down	Next text line; cursor column unchanged
Cur Up	Previous text line; cursor column unchanged
PgDn	Display next screen of text
PgUp	Display previous screen of text
Cur Left	Move cursor left one character
Cur Right	Move cursor right one character
Home	First character in current text line
End	One character past last character in current line
Tab	Traverse to next tab stop
Shift-Tab	Traverse to previous tab stop
Insert	Toggle between insert and overtype modes
Esc	Undo changes to current line
Backspace	Delete character left of cursor; move cursor left
Delete	Delete character at current cursor position
Ctrl-PgDn	Last text line in file
Ctrl-PgUp	First text line in file
Ctrl-End	Delete from current cursor postion to end of line
Ctrl-Cur Left	Previous word on text line
Ctrl-Cur Right	Next word on text line
Alt-A	Alter text in file (search and replace)
Alt-B	Block mark
Alt-C	Copy marked block below current line
Alt-D	Delete current line
Alt-E	Erase (delete) marked block
Alt-F	Find first occurrence of search string
Alt-G	Shift text in marked block 1 character left
Alt-H	Shift text in marked block 1 character right
Alt-I	Insert line below current line
Alt-K	Copy marked block and leave block highlighted
Alt-L	Go to specified line number
Alt-M	Move marked block below current line
Alt-N	Find next occurrence of search string in file
Alt-O	Open/redisplay a file for editing
Alt-P	Begin a new paragraph in WP mode
Alt-Q	Quit w/o saving and exit (prompt if text changed)
Alt-R	Recover last changed/deleted line above current line
Alt-S	Display Setup options (through PCSETUP)
Alt-T	Expand tabs in file to spaces

Alt-U	Unmark marked block
Alt-V	Toggle view mode between full- and split-screen
Alt-W	Toggle active split window
Alt-X	Exit current file to DOS, saving any changes
Alt-Y	Clear line marker(s)
Alt-Z	Display PCEDIT On-line help
Alt-1 through 5	Set/Goto line marker 1 to 5
Alt-+	Display next file in edited file list
Alt--	Display previous file in edited file list

Ctrl-A	Alter text in marked block (search and replace)
Ctrl-F	Format a paragraph (WP mode only)
Ctrl-H	Delete character left of cursor; move cursor left
Ctrl-I	Insert line above current line
Ctrl-J	Join next line to current line at cursor position
Ctrl-K	Display On-line Help for ASPECT Keycodes
Ctrl-L	Change characters in marked block to lowercase
Ctrl-M	Merge file at current line
Ctrl-S	Split current line at cursor position
Ctrl-U	Change characters in marked block to uppercase
Ctrl-V	Toggle screen size (lines per screen)
Ctrl-W	Write (save) current contents of file to disk
Ctrl-Z	Display On-line Help for ASPECT Language

PCEDIT has 40 built-in macros for use with ASPECT. Press one of the keys below to automatically insert the indicated words in a script file:

F1	PROC - ENDPROC
F2	IF - ENDIF
F3	ELSEIF
F4	ELSE
F5	SWITCH - ENDSWITCH
F6	CASE - ENDCASE
F7	DEFAULT - ENDCASE
F8	WHILE - ENDWHILE
F9	FOR UPTO - ENDFOR
F10	FOR DOWNTO - ENDFOR

Shift-F1	ATSAY
Shift-F2	FATSAY
Shift-F3	CLEAR
Shift-F4	SCROLL
Shift-F5	BOX
Shift-F6	MESSAGE
Shift-F7	LOCATE
Shift-F8	GETCUR

Shift-F9	CUROFF
Shift-10	CURON
Ctrl-F1	GET
Ctrl-F2	ATGET
Ctrl-F3	MATGET
Ctrl-F4	RGET
Ctrl-F5	STRCPY
Ctrl-F6	STRFMT
Ctrl-F7	STRCAT
Ctrl-F8	STRCMP
Ctrl-F9	SUBSTR
Ctrl-F10	INIT
Alt-F1	PAUSE
Alt-F2	WAITQUIET
Alt-F3	WAITFOR
Alt-F4	WHEN
Alt-F5	CWHEN
Alt-F6	SUSPEND UNTIL
Alt-F7	GETFILE
Alt-F8	SENDFILE
Alt-F9	TRANSMIT
Alt-F10	HANGUP

APPENDIX

B

Compiler and Run-Time Errors

This appendix lists both the possible error messages generated by ASPECT during the execution of a script and the possible warning messages generated by **ASPCOMP** during a compiliation.

