

Short Cuts™

for fast, efficient
programming

by Kelly Puckett



penguin software™

the graphics people

ShortCuts™

by Kelly Puckett

CONTENTS

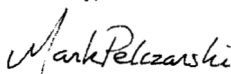
1. Introduction	3
2. Getting Started	4
2.1 Loading ShortCuts	
2.2 Sampler	
3. General Definitions	6
4. General Syntax Structure	8
5. Primary Input/Output Commands &INPUT and &PRINT	10
6. Commands Affecting Input and Output	15
6.1 Input and Output Fields and Positioning &TAB cursor positioning &POS output-only positioning &LEN length of input/output fields &LEFT\$, &RIGHT\$ left and right justification &NORMAL, &INVERSE video mode on &INPUT commands &PLOT, &GR printing to disk or a printer	
6.2 Input Directives &DEF, &NEW allowing default entries &TRACE allowing "trace-over" entries &OR, &GET allowing null entries &STEP input without RETURN key	
6.3 Commands Affecting Numeric Input/Output &PDL position decimal &EXP set magnitude limit &ABS, &SGN signed number entry &VAL, &FN, &SIN allowing calculation	
6.4 Commands Affecting String Input/Output &ASC defining classes of acceptable characters &FOR format masks	
7. Secondary Input/Output Commands	27
&HOME for 80 columns &TAB cursor positioning &HLIN using 24th screen line &SAVE, &RECALL saving a text page &SCRN setting screen width &PR# setting non-standard video output	

8. Other Secondary Commands	29
&RND rounding numeric values	
&& using other &utilities	
9. Program Flow Commands	30
&CONT control character definition	
&ON...GOTO...	
&ON...GOSUB control character branching	
&ON...RESTORE restarting input	
&RETURN normal returns from &GOSUB	
10. Sorting Commands	34
&LIST	
11. Using ShortCuts with Disk Files	37
11.1 &INPUT from Disk	
11.2 &PRINT to a Text File	
12. Operating Environment	38
12.1 Non-Standard Display Devices	
12.2 Programming Utilities	
Appendix A	40
Initialization Parameters	
Appendix B	41
Global and Local Parameters	
Appendix C	42
User Error Messages	
Appendix D	44
A detailed look at loading	
Appendix E	46
Command Reference Card	

The enclosed product is supplied on a disk that is NOT copy-protected. It is our intent to make this product as useful as possible, and we feel that with applications software the ability to make your own backup copies is a great asset. We ask that you not abuse our intentions by making copies for others, and by not copying this manual. The result of such activity only helps promote the unfortunate existing situation of most software being protected, and, in our opinions, less usable. We hope that the commercial success of non-protected products such as this one will signal other publishers that copy-protection is not necessary for a product's survival and help reverse the trend in protected applications software. At Penguin, through policies such as this and our pricing, we are trying to look out for your best interests. Please help us.

We hope you find many hours of enjoyment and use in this package.

Sincerely,



Mark Pelczarski
President, Penguin Software



penguin software™
the graphics people

P.O. Box 311, Geneva, IL 60134 (312) 232-1984

ShortCuts software and manual is copyrighted 1983 by Penguin Software. All rights reserved. Use of routines from **ShortCuts** in other products for sale must be licensed by Penguin Software. There is no fee. Apple is a trademark of Apple Computer, Inc. We also have to legally say the following: **ShortCuts** is sold entirely "as is". The buyer assumes the risk as to quality and performance. Penguin Software warrants to the original purchaser that the disk on which this software is recorded is free from defects in materials and workmanship under normal use, for a period of 60 days from the date of purchase. If a defect should occur within this period, Penguin will replace the disk at no charge. After 60 days, there is a \$5 replacement fee when the original disk is returned. Manuals will NOT be replaced, so don't lose it or let your dog eat it. At no time will Penguin Software or anyone else who has been involved in the creation, production, or distribution of this software be liable for indirect, special, or consequential damages resulting from use of this program, such as, but not limited to, hurricanes, fires, minor head colds, divorce, or beady eyes.

CHAPTER 1

INTRODUCTION

ShortCuts is a utility designed to simplify the development of error-proof interactive programs to give your software a polished and professional appearance. This is done by actually adding extra commands to Applesoft BASIC. ShortCuts is easy to use, adds extra power to your programming, and best of all, it will save you a lot of time.

ShortCuts adds the extra commands to Applesoft by using the '&' command. It can be used with other programs that use '&', and is compatible with most program editors. Each of the ShortCuts commands begins with '&'. They allow you to format and accept specific input in your programs, format output, and perform fast, powerful sorts.

Note that each ShortCuts command borrows an existing keyword from Applesoft, such as PRINT, FOR, PDL, and so on. While it may seem odd at first, there's a good reason for doing this: Applesoft keywords are each stored in one byte. Any word that's not a keyword takes one byte per letter. That means 2 - 6 bytes per command are saved by using the keywords.

Licensing

There is no licensing fee for using ShortCuts routines in your own programs for sale. All we require is acknowledgement that ShortCuts was used. We do require a license form obtained from and filed with Penguin Software, however. Requests for Licensing Agreements must be made in writing

CHAPTER 2

GETTING STARTED

ShortCuts is an addition to your Applesoft programming language, and includes more than 30 new commands. Learning to use ShortCuts will be easier than when you learned BASIC, but you can make it go even easier by doing the following:

- 1) Skim or glance over the manual
- 2) Load ShortCuts on your Apple
- 3) Run the SAMPLER program
- 4) Run, list, and look at the demonstration programs
- 5) Reread the manual

Experiment with the various commands. Start with simple input and output commands and increase the complexity as you desire. The simplest ShortCuts &INPUT statement packs a lot of power all by itself. We'll start by getting ShortCuts loaded on your Apple, but first make a backup copy of your ShortCuts master disk and put the original in a safe place. ShortCuts can be copied with any standard copy program. We allow this for convenience on your part; please don't give copies to friends.

2.1 LOADING ShortCuts

The ShortCuts routines are written entirely in machine language and are supplied on the program disk as a group of binary (type B) files. There are actually three versions of ShortCuts. Version one is the complete ShortCuts. The second version is a bit shorter and saves some memory by omitting the commands needed to sort data. The third version contains only the sorting functions.

In addition, each of these versions of ShortCuts is available in two varieties. The variety you will be using most often—while you are experimenting with ShortCuts, or editing and debugging an applications program—is designed to be located in the upper area of your Apple's memory. The second variety is designed to be appended to an Applesoft BASIC program once that program has been completed. Do not be dismayed by the apparent number of options. They each have their purpose, and you may eventually find a use for all of them.

Also on the program disk is a short Applesoft BASIC program named "LOADER" which should be RUN to load ShortCuts on your Apple. LOADER will install one of the versions of ShortCuts just below the HIMEM address in effect when it is run. Therefore, any other utilities or programs already in high memory will not be overwritten, provided that HIMEM has already been set to protect them. You may load ShortCuts while using PLE¹, RENUMBER² or any other ampersand utility.

1. Program Line Editor or Global Program Line Editor from Synergistic Software
2. Provided on your Apple DOS System Master

Go ahead and do it now. With the ShortCuts disk in your disk drive, type "RUN LOADER". When the disk drive stops, you will be given a menu from which you can choose one of the varieties of ShortCuts to be loaded. Unless there is reason for doing otherwise, select the complete version: "INPUT/OUTPUT WITH SORTING". When (RETURN) is pressed, loading will begin. In about seven seconds the message, "ShortCuts SUCCESSFULLY LOADED" should appear on the screen. If so, you may skip the next two paragraphs.

If ShortCuts was unable to load, and you got the message "MEMORY CONFLICT", then some other utility present on the computer has set HIMEM so low that there was not room for ShortCuts. Turn it off and reboot with the System Master or the program disk and rerun "LOADER". This will correct the memory conflict.

If the loading was unsuccessful and you got a DOS error message, the second possibility, a defective disk, must be considered. Just to make sure, try to load ShortCuts on another Apple. If ShortCuts loads correctly on another machine, a hardware failure on your machine is indicated.

If you'd like detailed information on the loading process, see Appendix E.

2.2 SAMPLER

You should now have RUN LOADER and have ShortCuts loaded and initialized on your computer. A short program named SAMPLER is supplied on the disk containing ShortCuts. Please RUN SAMPLER now.

SAMPLER is designed to demonstrate some basic features of ShortCuts. Running and completing SAMPLER now will make it easier to understand the rest of the manual.

SAMPLER consists of a number of one or two statement loops in the general form:

```
10 INPUT X
20 GOTO 10
```

On the upper half of the screen will be listed the program statement which is the heart of the loop, in this case 10. You will be able to do the loop as many times as you wish, trying different responses. To proceed to the next loop, use the ESCape key.

The first part of SAMPLER demonstrates INPUTting values for the three types of Applesoft variables: string, real, and integer. Next, SAMPLER shows some of ShortCuts' ability to format input and output. SAMPLER goes on to show how ShortCuts can be used to validate input and display meaningful error messages. Finally, the use of fully formatted input is demonstrated by allowing you to enter some telephone numbers.

SAMPLER does not demonstrate all of ShortCuts' features. Its main purpose is to let you act out the role of the user, while seeing the actual commands being executed. When you are through with SAMPLER, you might wish to run the other demonstration programs on the disk: CALCULATOR and ADDRESS BOOK. These two programs demonstrate some more advanced applications programmed with ShortCuts. CALCULATOR turns the Apple into a visible calculator, and uses many of ShortCuts' numeric formatting features. ADDRESS BOOK shows how ShortCuts can be used to fill data screens with input from the keyboard and from disk.

Now that you have seen some of what ShortCuts can do, you might want to begin playing with the utility. No harm done, but please come back later and finish reading the manual!

CHAPTER 3

GENERAL DEFINITIONS

Chapter 4 of this manual contains a definition of the general syntax structure used by ShortCuts. Chapters 5 - 10 contain a detailed look at every command available in the language. Before getting to this, we need to define some terms.

SCREEN FIELD: A portion of the screen where text or values may be PRINTed or where user input is echoed during an INPUT statement. (In Applesoft, all screen fields begin at the current cursor position and extend either 255 screen columns, in the case of INPUT statements, or indefinitely, as in PRINT statements.)

INPUT FIELD: The screen field being used during a user INPUT.

OUTPUT FIELD: The screen field being used by a PRINT statement or the screen field in which a user INPUT is displayed upon completion of the INPUT. (In Applesoft INPUTs, there is no difference between the input field and output fields.)

TARGET VARIABLE: The variable to which a value is being assigned. In an assignment statement such as $A = B + C$, the variable to the left of the equals sign is the target variable. In the statement, "INPUT X", X is the target variable.

VALIDATION CRITERIA: The criteria which a user INPUT must satisfy in order to be considered a valid input.

IMPLICIT VALIDATION CRITERIA: The validation criteria imposed upon an INPUT by the hardware or the programming language being used. For example, in the Applesoft statement "INPUT X%", implicit validation criteria are that the input must be numeric, contain no commas, and be in the range -32767 to 32768, since X% is an integer variable.

EXPLICIT VALIDATION CRITERIA: Validation criteria imposed by the programmer. The Applesoft BASIC statement.

100 INPUT "ENTER A NUMBER FROM 1 TO 10"; X: IF X > 10 OR X < 1 THEN 100

creates an explicit validation criteria for X. Values of X outside the range 1-10 will cause the input statement to be re-executed.

LINE: That part of a program beginning with a line number and ending with the beginning of the next line or with the end of the program.

STATEMENT: Part of a program delimited at its beginning by a line number or by the colon ending the previous statement, and at its end by a colon or the the end of the line.

APPLESOFT STATEMENT: A statement which is to be interpreted and executed by Applesoft BASIC.

SHORTCUTS STATEMENT: A statement which is to be interpreted and executed by ShortCuts, and not by Applesoft BASIC. All ShortCuts statements begin with "&".