Errors that occur while a script is executing are separated into two categories: *non-critical* and *critical errors*. Non-critical errors are those with error numbers less than 100; should one of these occur, you'll be prompted whether or not to abort the currently executing script. If you choose not to abort it, execution will continue. However, unreliable results may occur.

Critical errors are those with error numbers greater than 100. If one of these errors occurs, the script will terminate immediately after the error message is displayed.

If you **SET ASPDEBUG ON**, a file offset will accompany an error message, which allows you to trace the error to a particular source line when using a symbol map generated with a source line reference table (use the **/ML** command line switch with **ASPCOMP** to produce the symbol map file).

SCRIPT EXECUTION (RUN-TIME) ERRORS

Error 1: Value out of range

A value was encountered which was not within the valid range of values accepted for a particular command parameter. For example, a display row value of 25 is out of range when your video mode is 25 X 80 (row values are zero-based). This error message is likely to be the one most often encountered, and does indicate a problem which should probably be corrected. However, range checking can be suppressed by issuing the SET RANGECHK OFF command.

Error 2: Divide by zero

A **DIV** or **MOD** command (or a / or % operator) attempted to divide a value by zero. ASPECT will make no attempt to perform the operation as this would immediately terminate PROCOMM PLUS.

Error 3: String length limit exceeded

An operation which updates or writes to a string variable has attempted to write data past the end of string's data area. The maximum length of any string in ASPECT is 80 characters. This error will usually occur during a STRCAT, STRUPDT, or STRFMT command.

Error 4: Index refers to unopened file

A file I/O operation was attempted on a file which was not currently open. Make sure that the FOPEN command is used before any file I/O operations. This message can be suppressed by setting RANGECHK OFF.

Error 101: Insufficient memory

There is not enough memory available to run the script file. This will occur when attempting to allocate the data space for a script's global string variables. Try reducing the number of global string variables required in the script, or free up more memory by removing unnecessary TSR programs.

Error 102: Invalid ASPECT file name

An invalid file name was specified at the Alt-F5 prompt, or in a Meta Key definition, or with the EXECUTE command. A valid file name consists of up to eight characters optionally followed by a .ASP or .ASX extension. If the file name you specify is valid, but the file does not exist, a "FILE NOT FOUND" message will occur.

Error 103: Unable to compile ASPECT file

Either an error occurred when attempting to run the ASPCOMP compiler, or a compile-time error was found in the script file. In the latter case, an error file with a .ERR extension was probably created (which can be viewed from within PROCOMM PLUS using Alt-A or Alt-V). Otherwise, refer to "Problems When Running External Programs" in Appendix C of your User Manual.

Error 104: Error opening ASPECT file

The compiled .ASX script file was not found or could not be opened.

Error 105: Stack overflow in procedure call

A procedure call requiring passed parameters or a procedure with local variables was called when there was insufficient stack space remaining to continue execution of the script. Usually this occurs if there are procedure calls nested very deeply (as might result in a recursive call) or procedures with large numbers of local string variables.

Error 106: Invalid command encountered

A command was encountered which is not recognized among ASPECT's command set. This is a serious condition which indicates a corrupted file or an error within the compiler. First, attempt to recompile the script file and execute it again. If the error still exists, contact Datastorm's Technical Support Forum on CompuServe (GO DATASTORM), or call Technical Support directly.

Error 107: Unexpected end of file

An attempt to read the next script command from the compiled script failed. See error 106 for more details.

Error 108: Stack overflow in string format

A string format operation (FATSAY, FSTRFMT, or STRFMT) exceeded the maximum length allowed for the output of formatted data.

Error 109: Invalid LONGJMP encountered

A LONGJMP command was attempted before the corresponding SETJMP command was performed, or the corresponding SETJMP was used by another procedure called from the current procedure.

Error 110: Unable to execute remote command

A remote command was not entirely received or couldn't be compiled and/or executed. See the descriptions for errors 103 and 104.

Error 111: Unexpected data type encountered

A data type was not recognized as valid. Contact Datastorm's Technical Support Forum on CompuServe (GO DATASTORM), or call Technical Support directly.