ACTION COMMAND: Any command within a statement which, during the execution of that statement, causes the computer to actually do something, i.e., PRINT something to the screen, INPUT a value to assign to a variable, etc. (In Applesoft BASIC, all commands except ONERR GOTO are action commands.)

PARAMETER: A quantity or directive which may be specified by the programmer and will determine how ShortCuts executes a statement. Examples might include screen field length, whether output is to be left- or right-justified, or whether negative numbers are to be allowed as input.

GLOBAL PARAMETER: A parameter which, until changed, applies to all statements within a program, unless overridden by a local parameter.

LOCAL PARAMETER: A parameter which is in effect only for the statement currently being executed. Local parameters, when set, take precedence over global parameters.

CHAPTER 4

GENERAL SYNTAX STRUCTURE

The '&' starts each ShortCuts statement. If there is no '&', the statement is treated as Applesoft. If there is an '&', the entire statement, until the next colon or end of line, uses ShortCuts commands.

All Applesoft statements are simple in the sense that each statement may contain only one executable command. This limits the amount of "work" that may be gotten from a single statement. In contrast, any ShortCuts statement may be complex, containing multiple commands, subject to the rules which follow:

Command words—&HOME, &INVERSE, &RETURN—may be called verbs. Many ShortCuts commands consist only of a single verb. Other commands require additional information in order to operate correctly—&PRINT X or &LEN (8). Those quantities, qualifiers, or target variables which follow the verb are called the modifiers of the command verb.

A ShortCuts statement may contain, at most, one action command. It may contain zero or many parameter setting commands. A ShortCuts statement may be subdivided as follows:

Global Parameter-Setting Phrase

The global phrase of a ShortCuts statement begins with the opening of the statement and ends with the action verb. If the statement contains no action command, the entire statement is a global phrase.

Multiple parameter setting commands within the global phrase should be separated from each other by commas. The last command in the global phrase should be separated from an action verb by a comma.

Any parameter setting command which occurs within the global phrase will set the appropriate global parameter. If the parameter is one which may be set locally but not globally, the command is meaningless but will not generate an error message.

Action Command

After the global parameter setting, if one is used, comes the action command, such as &INPUT, &PRINT, or &LIST. If no action verb appears, the entire ShortCuts statement is treated as a global parameter setting statement.

Local Parameter-Setting Phrase

If the ShortCuts statement contains either action command &PRINT or &INPUT, a local parameter setting phrase may exist. The local parameter phrase begins immediately following the action verb and extends to the beginning of the action verb's modifiers.

Any parameter setting command which occurs within the local parameter phrase will set the appropriate local parameters. If the parameter is one which may be set globally but not locally, the global parameter will be set and no error message will be generated.

Within the local parameter setting phrase, commas should be used to:

- 1) separate the action command from the first parameter setting command,
- 2) separate multiple parameter setting commands, and
- 3) separate the last parameter setting command from the action verb's modifiers.

Action Verb Modifiers

If the action verb requires or has optional modifiers, they appear after local parameter settings, at the end of the ShortCuts statement.

The use of these rules will become apparent in the examples throughout the following text. For now, though, we can say that the most complex ShortCuts command will look like:

&global commands, action command, local parameters, modifier

Most of your uses will have only part of the above included.

The following sections contain a detailed look at each command used by ShortCuts. We have attempted to provide as thorough a description of each command as possible. However, the great degree of flexibility that ShortCuts gives to the programmer is achieved, in part, by making the execution commands dependent on parameters set by others. For the programmer, this means that the tree cannot be seen clearly until he understands the forest. If this is your first experience with ShortCuts, please skim the entire manual. The fifteen or twenty minutes that this takes will give you a good start at understanding the forest.

Note that in referring to ShortCuts verbs, we always precede them with an '&' to distinguish them from their Applesoft counterparts. As described in this section, though, the '&' only appears at the beginning of the ShortCuts statement. Each verb after that is a ShortCuts verb, even though the extra '&'s are not used.

CHAPTER 5

INPUT/OUTPUT COMMANDS

A major part of ShortCuts is to simplify keyboard input/video output interactive programming. Therefore, those commands which cause input or output to occur are ShortCuts' primary action commands. Not too surprisingly, these commands are "&INPUT" and "&PRINT."

Many aspects of the execution of the &INPUT and &PRINT commands may be controlled by the use of the parameter and directive setting commands described in Chapter 6. A complete definition of the &INPUT and &PRINT commands, therefore, would include these parameters and the means by which they are set.

The definitions in this section provide an overview of the primary action commands, stressing the features that are always, or usually, in effect.

In the syntax definitions, the square brackets [] mean the enclosed part of the command is optional.

&INPUT

&INPUT [prompt;] target variable [validation criteria] [,error message]

If the above definition is broken into its component parts, left to right, we find:

- 1) The action command verb, "&INPUT"
- 2) A prompt for the entry
- 3) The target variable to which the &INPUT will be assigned
- 4) An explicit validation criteria, used to determine the validity of the user's entry.
- 5) A programmer-defined error message to be used if the entry was invalid

The prompt, target variable, validation criteria, and error message (items 2-5 above) are the modifiers of the action verb, "&INPUT." The target variable is required. The other modifiers are optional.

The target variable may be any legal Applesoft variable: string, real or integer. It may be a simple variable or an array variable.

Prompts

The optional prompt, if used, must begin with a string in quotes. Except for this limitation, any legal Applesoft string expression may be used as a prompt. The requirement that the prompt begin with a string may be satisfied with a null string (""). If present, the prompt must be separated from the target variable by a semicolon.

The following statements all contain legal prompts:

- ```
10 &INPUT "NAME?"; X$
20 &INPUT "AGE OF" + X$ + "?"; Y
30 &INPUT "OK TO SAVE AS RECORD NO." + STR$(RN) + "?"; Z$
```

The prompt in line 20 is the entire string expression, "AGE OF" + X\$ + "?". In line 30 the prompt is the expression "OK TO SAVE AS RECORD NO." + STR\$(RN) + "?".

## Validation Criteria

The optional validation criteria may be used to examine the entry before it is assigned to the target variable. If the entry does not meet the criteria, the entry will be rejected. If the name of the target variable is followed by an arithmetic or logical operator (+, -, \*, /, <, >, =, AND, OR), ShortCuts assumes that a validation criteria has been established. Any expression which could be evaluated in an Applesoft IF...THEN... statement can be used as a ShortCuts validation criteria. The validation criteria is evaluated in the same way as an expression used in an Applesoft IF...THEN... statement. If the expression is true, the entry is accepted and assigned to the target variable. If the expression is evaluated as false, the entry is rejected.

Rejection of the entry causes the user error message "INVALID ENTRY" to be displayed. When the error is acknowledged by pressing the space bar, the cursor will be replaced at the beginning of the input field and the entry will be restarted.

Some examples of &INPUT statements which use validation criteria, and their effects, follow:

```
10 & INPUT X > 0
```

(An entry less than or equal to zero will be rejected.)

```
20 & INPUT "WHO WAS THE FIRST PRESIDENT>>"; P$ = "WASHINGTON"
```

(Any entry other than "WASHINGTON" will be rejected.)

```
30 & INPUT "ENTER A NUMBER FROM 1 TO 10>>"; X => 1 AND X <= 10
```

(Numbers less than one or greater than ten will be rejected.)

```
40 & INPUT "ENTER A NUMBER>"; X: & INPUT "NOW ENTER A LARGER NUMBER>"; Y
> X
```

(Any legal number may be entered for X. Any entry for Y must be greater than X or it will be rejected.)

## User Error Message

If a validation criteria has been established for the entry, the programmer may specify an error message to be used when an invalid entry is rejected. The programmer-defined error message will be used instead of the default error message, "INVALID ENTRY".

The programmer-defined error message must follow and be separated from the validation criteria by a comma. The error message may be any legal string expression, but like a prompt must begin with a quoted string, even if it is a null string.

In order to test the validity of the entry, ShortCuts momentarily assigns the entry to the target variable. If the entry is invalid, the target variable is restored to its original value. The timing of the assignment and restoration is designed so that the incorrect entry may be made a part of the programmer-defined error message.

Consider the following statement:

```
10 & INPUT "WHO IS BURIED IN GRANT'S TOMB?"; X$ = "GRANT", "" + X$ + " IS
INCORRECT"
```

The validation criteria makes "GRANT" the only legal entry. If the user enters "LINCOLN", he will be given the user error message "LINCOLN IS INCORRECT". Immediately after displaying this message, the value of X\$ will be restored to its value before the entry. (Note the use of a null string to satisfy the requirement that a programmer-defined error message must begin with a string.)

No programmer-defined error message may be longer than the width of the display screen minus two. Exceeding this limit will cause a "STRING TOO LONG" error.

## Execution of &INPUT

Having defined the elements of a ShortCuts &INPUT statement, we may now further describe the execution of the statement. (Please note that almost everything which follows may be affected or prevented by the use of the parameter setting commands in Sections 5.3-5.7 of this manual.)

ShortCuts begins the execution of the INPUT statement by printing the prompt, if one exists. It then clears the input field, turns to the keyboard and waits for the user to make an entry. The entry is checked character by character as it is typed. The following rules are used to determine whether the character is accepted or rejected:

- 1) If the character is a control character or the ESC character, it is compared to the currently valid control character table. (See Chapter 9) If it is in the table, the predefined program branch will be performed. If not, the character will be rejected. No other control character may be a part of any entry, whether string or numeric.
- 2) If the target variable is a string variable, and the character is a member of the current valid character group (see Section 6.4), it is accepted. Otherwise it is rejected.
- 3) If the target variable is numeric and the character is a digit, it will be accepted, as long as the resulting number does not exceed any of the possible magnitude or decimal precision limitations imposed on the entry. Decimal points, +, -, and calculation signs and functions are accepted if the current parameters allow.
- 4) ShortCuts was designed to reject null entries. If the first character of the entry is the RETURN key, it will be rejected. (But see the &DEF and &OR commands in Section 6.2.)

If the character is rejected, an appropriate user error message will be displayed and the keyboard will be locked, forcing the user to acknowledge the error. Pressing the space bar will allow the entry to resume. (Appendix C contains a list of system-generated user error messages.)

If the character entered at the keyboard is accepted, it will be displayed on the screen. The cursor will be advanced, and the next character may be entered. The entry is completed when the user presses the return key.

No entry may be longer than the input field. If the input field length is 10, for example, the entry may not contain more than ten characters. The carriage return is not counted here as a character. If the user attempts to enter one more character than the field length, the user error message, "ENTRY TOO LONG", will be displayed. Except for the different error message, ShortCuts treats the extra character as if it were an illegal character.

What happens after the entry is completed depends on whether the target variable is a string or a numeric variable.

If the target variable is a string variable, any spaces at the beginning of the entry will be eliminated. Execution will proceed with testing the validation criteria, if one exists.

If the target variable is numeric, ShortCuts will check to see that the entry is within the range of numbers supported by the variable type. For integer variables, the legal range is -32767 to 32768. For real variables, the legal range is -1E38 to 1E38. If the entry is out of range, a user error message will be displayed explaining the error. After the error is acknowledged, the cursor will be replaced at the beginning of the input field, without erasing the erroneous entry. This allows the user to correct the mistake, using the right arrow key to type over any part of the entry which is correct.

If a validation criteria for the entry has been established, it will now be tested, as described earlier. If the test fails, either the system-generated error message, "INVALID ENTRY", or the programmer-defined error message will be displayed. After the message is acknowledged, the cursor will be replaced at the beginning of the input field, without erasing the entry, and the entry process will be restarted.

If the validation test is successful, or if there is no test, the output field, which may be identical to the input field, will be cleared. The successful story will then be displayed in the output field, right- or left-justified and formatted according to the parameters set by the commands in Chapter 6.

With the display of the entry in the output field, the execution of the &INPUT statement is complete.

The &INPUT statement may contain both local and global parameter setting commands. Within an &INPUT statement:

\*\* Parameter and directive setting commands which occur before the action verb, "INPUT", are in the global phrase of the statement and set the appropriate parameter globally.

\*\* Parameter and directive setting commands which occur between the action verb and its first modifier (the prompt or target variable) are in the local phrase of the statement and set the appropriate parameter locally.

In this dummy statement

```
10 & !!!!!, INPUT, ****, "PROMPT"; X$
```

the exclamation points represent a parameter setting command in the global phrase. The asterisks represent a parameter setting command in the local phrase. The commas separating the action verb from the parameter setting commands are required.

## &PRINT

&PRINT [string message:] variable [;]

The ShortCuts &PRINT command may be used to print formatted data to the screen or other output device. Left- and right-justification, formatted strings, fixed decimal point format, and display rounding are all supported by the &PRINT command.

When a numeric variable is printed with fixed decimal formatting, the value of the variable is automatically rounded to the number of decimal places displayed.

Consider the following short program:

```
10 X = 1.437
```

```
20 & PRINT, PDL(2), X
```

PDL(2) means "Position Decimals at 2 places".

When X is printed with 2 decimal places, it will be rounded to and displayed as 1.44. Please note: the value of the variable is actually and irreversibly changed to equal the value displayed!

In a similar fashion, when a string variable is printed in a screen field which is shorter than the length of the string, the string will be truncated to the length of the screen field. After the following short program was run, X\$ would be equal to "WASH":

```
10 X$ = "WASHINGTON"
```

```
20 & PRINT, LEN (4), X$
```

LEN(4) sets the screen output field.

The ShortCuts &PRINT command may only be used to print the value of a single variable. Unlike Applesoft's PRINT, it will not print expressions or multiple variables separated by semicolons. It will not perform tabbing between variables separated by commas.

Ordinarily, when the value of the variable has been printed, ShortCuts sends a carriage return to the output device. If the name of the variable to be printed is followed by a semicolon, no carriage return will be printed.

Although the &PRINT command may be used to format and print only the value of a single variable, a string expression may be printed before the variable. The string expression is treated in the same manner as an &INPUT prompt and requires the same syntax. The first element in the expression must be a string, even if it is a null string. (The expression must be separated by a semicolon from the variable that is to be printed.)

The following statements use a string expression to precede the variable being &PRINTed:

10 & PRINT "YOUR ACCOUNT BALANCE IS"; B

20 & PRINT "YOU MADE YOUR LAST DEPOSIT ON"; DT\$

30 & PRINT "YOUR BALANCE BEFORE YOUR DEPOSIT ON" + DT\$ + "WAS"; PE

In the design of ShortCuts' commands, more emphasis was given to methods of creating "entry formats"—number definitions used to limit input—than to output formatting. In its ability to format output, therefore, ShortCuts is not as versatile as most forms of PRINT USING. If floating dollar signs, asterisk fills, leading zeros, etc. are really needed for a particular application, ShortCuts should be used in conjunction with a PRINT USING utility. However, most applications really require only fixed decimal position and right justification. ShortCuts fulfills these needs easily, and usually in a more memory-efficient way than any PRINT USING.

# CHAPTER 6

## COMMANDS AFFECTING INPUT AND OUTPUT

### 6.1 INPUT AND OUTPUT FIELDS AND POSITIONING

The following set of commands allow the programmer to define the screen fields and display modes used by ShortCuts &INPUT and &PRINT commands. These new commands, when used by themselves, do nothing. Instead, they establish parameters and directives which influence the operation of ShortCuts input/output commands.

Many of these commands may be used to set their parameters either globally or locally. If the parameter setting command is the only command in the statement or if it occurs within the global phrase (i.e., before &PRINT or &INPUT), the appropriate global parameter is set. If the parameter setting command occurs within a local phrase (i.e., between &INPUT or &PRINT and its modifiers), the corresponding parameter is set locally. Examples of the differences between global and local commands can be found in Appendix B.

In the following definitions, square brackets mean the enclosed is optional, "aexpr" means any valid arithmetic expression, and "\$expr" means any valid string expression.

**&TAB** local (for global definition, see Chapter 7)

**&TAB** (aexpr1, aexpr2)

When used within the local phrase of an &INPUT or &PRINT command, &TAB fixes the location of the left edge of the input or output field at the display coordinates specified by aexpr1 and aexpr2.

Aexpr1 specifies the number of the display line on which the screen fields begin. The topmost line is line number 1. The value of the expression must be in the range 1 to 24 or an ILLEGAL QUANTITY error will be reported.

Aexpr2 defines the column on the screen at which the input or output field begins. The leftmost column of the display is column number 1. The arithmetic expression must be in the range 1 to X where X is the width in columns of the display screen at the time ShortCuts was initialized. For the standard Apple display, this value is 40. If this range is exceeded, an ILLEGAL QUANTITY error will occur.

The statement:

```
10 & INPUT, TAB (12,1), X
```

will position the cursor on the twelfth line at the left edge of the video display before beginning to &INPUT a value for X.



**&POS** local only (POStion output)

**&POS** (aexpr1, aexpr2)

&POS is identical in usage, syntax and operation to TAB, with one exception. &POS sets only the position of the output field and has no effect on the input field.

Every ShortCuts &INPUT statement uses both an input field, where the user's entry appears during input, and an output field, where the entry is displayed after it has been validated and accepted. Ordinarily, these two screen fields occupy the same location and may be treated as one field.

There may, however, be occasions when the programmer desires to separate the fields. For instance, it might be desirable to get all entries on the 23rd line of the display and use those entries to fill in a form on lines 1 through 20. The &POS command makes this possible.

Because &TAB positions the output field as well as the input field, if both &TAB and &POS are used in a statement, &POS should be used after &TAB. Otherwise, the settings specified by the &POS command will be overwritten.

**&LEN** global or local (LENGth of field)

**&LEN** [#/\$/] (aexpr)

&LEN is used to set the length of input and output fields. The length, in characters, which is set is the value of aexpr. The value of the expression must be in the range 1 to 255 or an ILLEGAL QUANTITY error will result.

ShortCuts maintains separate values in its global or parameter tables for the lengths of screen fields used with numeric target variables and for those used with string target variables. The programmer may often desire to set different field lengths for the different types of target variables. If the "#" character is used immediately after "LEN", the field length for numeric target variables will be set. Using the "\$" character will cause the field length for string target variables to be set. Attempting to globally set screen field length without specifying "#" or "\$" will cause a SYNTAX ERROR.

When setting field length locally, it is unnecessary, but not an error, to specify "#" or "\$". Using both "#" and "\$" will cause a SYNTAX ERROR.

The statement:

```
10 & LEN # (16), LEN $ (22)
```

globally sets a length of 16 characters for screen fields used with numeric variables, and a length of 22 characters for fields used with string variables.

**&LEFT\$** local and global (LEFT justify)  
**&RIGHT\$** local and global (RIGHT justify)

**&LEFT\$** [#/\$/]

**&RIGHT\$** [#/\$/]

These commands are used to specify whether left or right justification should be used by ShortCuts when printing to an output field. The type of justification set will be used by both primary action commands. In the case of the &INPUT statement, the entry is justified when it is reprinted to the screen at the conclusion of the &INPUT.

With left justification, the target variable's value (formatted, if it is numeric) will be printed starting at the left edge of the output field. If the entry is shorter than the field, the balance of the field will be filled with blank spaces.

With right justification, sufficient blank spaces will be generated before printing the variable so that the last character of the variable will be at the right edge of the output field.

ShortCuts maintains separate justification modes for printing numeric and string target variables. When globally setting the justification mode, the "#" and "\$" characters must be included in the command to specify whether the mode is being set for numeric or string variables.

The statement

```
10 & LEFT$ $, RIGHT$ #
```

sets left justification for string variables and right justification for numeric variables. When globally setting the justification mode, omitting the "#" or "\$" characters will cause a SYNTAX ERROR.

When these commands are used locally, it is not necessary to specify the variable type.

**&NORMAL** local and global  
**&INVERSE** local and global

**&NORMAL**  
**&INVERSE**

These commands set the video mode to be used by the &INPUT command in the input field. If the mode is set to &INVERSE, the display within the input field will be black on white. If &NORMAL, then white on black. Using an inversed input field provides a nice highlight for the data being entered.

The video mode will revert to that set by Applesoft before the entry is redisplayed in the output field.

Some 80-column cards support inverse display in only a limited fashion or not at all. Therefore, if the programmer is writing for an 80-column environment, it might be safest to avoid the inversed input field.

**&PLOT** local and global

**&GR** local and global  
**&PLOT**  
**&GR**

A few extra functions of the &PRINT command must be disengaged before sending output to a printer or to a disk text file. ShortCuts usually clears the output field, does a carriage return, then replaces the cursor at the beginning of the output field. On a video display, all this activity is unnoticed; on a printer, or in a disk file, it makes a mess.

&PLOT may be used to inform ShortCuts that output is going to a printer or to a disk file, and the field clearing and extra carriage returns will be curtailed. The &GR command cancels &PLOT and causes ShortCuts to resume those activities needed to keep a clean screen field.

## 6.2 INPUT DIRECTIVES

One of ShortCuts' features is its support for default entries. When default entries are allowed, the user need not actually make an entry unless a change to the default value is desired.

Three of the commands described in this section—&DEF, &NEW and &TRACE—turn on or off the default entry mode, or otherwise affect the information presented to the user during an &INPUT. Two other commands—&OR and &GET—allow the programmer to specify whether null responses to &INPUT statements should be accepted or rejected. The last command, &STEP, allows the programmer to speed up data entry by eliminating the need for carriage returns.

These commands affect the operation of ShortCuts whether the &INPUT is for numeric or string variables.

**&DEF** Local and global (DEFault entry)

&DEF

The &DEF command makes ShortCuts' default entry mode active. In the default mode, when executing an &INPUT statement, ShortCuts displays the current value of the target variable in the input field before looking to the keyboard for an entry. If the first character entered by the user is the carriage return, ShortCuts proceeds exactly as if the user had typed in the current value. If any other key is pressed first, the default option is cancelled for that entry.

Using the default mode when it is appropriate can make your program faster and easier to use, and much friendlier. For example, if the user is editing previously entered data, making the default mode active will save a lot of retyping.

Setting the default mode active does not bypass the validation criteria. Consider the following program:

```
10 & DEF
20 X = 85
30 & INPUT X < 10
```

In the execution of line number 30, ShortCuts will print the number "85" in the input field and position the cursor just before the "8". This signals the user that a default entry is available. However, if the user accepts the default by hitting the carriage return, he will get an "INVALID ENTRY" message because the explicit validation criteria,  $X < 10$ , is not satisfied.

No default value may be offered without displaying the default. For this reason, the &TRACE command (see below) is ineffective when the default mode is active. If a default value is offered to the user, the programmer should also beware of permitting null entries. (See the &OR command, below.) Simultaneously allowing both null and default entries can create user confusion.

**&NEW** local and global (NEW entry)

&NEW

The &NEW command cancels ShortCuts' default entry mode, forcing the user to make a new and valid entry in response to an &INPUT statement. The input field will be blank, unless the &TRACE command has been used.

**&TRACE** local only (TRACE over entry)

**&TRACE**

Ordinarily, ShortCuts clears the input field before looking to the keyboard for an &INPUT. Using the &TRACE command will preserve the contents of the input field by preventing the clearing of the field. This may be useful in situations where it is likely that the user will want to use the right arrow key to overtype old information on the screen, but it would be incorrect to offer the old information as a default entry.

If the default mode is active, it is essential that the default value be visible. Therefore, the &TRACE command is ineffective when the default mode is active.

**&OR** local and global (Optional Response)

**&OR**

&OR—"Optional Response"—directs ShortCuts to accept a null entry as a valid response to an &INPUT statement. A null entry is an entry which consists of zero characters, or which contains only spaces.