COMPILER ERRORS AND WARNING MESSAGES

Compiler error messages are separated into two categories: those which allow continued compilation (error numbers less than 100), and those which don't (error numbers between 100 and 200). In either case, a .ASX file will not be generated. An *error* message is always accompanied by a source line number, and (where possible) the token which caused the error to occur. Most errors should be easily resolved by checking the description of a command's syntax.

Warning messages will not prevent the generation of a .ASX file; they inform you of a potential problem. Warning messages are numbered 200 and greater. The **/W** command line switch will suppress or control the types of warning messages issued.

Error 1: Invalid token

A parameter was specified which was not allowed. A parameter may be the wrong data type, or may not be an acceptable keyword. Check the command's description for the correct syntax.

Error 2: Unexpected token

An unnecessary token was specified. Check the command's syntax.

Error 3: Unexpected command

ASPECT found a command word not required within the current context of the source code. This can occur with commands that control program flow (ELSE, ENDIF, CASE, ENDWHILE, etc.) or if a command word appears within a numeric expression.

Error 4: Undefined label

A label was referenced in a GOTO command, but the label was not defined within the procedure wherein the GOTO command occurred.

Error 5: Duplicate symbol

A procedure, label, or variable name has been defined more than once. Select another unique name to correct the problem.

Error 6: Missing token

A token was required where none was found. Consult the description of the command for further details.

Error 7: Missing ENDSWITCH command

An ENDSWITCH command was expected where none was found.

Error 8: Missing ENDCASE command

An ENDCASE command was expected where none was found.

Error 9: Missing ENDIF command

An ENDIF command was expected where none was found.

Error 10: Missing ENDWHILE command

An ENDWHILE command was expected where none was found.

Error 11: Missing ENDFOR command

An ENDFOR command was expected where none was found.

Error 12: Unexpected colon

A colon character was found where one was not required. Colons are only used to define labels used in GOTO commands.

Error 13: Undefined procedure

A procedure name was referenced within the script file, but the procedure definition was never found.

Error 14: Missing ENDPROC command

An ENDPROC command was expected where none was found.

Error 15: Missing procedure MAIN

Every ASPECT program must contain a procedure named MAIN which indicates the place where execution of the script should begin.

Error 16: Missing PROC command

A PROC command was expected before the command on the indicated source line. Only certain commands may occur outside the body of a procedure.

Error 17: Variable limit exceeded

An attempt was made to define more variables than are allowed within a source file. There is a limit of 128 global variables of each data type, and 128 parameters and local variables per each procedure.

Error 18: Variable not defined

A variable name was used, but no variable with that name was previously defined. A variable must be defined before it is referenced.

Error 19: Variable already defined

An attempt to define a variable was made when the name was already used by another variable. Local variables must have names that are unique from other existing global variables.

Error 20: Invalid command or expression token

A token was encountered that was neither a valid ASPECT command, numeric expression operator, value or variable, nor a string variable.

Error 21: Invalid numeric variable or constant

A token was encountered that was neither a previously defined numeric variable nor a valid numeric constant of the acceptable data type(s).

Error 22: Invalid numeric variable

A token was encountered which does not represent a previously defined numeric variable of the acceptable data type(s).

Error 23: Invalid string constant (quotes missing)

A token presumed to represent a string constant was encountered, but the quotes surrounding the string were missing.

Error 24: Invalid string variable

A token was encountered which does not represent a previously defined string variable.

Error 25: Invalid identifier

An illegal name (used to define a procedure, label, variable or macro) has been encountered. See "Naming Elements in ASPECT" in Chapter 2 of the ASPECT Script Language Reference Manual for the correct naming conventions.

Error 26: Label used in another procedure

A label name was encountered which was already being used within another procedure. Label names within a script file must be unique, even though a label can only be referenced from within the procedure wherein it is defined.

Error 27: Invalid floating point operation

A floating point value was encountered in an numeric operation which does not accept floating point values. These operations include bit-wise manipulation, shifting, and unary complement.

Error 28: Invalid use of reserved word

A procedure, label, variable, or macro name was encountered which is identical to an ASPECT reserved word and therefore cannot be used. See Appendix C for a complete list of reserved words.

Error 29: Macro already defined

A macro with the same name was already defined earlier in the script file. To define a macro name more than once, the previous definition of the macro name must be removed using the UNDEF command.

Error 30: Missing ENDCOMMENT command

The end of the source file was reached while in a COMMENT block.

Error 31: Missing $ENDIF command

Then end of the source file was reached while within a $IFDEF command block.