One of ShortCuts' primary functions is to insure correct and valid data input during the execution of an application program. Therefore, ShortCuts' customary mode of operation, and its default mode, is to reject null entries. There may, however, be occasions when the user might reasonably have no response to the &INPUT statement, but it is still desirable to collect the response through ShortCuts. The &OR command makes this possible.

When a null entry is accepted, the value of the null entry is assigned to the target variable. For a string target variable, a null entry is equivalent to a null string. When the target variable is numeric, the null entry has a value of zero. A null entry, once accepted, will be tested (just like a normal entry) against any validation criteria established by the programmer.

It is possible to create logically contradictory programs. Consider the following:

```
10 & OR
20 & INPUT X > 0
```

Line 10 globally sets the null entry acceptable mode. Line 20 imposes an explicit validation criteria on X which cannot be satisfied by a null entry. Should the user make a null entry, he would be informed that the entry was invalid.

The programmer should beware of simultaneously offering a default entry and authorizing the acceptance of a null entry. If both entry options are offered, and the first character the user enters is the carriage return, ShortCuts will return the default value.

When &INPUTting fixed length, formatted strings (see the &FOR command), null entries will not be accepted regardless of the null entry acceptance mode. The null entry acceptance mode may be set either locally or globally.

**&GET** local and global

**&GET**

The &GET command directs ShortCuts to reject a null entry offered in response to an &INPUT statement.

When first initialized, ShortCuts is in the null entry rejection mode. Unless the &OR command has been used to globally authorize null entry acceptance, the programmer will have no need to use the &GET command.

## **&STEP** local

### **&STEP**

Ordinarily, ShortCuts waits for the user to end an &INPUT response with a carriage return. However, if the &STEP command is used in the local phrase of the &INPUT statement:

```
10 & INPUT, STEP, X$
```

the entry will be automatically concluded when the user has filled the screen field. For example, if the input field is five characters long, after the user has entered the fifth character, ShortCuts will proceed as if the fifth character and a carriage return had been entered.

As long as the programmer provides some means of getting back to edit previous entries, judicious use of the &STEP command can add a nice touch of automation to data entry. The demonstration program ADDRESS BOOK uses this technique, in conjunction with ShortCuts' program branching commands, to move forward and back through the data entry screen.

## **6.3 COMMANDS AFFECTING NUMERIC INPUT/OUTPUT**

This section describes a group of parameter setting commands which are used to control the operation of ShortCuts &INPUT and &PRINT statements when the target variable is numeric.

Using these commands, the programmer may:

- 1) set the number of decimal places to be allowed and displayed when &INPUTting or &PRINTing real variables.
- 2) limit the acceptable magnitude of a number being &INPUT.
- 3) allow or prevent the entry of signed numbers in response to an &INPUT statement, and
- 4) permit the user to make calculations, either arithmetic or scientific, during entries.

There is considerable interaction among the numeric formatting commands used by ShortCuts: &LEN #, &PDL, &EXP, &ABS, &SGN. It is easy for the programmer to attempt to create an 'impossible' format. ShortCuts works very hard to resolve improperly set parameters. If a command seems to be without effect, or an unexpected ILLEGAL QUANTITY ERROR appears, the best way of finding the problem is to consider the entire numeric format in effect.

Please keep in mind that there is nothing in ShortCuts which extends the precision of Applesoft beyond nine digits. You may not use ShortCuts commands to make it appear that a greater level of precision is being used.

## **&PDL** local and global (Position DecimaL)

**&PDL** (aexpr)

**&PDL** is used to set the decimal format parameter, or decimal precision, to be used when entering or printing the values of numeric variables. The decimal format parameter may be set either locally or globally.

The decimal format parameter specifies the number of digits to the right of the decimal point to be displayed, or allowed, when:

- 1) a numeric variable is **&PRINTed**,
- 2) a default value is displayed in the input field at the beginning of an **&INPUT** statement,
- 3) a successful entry is redisplayed in the output field at the conclusion of a ShortCuts **&INPUT** statement, and
- 4) the user is responding to an **&INPUT** statement with a real target variable and no calculations are allowed.

**&PDL** (aexpr) sets the decimal format parameter to the value of aexpr. The arithmetic expression must be in the range 0 to 9, or an **ILLEGAL QUANTITY** error will occur. Setting the decimal format parameter to 9 will cause all numeric formatting (except justification) during output to be curtailed. If this is done, ShortCuts will print numbers in the normal Applesoft format. For example, .008 will be printed "8E-3".

During the **&INPUT** of a real target variable, ShortCuts prevents the user from entering more decimal places than are specified by the decimal format parameter. The first digit entered in excess of the decimal format parameter will cause ShortCuts to present the user error message, **ONLY # DECIMAL PLACES ALLOWED**. (The "#" in the message will equal the decimal format parameter.)

Integer variables may be formatted and **&PRINTed** like real variables. However, if the target variable of an **&INPUT** statement is an integer variable, no decimal fraction can be entered.

When the numeric screen field length is too short to contain the number of decimal places specified by PDL, an unresolvable parameter conflict exists. The result will be an **ILLEGAL QUANTITY ERROR**. The error will not be detected and reported until ShortCuts tries to execute a **&PRINT** or **&INPUT** command with the conflicting parameters. As a result, the actual programming error may have occurred many statements before the line where the error occurs.

Each time the **&PDL** command is used, the current setting of the input magnitude limiter (see **&EXP**) is cleared. If the decimal format parameter is changed locally, only the local setting of the input magnitude limiter will be cleared.

## **&EXP** local and global

**&EXP** (aexpr)

The **&EXP** command sets the input magnitude limiter to the value of aexpr. Aexpr must be in the range 0-37 or an **ILLEGAL QUANTITY ERROR** will result. The **&EXP** command is intended to make it possible for the programmer to limit the magnitude of acceptable responses to ShortCuts **&INPUT** statements with numeric target variables. If the input magnitude limiter is set to X, only user entries with a value in the range  $\pm 10 \uparrow X$  will be accepted.

If an entry is rejected by ShortCuts because it exceeds the range given by the input magnitude limiter, a user error message of the form "MUST BE SMALLER THAN +/-XXXXXX" will be generated. The X's in the message will be replaced with the correct limit.

Every combination of screen field length, decimal format parameter and signed number acceptance rule (see &LEN, &PDL, &ABS, and &SGN) implies a maximum practical magnitude limit.

For example, suppose the screen field length is 8, signed numbers are allowed, and 2 decimal places are to be displayed. If this is the case, then only numbers smaller than +/- 10 ↑ 4 can be properly displayed. That is, there is only room in the screen field for 4 digits to the left of the decimal point. The maximum practical magnitude in this case is 4.

Each time ShortCuts executes an &INPUT statement it calculates the maximum practical magnitude for that &INPUT. &EXP may be used only to set the input magnitude limiter smaller than the calculated maximum practical magnitude. Attempting to set the input magnitude limiter to a value larger than the maximum practical magnitude will be ineffective, but will not result in an error message.

The input magnitude limiter must be in the range 0-37. However, values greater than 8 can be effective only if the decimal format parameter is equal to 9, which curtails ShortCuts' numeric formatting.

In the hierarchy of parameters, the input magnitude limiter is very low. If ShortCuts can resolve a parameter conflict by resetting the magnitude limiter, it will do so. Each time the &PDL command is executed, the corresponding local or global setting of the input magnitude limiter is cleared. For this reason, the &EXP command, if used, should always occur after the &PDL command.

**&ABS** local and global (ABSolute value entry)

**&SGN** local and global (SiGNed entry)

&ABS

&SGN

These commands may be used to allow or prevent the entry of signed numbers in response to an &INPUT statement. &SGN directs ShortCuts to allow the entry of signed numbers. &ABS directs ShortCuts to reject attempts to enter a signed number. If signed numbers (and calculations) are not allowed, the first character of the entry must be a digit or a space. If signed numbers are allowed, the first character may also be a plus or minus sign.

Setting the signed number acceptance rule with &ABS makes it impossible to enter a negative number, since a negative (minus) sign will not be allowed.

If any form of calculation at &INPUT time is allowed (see &VAL, &PN and &SIN below), &ABS will not prevent the entry of signed numbers.

The use of &SGN and &ABS may affect other numeric formatting parameters. If signed numbers are allowed, the first position of the input field is reserved for the display of the sign. The remaining size of the input field is then used to determine the number of digits left and right of the decimal point which may be correctly displayed.

**&VAL** local and global (VALues only)

**&FN** local and global (FuNctions allowed)

**&SIN** local and global (ScieNtific functions allowed)

&VAL

&FN

&SIN

If the programmer wishes, ShortCuts will allow the user to enter arithmetic or scientific expressions in response to &INPUTs of numeric target variables. This group of commands is used to specify the type of calculation, if any, that will be permitted.

The &VAL command prevents the entry of arithmetic or scientific expressions. When the calculation flags have been cleared by the &VAL command, ShortCuts will accept only a valid numeric entry in response to an &INPUT statement with a numeric target variable.

The &FN command directs ShortCuts to accept simple arithmetic expressions as valid entries. The arithmetic expressions may include addition, subtraction, multiplication, and division. The definition of valid numeric characters is expanded to include the arithmetic operator symbols: + - \* and/. In addition, parentheses will also be accepted as part of the entry. When the calculation mode has been set by &FN, expressions such as the following are legal entries:

$6 + 4 + 3$

$5 * 25 + 1.7$

$(22 + 5) / 103$

When the user has completed the entry of an expression by pressing the carriage return, the expression is evaluated. If decimal formatting is active (see the &PDL command, above), the value of the expression is rounded to the number of decimal places being displayed. This rounded value is used to test the entry against any explicit validation criteria established for the entry. If the validation is successful, the value is displayed in the output field. The automatic rounding assures that the displayed value is consistent with the value of the variable when it is used later in the program.

The expressions will be evaluated using the hierarchy of operators used by Applesoft. All expressions in parentheses are evaluated first. Multiplications and divisions are done next, from left to right. Additions and subtractions are done last, also left to right.

If the expression contains syntax errors or causes a division by zero, the user will be informed of the error by an appropriate user error message. After the error is acknowledged, the cursor will be replaced at the left edge of the input field, without clearing the field, and the user will be allowed to correct the entry.

The &SIN command directs ShortCuts to accept scientific expressions. In this mode, character by character checking of the entry is curtailed.

When the entry of scientific expressions has been authorized by the programmer, any character except a control character may be a part of the response to a numeric &INPUT statement. The result is that any expression which can be evaluated by Applesoft is a legal entry. The &SIN command should be used to allow the entry of expressions containing scientific calculations such as:

$\text{INT}(\text{COS}(1.5)) + 10 \quad 3$

$\text{LOG}(12) * \text{TAN}(3)$

$9\text{E}7 - 12 + \text{ATN}(2) * \text{COS}(3)$



The evaluation, rounding, validation and redisplay of the value of a scientific expression is identical to that performed with an arithmetic expression.

A danger associated with using the &SIN command is that it allows any expression to be entered. Expressions containing Applesoft variables are legal. If the expression references a previously undimensioned array variable, the memory space required by the array will be allocated. It is possible for this to cause an unanticipated OUT OF MEMORY error. For this reason, programmers should be careful is using the &SIN command to allow scientific expressions in programs which use vital or irrecoverable data and which will be operated by relatively naive users.

If either arithmetic or scientific expressions are allowed by the programmer, and the program is intended for use by anyone else, the program documentation should explain the rounding rule and the hierarchy of operations.

## 6.4 COMMANDS AFFECTING STRING INPUT/OUTPUT

The commands described in this section allow the programmer to:

- 1) define the kind of character which is to be considered a valid response to an &INPUT statement with a string target variable, and
- 2) create formatted, fixed length input fields, which force the user to make a complete and correct entry.

**&ASC** local and global (AScertain Character set)

&ASC (string/svar)

The &ASC command is used to define the class of characters which may be entered in response to an &INPUT for a string target variable. ShortCuts groups all characters which may be entered from the Apple keyboard to seven groups, each identified by a letter:

- 1) Group D: The digits 0 through 9
- 2) Group N: The digits 0 through 9, the space, and the decimal point.
- 3) Group L: The letters A through Z.
- 4) Group A: The letters A through Z, the space, and the period.
- 5) Group B: The digits 0 through 9, the letters A through Z, the space, and the period or decimal point.
- 6) Group X: All characters except the comma, the double quote, or the control characters.
- 7) Group Z: All characters except the control characters.

During the execution of an &INPUT statement with a string target variable, ShortCuts will check each character as it is entered. If it is not a member of the programmer-defined legal character group, an appropriate user error message will be generated.

The first character of the string or string variable in the &ASC command is used to define the group of characters which is to be considered by legal ShortCuts. For example, the statement:

```
10 & ASC ("D")
```

globally defines character group D, i.e. digits, as the legal character group for future string entries. The statements:

```
10 X$ = "LAST"
20 & ASC (X$)
```

make character group L the legal character group. ("L" is the first character in X\$.)

Only the letters D,N,L,A,B,X and Z may be used to specify a character group. If any other letter is the first character in the string or string variable following the &ASC command, a SYNTAX ERROR will result.

To illustrate the use of the &ASC command, suppose you are having the user enter a person's name. The command,

```
10 & INPUT, ASC ("A"), X$
```

will restrict the entry to alphabetic characters, spaces and periods. Any other character will cause the user error message, "ENTRY MUST BE ALPHABETIC."

In another example, you want the user to enter a numeric serial number, but want the data stored in a string variable. Using the statement:

```
10 & INPUT, ASC ("D"), X$
```

will limit the entry to digits. Characters other than digits will result in the user error message, "ENTRY MUST BE NUMERIC".

An example that uses several of ShortCuts' features: You need to have the user enter a valid DOS file name for his data file. The statement:

```
10 &INPUT, LEN(20), ASC ("B"), "FILE NAME>>";X$<<"AND ASC (LEFT$ (X$, 1))>
64, "MUST BEGIN WITH LETTER!"
```

will do the trick. This statement sets the field length to 20, then defines letters, digits, spaces and periods as legal characters. The prompt, "FILE NAME>>" will be printed and then the user may enter the file name, X\$. After the entry is complete, the validation criteria will be tested. The test will pass if X\$ is not null and the ASCII value of the first character is greater than 64, that is, if it is a letter. Otherwise, the test will fail and the programmer-defined error message, "MUST BEGIN WITH LETTER!", will be presented to the user. (Not all legal file names will be accepted by this statement, but any name that is accepted will be legal.)

**&FOR** local and global (FORmat)

&FOR (string/svar)

The &FOR command is used to define a format mask for the entry or output of a fixed length string variable. The format mask may be a string or a string variable and must be enclosed in parentheses.

A format mask is a pre-definition of the string to be input or output. The definition encompasses the size of the string, the type of character which may be in each position in the string, and any "edit" or format characters which should be displayed with the string. The format mask may be considered a "picture" of the data to be entered.

Each character in the format mask string is either 1) a character group identification letter which defines the type of data which is to be placed in that position in the string, or 2) an edit character, which is to be displayed along with the data. If the character in the format string is one of the following:

L,D,A,N,B,X or Z

then that character is a place holder for a character in the data string and defines the type of character which is to be in that position in the data string. These are the same entry character group identifiers used by the &ASC command. If the character in the format string is not one of these letters, then it is considered an edit character which is to be displayed when the data string is &INPUT or &PRINTed, but is not itself a part of the data string.

At least one of the first four characters of the format mask should be a data character place holder. If not, default entries cannot be implemented, and ShortCuts will be unable to

determine whether or not the input is coming from a disk text file.

The fixed length, formatted mode is intended to be used when &INPUTing or &PRINTing data which has a fixed, invariable format. The examples which follow are typical of this kind of data and will help explain the use of fixed length, formatted string input and output.

Social security numbers consist of nine digits which are almost always printed in three groups, separated by hyphens: 123-45-6789. The unique data actually consists only of the nine digits. The hyphens make the data more recognizable and more memorable. To create a format mask for this data type, replace, character by character, the actual data with the most restrictive ShortCuts character group identifier which includes all of the legal characters which might be in that position. Leave the two hyphens in place. The resulting format mask is "DDD-DD-DDDD." That is, three data digits, a format hyphen, two data digits, a format hyphen and four data digits.

The ShortCuts statement:

```
10 & INPUT, FOR ("DD-DD-DDD"), "SOC SEC NO: "; X$
```

will prompt the user to enter a "SOC SEC NO". After the third and fifth digits, hyphens will be printed to the screen, without the user having typed them. The complete entry will be returned in the variable, X\$. X\$ will contain only the data (nine digits) and not the hyphens, which are format characters.

Telephone numbers consist of a three digit area code, followed by a seven digit phone number. Often, for readability, the area code is enclosed in parentheses and the first three digits of the phone number are separated from the last four by a hyphen: (123) 456-7890. The unique data of a phone number is contained in the ten digits which make up the number. The parentheses and hyphen are format characters. A ShortCuts format string which could be used to &INPUT or &PRINT a telephone number is "(DDD) DDD-DDD".

Suppose that within an application program the ten digits of a telephone number ("1214546789") are stored in the variable X\$, and the following commands are executed:

```
10 TF$ = "(DDD) DDD-DDDD"
20 & PRINT, FOR (TF$), X$
```

Working from left to right, ShortCuts scans the format mask and inserts a character from the data string in each place holder of the format string, skipping over the edit characters. The resulting string is then printed. The screen will display:

```
(121) 454-6789
```

The format string "LL-DDDD" could be used to control the input or output of product identification numbers which consist of two letters, a hyphen and four digits. To enter or print five digit zip codes, use the format mask: "DDDDDD".

When executing a fixed length, formatted &INPUT, ShortCuts will force a complete data entry. If the user hits the carriage return before entering the number of characters given by the format mask, the user error message, "COMPLETE ENTRY REQUIRED", will be presented. If ShortCuts tries to format a data string which does not contain enough characters to fill the format mask, garbage will be printed to the screen.

If the target variable of the &INPUT or &PRINT statement is a numeric rather than a string variable, the &FOR command will be ineffective, but no error message will be generated.

When &INPUTing or &PRINTing a fixed length, formatted string, ShortCuts does not use the screen field length parameter set by &LEN. The length of the input and output fields is determined by the length of the format mask. For this reason, left- and right-justification during output are not supported when fixed length, formatted strings are being &INPUT or &PRINTed.

# CHAPTER 7

## SECONDARY INPUT/OUTPUT COMMANDS

The commands described in this section are miscellaneous commands which have been included to make using ShortCuts more convenient or more effective for the programmer. In the following syntax definitions, square brackets [ ] mean the enclosed part is optional, 'aexpr' stands for any valid arithmetic expression, and '\$expr' stands for any valid string expression.

### **&HOME**

**&HOME**

This command is provided for the convenience of programmers who are working with 80-column video cards. When the standard 40-column Apple display is in use, & HOME is identical to the Applesoft HOME. However, &HOME will also clear the screen of some 80-column displays that HOME won't.

**&TAB**      global

**&TAB** (aexpr1, aexpr2)

When used by itself, or in the global phrase of a ShortCuts command, &TAB causes the cursor to be positioned on the aexpr1'th line, at the aexpr2'th column of the video display. Aexpr1 must be in the range 1 to 24 or an ILLEGAL QUANTITY error will result. Aexpr2 must be in the range 1 to N, where N equals the width of the screen, or an ILLEGAL QUANTITY error will result.

(&TAB has a somewhat different effect when used within the local phrase of a primary action statement. Please see the section describing screen field parameter setting commands for details on the local usage).

### **&HLIN**

**&HLIN** \$expr

ShortCuts uses the 24th line of the video screen for displaying user error messages. Therefore, the 24th line may not practically be used for &INPUT, and anything &PRINTed to the 24th line will be destroyed by the first error message.

&HLIN causes the string expression in the statement to be printed on the 24th line, centered left and right. In addition, the resulting string is saved within ShortCuts so that it may be redisplayed after being temporarily overwritten by a user error message.

The string, or the result of the string expression, to be displayed by HLIN must be shorter than the width of the display less 2; i.e., for the normal Apple video display, the string may not be longer than 38 characters. If this length is exceeded, a STRING TOO LONG error will occur. &HLIN leaves the cursor in the lower right hand corner of the video display. To avoid scrolling, the programmer should move the cursor to a free part of the screen before allowing anything else to be printed.

The video mode—normal or inverse—in effect at the time HLIN is executed will be used in printing the string, but will not be saved. Subsequent redispays will be done in the video mode active at the time of the redisplay.

## **&SAVE and &RECALL**

Occasionally, a programmer might wish to save a video display screen and be able to redisplay it at a later time, perhaps after using the display to enter other information or display a "help" message. The &SAVE and &RECALL commands are provided to make this possible.

&SAVE copies the screen in its entirety to a block of memory set aside for that purpose. &RECALL copies the memory block back to the screen.

ShortCuts does not contain or reserve the memory needed to &SAVE a screen. If the programmer intends to use these commands, it is his responsibility to make available the 1024 bytes of memory required by these commands. The memory used must be located between the end of the Applesoft program and LOMEM. The following Applesoft command may be used to reserve the required memory:

```
10 LOMEM: PEEK(175) + 256 * (PEEK(176) + 4)
```

In order to avoid overwriting any numeric variables, this command must be used before any numeric or array variables are used or dimensioned.

## **&SCRN**

&SCRN (aexpr)

During its initialization sequence, ShortCuts attempts to determine the type of display device currently being used. Based on this attempt, it sets its screen width to either 40 or 80 columns. If a non-standard DOS is being used, or the display device is changed after initialization, it may be necessary to reset the screen width value.

&SCRN (N) sets the display width to N columns. N may be in the range 0 to 255, but only values in the range 40-80 are practical. Setting the screen width less than 40 will cause STRING TOO LONG ERRORS when some user error messages are printed. As there are no display cards which support a display of more than eighty columns, values greater than 80 for screen width are not useful.

ShortCuts uses the screen width value only when centering messages at the bottom of the screen and when positioning the cursor with the &TAB and &POS commands. The screen width value has no other function.

## **&PR#**

&PR# aexpr

During initialization, ShortCuts attempts to discover in which peripheral slot the output device is located. The normal Apple output is considered to be in slot 0. An 80-column display card might be in slots 1-7. Unless a non-standard DOS is being used, there will be no need to inform ShortCuts of the slot location of the video output, except when changing the output device after initialization.

&PR# N informs ShortCuts that the output device is in slot N. Setting the output slot incorrectly can have very deleterious effects. If the slot is set greater than 7, those effects can include partial erasure of the program in memory.

# CHAPTER 8

## OTHER SECONDARY COMMANDS

### &RND

&RND realvariable [ $>$ aexpr]

If the optional arithmetic expression is present, &RND causes the value of the real variable to be rounded to the number of decimal places specified by the arithmetic expression. For example,

```
10 & RND X > 2
```

causes the value of X to be rounded so that it is accurate to two decimal places.

If the optional " $>$  aexpr" is not used, the real variable will be rounded to the number of decimal places specified by the global decimal setting. (See &PDL in Section 6.3).

If the variable to be rounded is not a real variable, a TYPEMISMATCH error will occur. The arithmetic expression must have a value in the range 0 to 255. If this range is exceeded, an ILLEGAL QUANTITY error will be displayed. If the value of the arithmetic expression (or of the global decimal format, when the aexpr is not used) is greater than 8, no rounding will occur, but no error message will be generated.

Assume the variable to be rounded is X and the number of decimal places specified is given by Y. The formula used for rounding is:

$$X = \text{SGN}(X) * (\text{INT}(X * 10 \uparrow Y + .5) / (10 \uparrow Y))$$

The following table shows the initial values and values after rounding to two decimal places of several numbers:

| Initial value | Rounded value |
|---------------|---------------|
| 0.035         | 0.04          |
| 0.0004        | 0.00          |
| 1.334         | 1.33          |
| -4.233        | -4.23         |
| -0.277        | -0.28         |

In the course of rounding, the real variable is multiplied by 10 raised to the number of decimal places. If the result of this multiplication exceeds 1E38, an OUT OF RANGE error will result, even though the actual value of the variable was within range.