Error 32: Missing terminating quote

A terminating quote character is required within a string or character constant.

Error 33: Line truncated during macro expansion

A macro name was encountered in a command line and upon replacing the macro name with its defined text, the maximum allowable source line length was exceeded. Split the source line using a backslash character just before the macro name.

Error 34: Invalid expression token

A token was encountered within a numeric expression that was neither a valid operator nor a numeric constant or variable.

Error 35: Missing operator

In a numeric expression, an operator was expected between two operands, or between an operand and a unary operator.

Error 36: Missing operand

In a numeric expression, an operator was being processed without an available operand.

Error 37: Missing right paren

In a numeric expression, a right parenthesis was expected to match a corresponding left parenthesis.

Error 38: Missing left paren

In a numeric expression, a right parenthesis was encountered without a corresponding left parenthesis.

Error 39: Missing variable operand

In a numeric expression, a variable operand was expected between two operators.

Error 40: Expression too complex

A numeric expression could not be evaluated due to the large number of nested operators and operands. Break the expression into several smaller sub-expressions.

Error 41: Missing space between tokens

A space is required between each token on a command line. This error occurs when a space is missing between a token on either side of a quoted string constant.

Error 42: Unexpected escape sequence

An escape character was encountered outside of a string or character constant.

Error 43: Maximum token length exceeded

A token was encountered which exceeds the maximum allowable token length of 80 characters.

Error 44: Invalid variable operand

In a numeric expression, a variable operand type was required in order to apply a particular operator (usually =, ++, or −).

Error 45: Invalid pass by reference

A parameter was encountered in a procedure call which is not a defined variable and therefore cannot be passed by reference. Constants may only be passed by value (no ampersand allowed).

Error 46: Invalid number of parameters

A procedure call has been encountered which does not include the correct number of parameters as defined by the procedure being called.

Error 47: Missing WITH before parameter list

A procedure call was encountered which includes a parameter list without the required WITH keyword.

Error 48: Missing comma in token list

A comma is required to separate a list of tokens which are used to define variables or parameters.

Error 49: Mismatched parameter type

A parameter in a procedure call was found to be a different data type than expected by the procedure being called. Make sure your parameters are the same type and in the same exact order as those listed in the procedure's definition.

Error 50: Escape sequence character out of range

An escape sequence evaluated to an invalid character value. Escape sequences can only represent character values from 0 to 255.

Error 51: Invalid character constant

A character constant represents a character value from 0 to 255, and therefore cannot consist of less or more than one character. Note that valid escape sequence evaluates to a single character value even though the sequence may be defined using more than one character.

Error 101: Stack overflow (nesting too deep)

A command which defines a nested command block (such as IF, WHILE, FOR, SWITCH, etc.) is nested too deeply to be evaluated by the compiler; or, an INCLUDE command was nested more than 10 levels deep. The script file or INCLUDE files must be written differently to avoid the nesting problem.

Error 102: Stack underflow

An internal stack used by the compiler was found to be in an incorrect state. Please report the problem on DATASTORM's Technical Support Forum on CompuServe (GO DATASTORM), or call Technical Support directly.

Error 103: Insufficient memory

There is insufficient memory to compile the script file. First, check that there is at least 128K of memory available, and then attempt to compile the script again. If this fails, the script is too large to be compiled and should probably be split up into smaller files which can then be executed separately with the EXECUTE command. Please report the problem on DATASTORM's Technical Support Forum on CompuServe (GO DATASTORM), or call Technical Support directly.

Error 104: Invalid command line option

A command line option was encountered which is not supported by the compiler. Run the compiler without any command line arguments to display the available options.

Error 105: Invalid script file name

The script file name supplied is longer than 8 characters or does not have a .ASP extension.

Error 106: Error opening script file

The script file name supplied refers to a non-existent file, or a file which cannot be opened successfully.

Error 107: Error opening output file

The compiler was unable to open the output file (.ASX extension) for code generation. If the file already exists, make sure that it's not marked read-only.

Error 108: Missing script filespec

The script file name must be provided on the command line along with any options. The compiler will not prompt for it.

Error 109: Error writing to output file

An error has occurred while generating the output file. Make sure that there is sufficient disk space to create the file. The compiled .ASX output file usually requires less space than the .ASP source file.

Warning 201: Unreferenced local label

A label was defined within a procedure, but it was never referenced with a GOTO command.