Keep in mind that Applesoft is limited to 9 digits of precision. No numeric variable has a tenth, eleventh or further digit. Attempting to round nonexistent digits is meaningless, but not an error condition.

As numbers reach the limits of Applesoft's precision, rounding may be inaccurate. In general, any number which, correctly represented, would require nine digits—eight '9's and a final digit greater than '7'—may round incorrectly. For example 9.99999998 will round to 10.

### &&

The second ampersand in a ShortCuts statement causes an immediate jump to the machine language program whose address was in \$3F6, \$3F7 at the time ShortCuts was initialized. If you had previously loaded another ampersand utility, this is how you access its commands. If no ampersand utility was present in the computer when ShortCuts was initialized, two ampersands will cause a SYNTAX ERROR.

# CHAPTER 9

## PROGRAM FLOW COMMANDS

ShortCuts contains a set of commands which make possible a higher degree of user-interaction with Applesoft BASIC programs than possible with any version of BASIC currently available. ShortCuts will not allow any control character to be &INPUT to any target variable, whether numeric or string. Instead, control characters are reserved to allow the user, at the programmer's discretion, to trigger predefined program branching.

The program flow directives of ShortCuts allow the programmer to designate selected control characters for the purpose of controlling program branching during &INPUT statements, and to define the branches which may be triggered. The possible uses of ShortCuts' program flow directives will be limited only by the programmer's creativity. Some obvious ideas might include displaying "help" messages, escaping from an input subroutine, or loading a new program module. Creating a program branching option requires the programmer to define both the control character which will trigger the branch and the type of action desired. If only the trigger character has been defined, the control character is partially defined. When the action which it triggers has been defined by the programmer, the character is said to be fully or completely defined.

### **&CONT** (CONTrol table)

**&CONT \$expr**

This command defines a table of control characters which will be recognized during ShortCuts INPUT statements as valid program flow commands. For example, &CONT "DOG" creates a table of legal control characters containing <ctrl>D, <ctrl>O, and <ctrl>G. Subsequent references to the control character table are made by using the number of the character in the table, and not its name. Within this table, <ctrl>D is element number one, <ctrl>O is element two, and <ctrl>G is element three.

The ESCAPE key is seen by ShortCuts as a control character. It is unnecessary for the programmer to define the ESCAPE character as a member of the control character table. Every control character table referenced by ShortCuts contains the ESCAPE character as element number zero.

The string expression following &CONT may be a null string or may contain as many as eight characters. Using a string longer than eight characters will result in a STRING TOO LONG error. The characters in the string should be normal, upper-case, alphabetic characters. They will be converted to their corresponding control characters by ShortCuts. If any of the characters are not alphabetic or not upper-case, they will be entered into the table, but will not be accessible. No error message will be generated.

The <ctrl>M and <ctrl>U characters are identical to the carriage return and the right arrow (typeover) keys. They are not, therefore, usable as triggers for program branching. If included in the table, they are meaningless, but generate no error.

The <ctrl>H is identical to the left arrow (backspace) key. It may be used like any other control character subject to the following rule. If the left arrow (or <ctrl>H) is entered as the first character in response to a ShortCuts' &INPUT statement (i.e., the cursor is at the left edge of the input field), it will be interpreted as a program flow trigger character (providing it has been fully defined). After the first valid character has been entered, the left arrow will operate only as a backspace key.

Multiple CONT commands are not cumulative. This sequence:

```
10 & CONT "A"
20 & CONT "BC"
```

leaves the control character table containing only <ctrl>B and <ctrl>C. <Ctrl>A is lost. Only one control character table may exist at any given time. ShortCuts does not maintain separate global and local control character tables.

The &CONT command only creates the control character table. The characters in the table following the execution of this command are only partially defined. The other commands in this section must be used to assign functions to the characters before they are fully defined. Keyboard entry of a partially defined control character during a ShortCuts &INPUT statement will cause the user error message, ILLEGAL CONTROL CHARACTER, to be displayed, just as if the character were not a member of the table.

**&ON...GOTO...** local and global

**&ON...GOSUB...** local and global

ON aexpr GOTO linnum

ON aexpr GOSUB linnum

where "linnum" is a program line number.

These commands complete the definition of the aexpr<sup>th</sup> character in the control character table by specifying the action that is to be taken if that control character is entered by the user during the execution of a ShortCuts &INPUT statement.

&ON...GOTO... will cause an immediate branch to the line number specified upon the entry of the corresponding control character during an &INPUT statement.

&ON...GOSUB... will cause, when the corresponding control character is entered during an &INPUT statement, a branch to the subroutine specified. The address of the &INPUT statement being executed at the time the branch is called will be placed on the return stack.

Please note this important difference between GOSUBs caused by Applesoft statements and &GOSUBs called by ShortCuts' program flow options. An Applesoft RETURN from GOSUB will reenter the program at the statement following the statement containing the GOSUB. An Applesoft RETURN from a &GOSUB called by ShortCuts will reenter the program at the beginning of the ShortCuts' &INPUT statement whose execution was interrupted by the user directed branch. This return protocol makes it possible for the programmer to allow the execution of subroutines and still be certain of setting the user &INPUT his program requires.

(The ShortCuts &RETURN command can be used to avoid the reexecution of the &INPUT statement, if this feature is not desired.)

Consider the following program segment:

```
10 &CONT "PT"
20 & ON 0 GOTO 500
30 & ON 1 GOTO 100
40 & ON 2 GOSUB 200
50 & INPUT X
```



Line 10 defines a control character table containing ESCAPE, <ctrl>P, and <ctrl>T. (The ESCAPE key is always the zero'th element of any control character table.) Lines 20, 30 and 40 specify the actions to be taken if the control characters occupying positions 0, 1 and 2 are entered by the user during a ShortCuts &INPUT statement.

If, during the execution of line 50, the user presses the ESCAPE key, the program will immediately branch to line 500. If the user enters a <ctrl>P, the program will branch to line 100, as specified by line 30. If the user enters a <ctrl>T, the program will branch to the subroutine at line 200, saving the return address of line 50 on the stack. Returning from the subroutine will reenter the program at the beginning of line 50.

Using ShortCuts' syntax, the above program could have been included in a single statement:

```
10 & CONT "PT", ON 0 GOTO 500, ON 1 GOTO 100, ON 2 GOSUB 200, INPUT X
```

If the arithmetic expression used to identify the position of the trigger character in the control character table is less than zero or greater than eight, an ILLEGAL QUANTITY ERROR will result. If the arithmetic expression is otherwise legal, but is greater than the length of the control character table, the command is meaningless but will generate no error message.

&ON..GOTO and &ON..GOSUB support both global and local settings. If the command is used within a global phrase, the branch option may be invoked during the execution of any subsequent ShortCuts &INPUT statement (unless overridden by a local setting), and will remain in effect until altered by another global program flow directive. If &ON..GOTO or &ON..GOSUB are used within a local phrase, the setting is effective only during the execution of the statement containing the command.

&ON..GOTO and &ON..GOSUB will enable branches only to legal Applesoft line numbers, ie., in the range 0 to 63999. If a line number in the range 64000 to 65535 is specified, the aexpr'th character in the control character table will revert to a partially defined character. The character is not removed from the table, however, but only disabled. If the illegal line number is set globally, the character becomes globally inactive. If done locally, the character is disabled only for the duration of that statement. This is a useful technique when the programmer wants to temporarily remove a program branching option.

&ON..GOTO and &ON..GOSUB do not check to determine if the specified line number exists within the program. If it does not, the error will not be discovered until the branch is attempted. The resulting UNDEF'D STATEMENT error message will contain the line number of the &INPUT statement being executed even though the programming error may have occurred many statements before.

## **&ON...RESTORE**      local and global

&ON aexpr RESTORE

This comand can also complete the definition of the aexpr'th character of the control character table. When this directive has been used, entering the corresponding character during a ShortCuts &INPUT statement causes the &INPUT to be restarted. The input field will be cleared, any partial entry will be lost, the default flag will be reset, and, if the default mode was active, the default value will be redisplayed. Aexpr must be in the range 0 to 8 or an ILLEGAL QUANTITY error will result. It is strongly recommended that any program using default values also define one character in the control character table for this function. This directive may be cancelled only by redefining the control character table or by redefining the character's function with &ON..GOTO or &ON..GOSUB.

## **&RETURN**

### **&RETURN**

Causes a program branch to the statement following the statement whose address was placed on the stack by either an Applesoft or a ShortCuts &GOSUB. (An Applesoft RETURN causes a branch to the statement whose return address is on the stack.)

Applesoft places the address of the statement following the GOSUB on the return stack. In contrast, ShortCuts places the address of the ShortCuts statement being executed when the &GOSUB is triggered on the stack. Therefore, using an Applesoft RETURN to end the execution of a ShortCuts &GOSUB will cause the re-execution of the statement which was interrupted by the &GOSUB. If the re-execution of that statement is not desired, returning via the ShortCuts &RETURN command will skip to the next statement following the calling statement.

If the GOSUB is an ordinary Applesoft GOSUB and return is made via the ShortCuts &RETURN, the first statement following the GOSUB statement will be skipped. As in Applesoft, the number of &RETURNS executed may not exceed the number of GOSUBS executed. The first extra &RETURN will cause a RETURN WITHOUT GOSUB error.

# CHAPTER 10

## SORTING COMMANDS

ShortCuts provides a powerful, versatile and fast tool to satisfy most sorting needs. It can sort real and integer numeric data as well as string data, in either ascending or descending order. ShortCuts can be used to do either conventional sorting or tag sorting. Using a tag sort, the programmer can create multiple key sorts, or sort records from a random text file by some key data item in the record, and then retrieve each record in sorted order.

Defined broadly, sorting is the ordering or arranging of a set of items according to a predefined criteria. In the context of computer programming, this definition is limited by several conventions associated with sorting:

- 1) The items to be sorted are data items.
- 2) The data items are contained in a variable array.
- 3) If the data is numeric, it will be arranged in numerical order.
- 4) If the data is string or character data, it is arranged in the numerical order of the ASCII values of the characters which make up the data. (For data which contains only alphabetic characters, this is the same as alphabetic order.)

Sorts in both ascending and descending order are possible.

It is often desirable to sort only part of an array. For example, an array may have been dimensioned so that it could ultimately contain 300 elements, but only the first 100 elements currently contain data. Specifying the indices of the first and last elements of the portion of the array to be sorted can be called defining the range of the sorted array.

The kind of sorting described so far is conventional, single array sorting. It can be summarized as a series of steps:

- 1) An array is filled with the data to be sorted.
- 2) The range of the array which is to be sorted is defined.
- 3) The order of the sort, whether ascending or descending, is specified.
- 4) The sort is executed.
- 5) After the sort, the elements within the specified range of the array have been rearranged so that they are in the proper numeric or ASCII order.

Conventional, single array sorting is often sufficient for many scientific and statistical programming applications. However, in many other applications, a more comprehensive or involved kind of sorting is required.

Consider a firm's accounts receivable system. On disk are a number of files containing customers' names, outstanding balances and other information. Suppose that the owner of the firm requires a monthly report of customers with unpaid balances, printed in order of largest unpaid balance first.

Using a single array sort, it would be possible to open each file, read each outstanding balance into an array, and sort the array in descending order. But at that point, any connection between the amount of the balance and the name of the customer, and even the number of the file would have been lost, and it would be impossible to generate the desired report.

This accounts receivable problem is typical of a broad range of programming problems involving sorting, in which the main problem is not simply how to order an array of data. Rather, the problem is how to retrieve, or print a number of records containing several associated items—customer name, address, and account balance—so that one of the items is in order, but the other items are still linked to the sorted, or key, item. The tag sort provides one means of doing this.

A tag sort rearranges the values of two separate arrays. The first array, the key array, contains the data which is to be placed in order by the sort. The second array, known as the tag array, is an array of integers which is rearranged simultaneously with the key array, without regard to value. Whenever two elements of the key array must be exchanged to place them in the correct order, the corresponding tag elements are also exchanged.

How can a tag sort be used to solve the accounts receivable outstanding balance report problem? Suppose that we begin reading files on the disk in numerical order. Each time an outstanding balance is found, the balance is saved in a real array, and the record number in the file is saved in an integer array. After reaching the end of the file, four records have been found:

| INDEX | BALANCE | FILE |
|-------|---------|------|
| 1     | 863.00  | 17   |
| 2     | 86.50   | 24   |
| 3     | 198.88  | 26   |
| 4     | 38.75   | 39   |

Now, a tag sort is performed, using the array of account balances as the key file and the array of record numbers as the tag array. This results in:

| INDEX | BALANCE | FILE |
|-------|---------|------|
| 1     | 863.00  | 17   |
| 2     | 198.88  | 26   |
| 3     | 86.50   | 24   |
| 4     | 38.75   | 39   |

All that is now required is to open record number given by Tag(1), that is Record 17, read and report the name and account balance found there, and do the same for the records identified by Tags 2, 3, and 4.

In this way the tag sort can be used to go back to the original data after it has been once sorted. Keeping track of record numbers or of original array subscripts are the two most common uses of tag arrays, but not the only uses. The data contained in the tag array may be anything the programmer desires. It is possible with a bit of programming to create, for example, data reports sorted on multiple keys.

There are two restrictions placed on arrays to be used as tag arrays. First, the tag array must be an array of integers. Second, it must contain at least as many elements as specified by the sorting range.

## **&LIST**

`&LIST aexpr1 TO aexpr2, varname1 [, varname2]`

where "varname" is an array variable name.

The LIST command is an instruction to sort the data in the array named by varname1.

Aexpr1 and aexpr2 define the range of the array to be sorted. That is, the part of the array sorted will include the aexpr1'th and the aexpr2'th elements and all of the elements between them. If either of the arithmetic expressions is an invalid subscript for the array being sorted, "BAD SUBSCRIPT ERROR" will be reported. If aexpr2 is greater than aexpr1, the sort will be done in ascending order. If aexpr2 is less than aexpr1, the data will be sorted in descending order.

The name of the array to be sorted is given by varname1, and must be the name of a previously dimensioned Applesoft array. If the array has not been dimensioned (either by a DIM statement, or by using it), an "OUT OF DATA ERROR" will be reported. Any type of array—real, integer, string—may be sorted.

Subscripts are not a part of an array name: "A\$" is the name of an array which contains the element identified as "A\$(0)".

Varname2 is optional. If given, it is an instruction that a tag sort is to be performed. In this case, varname1 identifies the key array and varname2 is the name of the tag array to be used in the sort. Only integer arrays may be used as tag arrays. Attempting to use any other type of array as a tag array will cause a "TYPE MISMATCH ERROR". The name of a tag array must be separated from the name of the key array by a comma.

Like the key array, the tag array must include the elements whose indices are given by aexpr1 and aexpr2. If the tag array does not include the sorting range, a "BAD SUBSCRIPT ERROR" will result.

Some programming examples:

```
10 & LIST 1 TO 125, A$
```

will cause the string data included in A\$(1)...A\$(N)...A\$(125) to be sorted in ascending order, according to the ASCII values of the characters in the strings. The zero'th element and any elements with indices higher than 125 will be unchanged.

```
10 & LIST 125 TO 1, A$
```

would sort the same portion of the same array, but leave the data in descending order.

```
10 & LIST 10 TO 80, K, T%
```

will perform a tag sort on the 10th through the 80th elements of the arrays named K and T%. In this case, the real array, K, is the key or sorted array, and T% is the tag array. Because the lowest index is given first, the sort will be done in ascending order.

The &LIST command is considered a primary action command. Its operation is independent of the setting of any of ShortCuts' global or local parameters.

The sorting method used in ShortCuts was chosen to provide the best combination of versatility, memory conservation, and speed. The algorithm used is technically known as a Shell Sort. Typically, ShortCuts will sort a single array of 300 random real numbers in 2.2 seconds. A tag sort of 300 random reals will require about 2.4 seconds. Integer arrays and most string arrays of the same size will require less sorting time. In general, any degree of presorting will decrease the required time to sort the array.

# CHAPTER 11

## USING ShortCuts WITH DISK FILES

ShortCuts was designed primarily to support keyboard input/video output interactive programming. But it is also possible to use ShortCuts to read from and write to disk files. The results can be both interesting and useful.

### 11.1 &INPUTTING DATA FROM TEXT FILES

If a text file has been opened and is being read at the time a ShortCuts &INPUT statement is executed, the input will come from the disk. As each character is received, it will be printed in the input field, exactly as if it had been typed at the keyboard. When the carriage return ending the entry is received, ShortCuts will display the data in the output screen field fully formatted and justified according to the current setting of the applicable parameters.

How can this be utilized by the applications programmer? It means that the same subroutine used to enter the data can be used to retrieve and display the data! Entire screens of data can be read from a text file and displayed this way. The demonstration program, ADDRESS BOOK, uses this technique when recalling old files.

Should you wish to make use of this technique, take careful note of the following:

- 1) Most character by character checking of the entry is suspended when the target variable is numeric and ShortCuts senses that the input is coming from a disk file.
- 2) The testing of validation criteria, whether implicit or explicit, is not suspended merely because the input is from disk.
- 3) Should an entry not satisfy the validation criteria, the result will, at best, be very ugly.
- 4) "MON I" should not be used to monitor input. The result would be unaesthetic.
- 5) If a text file has been OPENed, and the DOS WRITE command has been issued, but nothing has been printed to the file, ShortCuts may assume that input is coming from the disk.

The following principle will avoid most of the problems which might result from using ShortCuts to retrieve data from a disk file: Use ShortCuts to retrieve only that which ShortCuts created in the first place.

### 11.2 &PRINTING DATA TO A TEXT FILE

ShortCuts may also be used to write data to a text file. This may be useful if a fully formatted, or fixed length representation of the data is needed in the text file. One possible situation where this might be useful would be an application which needed to retrieve (originally) numeric data as strings. Using ShortCuts' &PRINT statement to write the data to disk could avoid retrieving "8E-03" instead of ".003".

If you are considering using ShortCuts to write to text files, take note:

- 1) Always use the &PLOT command to curtail the multiple carriage returns etc., which ShortCuts uses to keep a clean screen.
- 2) Remember that right justification, in a screen field of length N, is accomplished by printing sufficient spaces before the data so that the last character of data is the N'th character printed. This may be desirable if a fixed length file entry is needed.
- 3) If ShortCuts' fixed length, formatted string capability (see the &FOR command) is used to PRINT to a text file, the entry in the text file will contain the formatting characters specified by the format mask.
- 4) A data string thus &PRINTed with a format mask cannot later be retrieved using the same format mask. (Unless the mask contains no edit characters.)

# CHAPTER 12

## OPERATING ENVIRONMENT

The standard operating environment for ShortCuts consists of an Apple II Plus or Apple IIe computer, DOS 3.3, and the normal, 40 column, Apple video display. In this environment, all of ShortCuts' features should operate correctly, without surprises.

However, as time goes on, it seems that fewer and fewer Apples present that standard environment. Therefore, ShortCuts has been designed to operate correctly in many of the possible Apple configurations. Nevertheless, the great variety of disk operating systems, display devices and utility programs in current use make it impossible to guarantee or predict correct operation in non-standard environments. In this section, the possible effects of non-standard Apple environments on ShortCuts are discussed.

### 12.1 NON-STANDARD DISPLAY DEVICES

The standard display device is the Apple monitor video driver. The typical non-standard display device is an 80-column display card. ShortCuts was designed so that it could be used with 80-column display devices as well as with the standard Apple display. However, the number of 80-column display cards available and the different methods they use to control the display make it impossible to assure proper operation with every card.

To determine whether ShortCuts is compatible with a given display card, load ShortCuts and type in this one line program:

```
10 & INPUT, TAB(12, 15), X
```

Run it and check to see if the cursor is halfway down the screen and indented 15 characters. Now type a <ctrl>C. The user error message should be on the 24th line. Next, tap the space bar and verify that the cursor returned to its original position. If so, the display card is probably compatible with ShortCuts. You might want to run the demonstration programs, CALCULATOR and ADDRESS BOOK to check further.

(If the display scrolled when the error message was printed, shorten the ShortCuts display width with the &SCRN(xx) command.)

Many 80-column display cards act as keyboard drivers as well as display devices. If this is the case, the programmer should note the following limitations when using ShortCuts with that display card:

- 1) The user may still be able to move around the screen at will, using the ESC editing keys. If so, truly indestructible screens cannot be created.
- 2) If the display card intercepts the ESC or any control characters, those characters cannot be used to control program branching.

A little testing will determine just what is possible with the 80-column display card you are using.

## 12.2 PROGRAMMING UTILITIES

There are few Apple programmers around who do not use one of the many Apple programming utilities available.

ShortCuts has been tested with PLE<sup>1</sup>, GPLE<sup>2</sup>, and ES-CAPE<sup>3</sup>. Although ShortCuts is not incompatible with any of these utilities, two of them do pose special problems for the programmer who is testing and editing his program at the same time. Those problems and some solutions are the subject of this chapter.

First, if GPLE is present on the computer when ShortCuts is loaded, all should go well. GPLE does not seem to interfere with ShortCuts in any way.

PLE, on the other hand, does pose a problem, but only a small one. During the execution of an input statement, PLE accepts any control characters verbatim, and passes them along to the program.

1. and 2. Program Line Editor and Global Program Line Editor from Synergistic Software.
3. ES-CAPE from S-C Software.

However, PLE still intercepts the ESC key and tries to print the escape function. If the program being edited used the ESC key to control program branching (like the CALCULATOR demonstration program), this function will be difficult to test while PLE is active.

The fix is to issue an IN#0 command before running the program, thereby removing PLE's input hook. When it's time to edit again, pressing reset will restore PLE's intercepts. Should you tire of typing "IN#0," you can create an escape function to type it for you.

The ES-CAPE program editor poses a completely different sort of problem. Each time ES-CAPE is activated, it replaces its own address in Applesoft's ampersand handler \$3F6,3F7, disconnecting and completely losing ShortCuts. When the program being edited is next executed, therefore, the first ShortCuts command restarts ES-CAPE.

The solution lies in restoring ShortCuts' ampersand address before running the edited program. First, set ES-CAPE loaded, then load and run ShortCuts' LOADER program. Now type "PRINT PEEK(1014)" and "PRINT PEEK(1015)" to find ShortCuts' address, and make a note of the result. Pressing reset will reactivate ES-CAPE. You may load and edit your program. Before executing it, use POKE to replace ShortCuts' address in locations 1014 and 1015.

If you are doing a lot of switching between editing and running, it is convenient to create a keyboard macro to poke the correct address back into place.



## **APPENDIX A: INITIALIZATION PARAMETERS**

When ShortCuts is first initialized, the following global parameters are in effect:

- 1) Numeric output is right justified.
- 2) String output is left justified.
- 3) The screen field length for numeric input and output is 11 characters.
- 4) The screen field length for string input and output is 20 characters.
- 5) Numeric variables are displayed with and rounded to two decimal places.
- 6) The default input mode is inactive. New entries are required.
- 7) Signed numbers are allowed in numeric inputs.
- 8) No calculations are allowed during numeric inputs.
- 9) Inverse highlighting of the input field is not used.
- 10) The program branching control character table is empty.
- 11) Following the acknowledgement of a user error message, a line of blanks will be printed on the 24th line of the display.
- 12) Null entries are not considered valid responses to &INPUT statements.