Warning 202: Unreferenced parameter

A procedure defines a parameter which was never used within the procedure's command block.

Warning 203: Unreferenced local variable

A procedure defines a local variable which was never used within the procedure's command block.

Warning 204: Unreferenced global variable

The script file defined a global variable which was never referenced by the procedures contained in the generated output file.

Warning 301: Ignoring unreferenced procedure

The script file contains a procedure which was never called by another procedure within the script file, or was called only by another procedure which was never called. The compiler will not generate code for this procedure thus reducing the output file size and saving disk space.

APPENDIX

C

Reserved Words

This appendix lists the reserved words set aside in ASPECT for named elements.

For more information on the restrictions concerning reserved words and ASPECT named elements, please refer to Chapter 2 of this manual.

$COL
$DATE
$D_BAUD
$D_DATABIT
$D_DUPLEX
$D_ENTRY
$D_KBDFILE
$D_LDATE
$D_METAKEY
$D_MODE
$D_NAME
$D_NOTE
$D_PARITY
$D_PHONE
$D_PORT
$D_PROTO
$D_PWORD
$D_SCRIPT
$D_STOPBIT
$D_TERM
$D_TOTAL
$ELSE
$ELSEIF
$ENDIF
$FATTR
$FDATE
$FEXT
$FILENAME
$FNAME
$FSIZE
$FTIME
$H_BAUD
$H_ELAPSED
$H_LEVEL
$H_NAME
$H_OFFLINE
$H_ONLINE
$IFDEF
$NULL
$ROW
$SCRNCOLS
$SCRNROWS
$TERMBOLD
$TERMDIM
$TERMNORM
$TERMREV
$TERMULINE
$TIME0
$TIME1
1KXMODEM
1KXMODEMG
25X80
2KWINDOW
4KWINDOW
8QUOTE
8STRIP
ABORTDL
ADD
ADDS60
ADDS90
ADM3
ADM31
ADM5
ALARM
AND
ANDL
ANSI
ANSI8BIT
ASCII
ASK
ASPDEBUG
ASPECT
ASSIGN
ATGET
ATIME
ATOF
ATOI
ATOL
ATSAY
ATT4410
ATT605
AUTOANSOFF
AUTOANSON
AUTOBAUD
AUTODLOAD
BACKSPACE
BAUD
BAUDRATE
BINARY
BLANKEX
BLANKON
BLINKON
BLOCK
BLOCKCHECK
BLOCKCUR
BOLDON
BOX
BREAK
BYE
CALL
CALLOG
CALLPAUSE
CASE
CDHIGHINIT
CDINXFER
CEIL
CHARACTER
CHARPACE
CHATMODE
CHDIR
CISB
CLEAR
CLIPCHAR
CLOSE
CLOSED
CNCT1200
CNCT19200
CNCT2400
CNCT300
CNCT38400
CNCT4800
CNCT9600
COM1
COM2
COM3
COM4
COM5
COM6
COM7
COM8
COMDATA
COMGETC
COMGETCD
COMMENT
COMP

COMPUTC
CONNECT
CONNECTED
CONNECTION
CQUOTE
CR
CRC16
CRC32
CR_LF
CTRL
CURDN
CURLF
CUROFF
CURON
CURRT
CURUP
CWHEN
DATABITS
DATE
DEC
DECIMAL
DEFAULT
DEFINE
DELCHAR
DELETE
DELLINE
DEST
DGD100
DGD200
DGD210
DIAL
DIALCMND
DIALENTRY
DIALSUFFIX
DIMON
DIR
DIRECT
DISCONNECT
DISKFREE
DISPLAY
DIV
DLDIR
DLOAD
DLXPROTO1
DLXPROTO2
DLXPROTO3
DN_CR
DN_LF
DN_TO
DOS
DOSMODE
DOSVER
DOWNTO
DROPDTR
DSCROLL
DUPLEX
EBOL
EBOS
ECHO
EDITOR
EEOL
EEOS
ELSE
ELSEIF
EMULATE
EMULATION
ENDCASE
ENDCOMMENT
ENDFOR
ENDIF
ENDPROC
ENDSWITCH
ENDWHILE
ENQ
EOF
EOLCHAR
EQ
ERRDETECT
ERRORMSG
ESPRIT3
EVEN
EXECUTE
EXIT
EXITCDHIGH
EXITFOR
EXITSWITCH
EXITWHILE
EXPLODE
EXTPROTO1
EXTPROTO2
EXTPROTO3
EXTRAX80
EXTRAXUSER
FAILURE
FASTKBD
FATSAY
FCLEAR
FCLOSE
FETCH
FFLUSH
FGETC
FGETS
FILETYPE
FIND
FINDCASE
FINDFIRST
FINDNEXT
FINISH
FLOAT
FLOATPARM
FLOOR
FNLOOKUP
FOPEN
FOR
FOREVER
FOUND
FPUTC
FPUTS
FREAD
FROMDDIR
FSEEK
FSTRFMT
FTELL
FTOA
FULL
FWRITE
GE
GET
GETCUR
GETDIR
GETENV
GETFATTR
GETFDATE
GETFILE
GETFSIZE

GETFTIME
GETVATTR
GETVCHAR
GOODBYE
GOSUB
GOTO
GT
H19
HALF
HANDSHAKE
HANGUP
HARDFLOW
HELP
HITKEY
HOME
HOOK
HOST
IBM3101
IBM3161
IBM3270
IBMPC
IF
IGNORE
IMODEM
INC
INCLUDE
INIT
INPORT
INSCHAR
INSLINE
INSMODE
INTEGER
INTPARM
ISFILE
ITOA
KEEP
KERMIT
KERMSERVE
KEY2ASCII
KEYGET
KEYS
KFLUSH
LE
LF
LINEFEED
LINEPACE
LOCATE
LOG
LOGFILE
LOGOUT
LONG
LONGJMP
LONGPARM
LOOPFOR
LOOPWHILE
LT
LTOA
MARK
MATGET
MAXDIAL
MDIAL
MEMFREE
MEMPEEK
MEMPOKE
MEMREAD
MEMWRITE
MESSAGE
METAKEY
MGET
MKDIR
MLOAD
MOD
MODEM
MODEM7
MONO
MOUSEX
MOUSEY
MSPAUSE
MUL
N0
N1
N2
N3
N4
N5
N6
N7
N8
N9
NEG
NEGOTIATE
NEQ
NEWUSERLVL
NO
NOCLEAR
NOCNCT1
NOCNCT2
NOCNCT3
NOCNCT4
NONDEST
NONE
NORMON
NOT
NULL
ODD
OFF
ON
OPEN
OR
ORL
OUTPORT
OVERWRITE
PACECHAR
PACKSIZE
PADCHAR
PADNUM
PARITY
PARMREST
PARMSAVE
PAUSE
PAUSECHAR
PORT
PRINTER
PROC
PROGRAM
PROTECT
PROTOCOL
PRTNAME
PULLDNKEY
PULLDNMENU
PUSHBACK
PUTENV
PUTVATTR
PUTVCHAR
QUIT

RANGECHK
RASCII
RCA
RDFLUSH
RDWRITE
RECVCRASH
RECYCLE
REDIAL
REDISPLAY
REG
RELAXED
REMOTECMD
RENAME
RESUME
RETURN
REVON
REWIND
RFLUSH
RGET
RMDIR
RSTRCMP
RUN
RXDATA
S0
S1
S2
S3
S4
S5
S6
S7
S8
S9
SCREEN
SCROLL
SEALINK
SENDCR
SENDCRASH
SENDFILE
SET
SETFATTR
SETFDATE
SETFTIME
SETJMP
SHELL
SHELLBOOT
SHL
SHR
SNAPSHOT
SNOW
SOFTFLOW
SOUND
SPACE
STARTCHAR
STATLINE
STATMSG
STATREST
STOPBITS
STRCAT
STRCMP
STRCPY
STREAMING
STRFMT
STRING
STRIP
STRLEN
STRLWR
STRPARM
STRPEEK
STRPOKE
STRSET
STRUPDT
STRUPR
SUB
SUBSTR
SUCCESS
SUSPEND
SWITCH
SWITCHCASE
SYSTYPE
TABEX
TELINK
TERMBOLD
TERMDIM
TERMINAL
TERMKEY
TERMNORM
TERMRESET
TERMREV
TERMULINE
TERMWIDTH
TERMWRT
TEXT
TIME
TIMEOUT
TIMESTAMP
TRANSLATE
TRANSMIT
TTY
TV910
TV912
TV920
TV922
TV925
TV950
TV955
TXMETHOD
TXPACE
TYPE
ULDIR
ULINEFONT
ULXPROTO1
ULXPROTO2
ULXPROTO3
UNDEF
UNTIL
UPTO
UP_CR
UP_LF
ULINEON
USERMODE
USERMSG
USERVID
VGALINES
VIDREST
VIDSAVE
VIDSTART
VIEWUTIL
VT100
VT102
VT220
VT320
VT52
WAIT
WAITCASE

WAITCNCT
WAITFOR
WAITQUIET
WHEN
WHENCASE
WHILE
WITH
WRAP
WRITEC
WRITEPROT
WXMODEM
WYSE100
WYSE50
WYSE75
XFERKEY
XMODEM
XOR
YES
YMODEM
YMODEMG
ZERO
ZMODEM

APPENDIX

D

Operators

This appendix provides technical details concerning the use of operators in ASPECT.

Several operators may be used in mathematical expressions in the ASPECT programming language. Each operator has a precedence level and associativity, and most operators have an equivalent command form (except for the parens, compound assignment operators and comma operator).

Precedence defines order of operator evaluation in the absence of parentheses. Operators of higher precedence are evaluated before those with lower precedence. Parentheses are used to override the default precedence. For example, the expression A + B * C would be evaluated by multiplying B and C together and then adding A because * has a higher precedence level than +. The expression (A + B) * C, however, would first evaluate A + B, and then would multiply the result by C.

Associativity determines the order of evaluation when examining two operators with equal precedence (note: all operators having the same precedence level always have the same associativity.) Left to right associativity means that an operand is grouped with the operator on its left, whereas right to left associativity means that an operand is grouped with the operator on its right. For example, the expression A * B / C would be evaluated by multiplying A and B, and then dividing that result by C. Operand B groups with * rather than / because * and / are left to right associative. The expression A = B = C, however, is evaluated by assigning the value of C to B and then to A. B is assigned a value before A because = is right to left associative.

Two special unary operators, ++ and --, may be applied to either side of an operand. In either case the effect is to increment or decrement the operand. However, when the operator is applied to the right side of an operand, the value of the operand is used in the context of the expression before it is incremented or decremented. When applied to the left side of an operand, the value is used in an expression is the result after the operand is incremented or decremented. For example, in the expression A + B--, A is added to B, and then B is decremented. In the expression A + --B, first B is decremented, and then it is added to A.

The compound assignment operators can both transform and assign values in a single step. For example, the expression A += B is equivalent to the expression A = A + B.

Except for the left and right parens and the unary operators, all operators listed here are binary and act upon two operands.

ASPECT OPERATOR TABLE

Operator	Type	Precedence	Associativity	Command
(	GROUP	13	Left to Right	---
)	GROUP	13	Left to Right	---
!	UNARY	12	Right to Left	NOT
-	UNARY	12	Right to Left	NEG
~	UNARY	12	Right to Left	COMP
++	UNARY	12	Right to Left	INC
--	UNARY	12	Right to Left	DEC
*	BINARY	11	Left to Right	MUL
/	BINARY	11	Left to Right	DIV
%	BINARY	11	Left to Right	MOD
+	BINARY	10	Left to Right	ADD
-	BINARY	10	Left to Right	SUB
>>	SHIFT	9	Left to Right	SHR
<<	SHIFT	9	Left to Right	SHL
<=	RELATE	8	Left to Right	LE
>=	RELATE	8	Left to Right	GE
<	RELATE	8	Left to Right	LT
>	RELATE	8	Left to Right	GT
==	RELATE	7	Left to Right	EQ
!=	RELATE	7	Left to Right	NEQ
&	BITWISE	6	Left to Right	AND
^	BITWISE	5	Left to Right	XOR
|	BITWISE	4	Left to Right	OR
&&	LOGICAL	3	Left to Right	ANDL
||	LOGICAL	2	Left to Right	ORL
=	ASSIGN	1	Right to Left	INIT
*=	ASSIGN	1	Right to Left	---
/=	ASSIGN	1	Right to Left	---
%=	ASSIGN	1	Right to Left	---
+=	ASSIGN	1	Right to Left	---
-=	ASSIGN	1	Right to Left	---
<<=	ASSIGN	1	Right to Left	---
>>=	ASSIGN	1	Right to Left	---
&=	ASSIGN	1	Right to Left	---
|=	ASSIGN	1	Right to Left	---
^=	ASSIGN	1	Right to Left	---
,	COMMA	0	Left to Right	---

Index