Any of the above aspects of ShortCuts' operation may be changed with the appropriate ShortCuts command.

## APPENDIX B: GLOBAL AND LOCAL PARAMETERS

Most of ShortCuts' parameters can be set either locally or globally.

A locally set parameter is in effect only during the execution of the statement which contains the parameter setting command. A globally set parameter is in effect until overridden by another global parameter setting command.

Correctly choosing when to set a parameter locally and when to set it globally will result in a shorter program. For example, in applications programs where most of the numeric input and output is in dollars and cents, the decimal format (see PDL) should be set globally to two (2). In those instances where two decimal places would be inappropriate, the decimal format should be changed locally.

Any parameter setting command which may be used globally will set a global parameter if it 1) occurs in a statement which contains no primary action command, or 2) is placed in the statement before the primary action command.

The following statements all set the screen field length and decimal format parameters globally:

```
10 & LEN #(15), PDL(2)
20 & LEN #(12)
30 & PDL(4)
40 & LEN #(16), INPUT X
50 & PDL(3), PRINT X
```

If the parameter setting command is one which can set a strictly local parameter, the parameter will be set locally if the command is placed between the primary action command verb and its modifiers.

In the following statements, screen field length and decimal format are set only locally and will revert to their global settings after the execution of the statement:

```
100 & INPUT, LEN (14), PDL(0), X
200 & PRINT, PDL (2), LEN (5), X
```

## APPENDIX C: USER ERROR MESSAGES

When, during a response to an &INPUT statement, ShortCuts detects an error in the entry, or determines that the data does not satisfy programmer-defined validation criteria, the following events occur:

- 1) The computer beeps.
- 2) A user error message is printed at the bottom of the display screen.
- 3) The keyboard (except for the space bar) is disabled.

To continue, the user must acknowledge the error by pressing the space bar.

The system-supplied user error messages and the situations in which they will appear are given below.

### **ENTRY TOO LONG!**

Occurs when the user has typed one more character than the screen field can contain. (See the &LEN command.)

### **COMPLETE ENTRY REQUIRED**

If, during the response to an input of a fixed length, formatted string, the user presses the carriage return before entering as many data characters as are given by the format mask, this message will be displayed. (See the &FOR command.)

### **ILLEGAL CONTROL CHARACTER**

Attempting to enter any control character other than a fully defined program branching trigger character will cause this message to be displayed. (See the &CONT, &ON... GOTO, &ON... GOSUB and &ON... RESTORE commands.)

### **ALPHABETIC ENTRY REQUIRED**

Occurs when, during either a variable or fixed length string entry, the programmer-defined valid character group is Group L or Group A, and the user attempts to enter a character outside that group. (See the &ASC command.)

### **ENTRY MUST BE NUMERIC**

If the target variable of the input statement is numeric, and no calculations are allowed, the entry of characters other than digits, the decimal point, and the space, will cause this error to occur. (See the &VAL command.)

If arithmetic calculations are allowed, the attempted entry of characters other than digits, decimal point, spaces, arithmetic operators, and parentheses, will cause this message to be displayed. (See the &FN command.) If the target variable is a string variable and the valid character group is Group N or Group D, this message will be displayed if the user attempts to enter a character outside of the valid character group. (See the &ASC and &FOR commands.)

### **INVALID NUMBER**

This message will be displayed when the user tries to enter two decimal points if the target variable is numeric, and no calculations are allowed. If the entry of signed numbers is prevented by the use of the &ABS command, attempting to enter a plus or minus sign will also generate this message. (See the &ABS, &ASC, and &VAL commands.)

**ENTRY MUST BE WHOLE NUMBER**

This user error message will be generated if the user attempts to enter a decimal point and the target variable is an integer variable or it is a real variable and the decimal format has been set to zero. (See the &PDL command.)

**ONLY 9 DIGITS ALLOWED**

If the target variable is numeric and calculations are not allowed, the entry of the first digit in excess of Applesoft's limit of precision will cause this message to be reported.

**ONLY N DECIMAL PLACES ALLOWED**

When the target variable is a real variable, and the decimal format has been set to N, the entry of the N+1th digit after the decimal point will cause this user error message to be generated. (See the &PDL command.)

**MUST BE SMALLER THAN +/- NNNNNNNN**

If the target variable is numeric, and the value of the entry is in excess of either the maximum practical magnitude or the programmer-defined input magnitude limiter, the error will be reported with this message. The N's in the message will be replaced with the correct magnitude limit. (See the &EXP command.)

**OUT OF RANGE: -32767 TO 32767**

Should the target variable be an integer variable, and the user's entry be outside the range of values supported by integer variables, this error will be reported.

**OUT OF RANGE: +/- 10 37TH POWER**

When the target variable is a real variable and the user's entry has a value outside the range of values supported by Applesoft real variables, this error message will be displayed.

**INVALID EXPRESSION! CHECK SYNTAX**

If the target variable is numeric, and calculations are supported, an entry which contains Applesoft syntax errors will cause this message to be generated.

**DIVISION BY ZERO ERROR!**

When calculations are allowed with a numeric target variable, and the user's entry causes a division by zero error, the error will be reported.

**INVALID ENTRY**

If a programmer-defined validation criteria is not satisfied by the user's entry, this is the error message which will be displayed in the absence of a programmer-defined error message. (See the &INPUT command.) If, during the execution of an &INPUT for a fixed or variable length string when the valid character group is Group B or Group X, the attempted entry of a character outside the valid character group will cause the "INVALID ENTRY" user error to be reported. (See the &ASC and &FOR commands.)

# APPENDIX D

Getting ShortCuts installed in your Apple is most easily accomplished by RUNning the "LOADER" program on the program disk. Eventually, however, you will want to append ShortCuts to one of your own application programs. Or you may need to place it somewhere in memory other than at HIMEM. This section provides the information you will need to do that.

There are actually three versions of ShortCuts:

- 1) The complete version, called "SHORTCUTS: COMPLETE", which supports all of ShortCuts' commands as described in this manual.
- 2) A somewhat shorter (667 bytes) version named "SHORTCUTS: INPUT/OUTPUT" which supports all of ShortCuts' commands except the sorting command, &LIST. This version is provided for the convenience of programmers who have no need to sort data and would rather have the extra memory.
- 3) A version called "SHORTCUTS: SORTING". This version supports only two commands: & LIST and &&. This version can be used only to perform sorting or to transfer program control to another ampersand utility.

Each version of ShortCuts is also available in two forms: the program development, or HIMEM, form, and the program appendable, or LOMEM, form. On the program disk, the LOMEM forms of the three versions are identified by the suffix, ".APND".

During the development—writing, editing, debugging—of an applications program, the best location for ShortCuts is in the upper part of the Apple's memory, above HIMEM. In this location, adding and deleting program lines, or running the program will not displace or overwrite ShortCuts. For this reason, during the development of a program, the programmer should use the HIMEM form of ShortCuts. This is the form which is installed by the LOADER program. If any other ampersand or USR machine language utilities are needed, they should be installed before installing ShortCuts so that ShortCuts will not be overwritten.

## Appending ShortCuts

When, finally, an applications program has been finished and is ready to be used, running the program should be no more complicated than booting the disk and running its greeting program or typing "RUN APPLICATION". Although it is possible to install ShortCuts at HIMEM from an Applesoft program, it is far easier to make ShortCuts a part of the applications program. This is the purpose for which the LOMEM, or appendable, forms of ShortCuts were created.

Any of the three versions of ShortCuts may be appended to an applications program by doing the following:

- 1) Load the applications program.
- 2) Put the ShortCuts program disk in the disk drive.
- 3) Type "BRUN APPENDER".
- 4) Select the version of ShortCuts to be appended to the application program and wait a few seconds while it is being installed.
- 5) Save the applications program (before running it!) using a new file name.

ShortCuts will now have been attached to that applications program, and it can simply be RUN. There is no need for any separate ShortCuts file, or any need to install it!

During the appending process, several things happened. The ShortCuts language was attached to the end of the program, and its internal addresses were corrected with the self-relocating sub-program. One Applesoft line (line number 0) was added to the program. That line is required to initialize ShortCuts each time the program is run.

The program with ShortCuts appended must be saved before it is run because, after it has been run, the initialization command (line 0) will have converted itself into a REM statement. The programmer should save the appended program under a new file name because it is not practical to remove ShortCuts from a program once it has been appended. If any future program modifications are to be possible, they must be made to a copy of the program which has not had ShortCuts appended to it.

### **Loading ShortCuts in Other Locations or from a Program**

In most programming situations, ShortCuts may be installed most easily by means of the LOADER AND APPENDER programs. Unusual situations may, however, make it necessary for the programmer to load ShortCuts from his own program.

The HIMEM forms are much easier to install and initialize. In fact, all that is required to install and initialize these versions is to BRUN them at an address which allows them sufficient memory. Therefore, these forms should be used whenever the programmer needs to write his own installation program.

The HIMEM versions of ShortCuts may be successfully installed anywhere in RAM provided that 1) the address at which it is BRUN is greater than the end of screen memory (\$07FF) and 2) there is enough memory between that address and HIMEM for the program.

How much room is required? The pre- and post-initialization lengths of the three versions of ShortCuts are as follows:

- 1) SHORTCUTS: COMPLETE has a total length of 5188 (\$1444) bytes. After initialization, it is shortened to 4602 bytes.
- 2) SHORTCUTS:INPUT/OUTPUT has a total length of 4521 (\$11A9) bytes, and is shortened to 3935 bytes.
- 3) SHORTCUTS: SORTING requires 1008 (\$3F0) bytes for initialization, but shortens itself to 682 bytes for running.

(The lengths of the LOMEM versions differ very slightly.)

Although you may place ShortCuts anywhere in memory that you desire (within the limits given above), the preferred location for the language is just below HIMEM. If it is BRUN in this position, the nearly 600 bytes which comprise the initialization procedure are made available for Applesoft variable storage, and HIMEM will be correctly set.

If you choose to install ShortCuts in any location, you should take care to insure that it will not be overwritten during the operation of your program. In addition, you will have to correct the setting of HIMEM, which will have been changed by ShortCuts.

### **Extra Functions in Page 3**

The following short programs will have been loaded into memory page 3 only if neither the ampersand vector nor the USR vector prior to BRUNning ShortCuts pointed to page 3:

APPLESOFT ONERR STACK FIX: Applesoft runtime errors disturb the Applesoft stack. If ONERR GOTO is active and causes a RESUME instruction to be executed, the stack should be fixed prior to the RESUME. This may be accomplished by a "CALL 768".

DISABLE RESET KEY: A "CALL 896" will change the reset vector so that future RESETS will, usually, do nothing. (Occasionally, if the RESET key is repeatedly pressed, very quickly, an irreversible "hang" of the system may occur. Only turning the computer off and restarting it will eliminate the "hang".

REENABLE RESET KEY: A CALL 832" will restore the RESET KEY vector to its state at the time ShortCuts was initialized.

# APPENDIX E - COMMAND REFERENCE

## **&INPUT** Input command

& [global commands,] INPUT [, local commands,] [prompt;] target variable [validation criteria] [,error message]

## **&PRINT** Print command

& [global commands,] PRINT [, local commands,] [string message;] variable [;]

## **&TAB** (aexpr1, aexpr2)

position input/output field at coordinates specified by aexpr1, aexpr2

## **&POS** (aexpr1, aexpr2)

position output field only to coordinates specified by aexpr1, aexpr2

## **&LEN** [/#/\$] (aexpr)

set numeric or string input/output field to length aexpr

## **&LEFT\$** [/#/\$]

left justify numeric or string output

## **&RIGHT\$** [/#/\$]

right justify numeric or string output

## **&NORMAL**

set input field to normal character mode

## **&INVERSE**

set input field to inverse characters

## **&PLOT**

restrict functions that would adversely affect printer or disk output

## **&GR**

reenables normal output functions to screen

## **&DEF**

allow DEFault entry

## **&NEW**

force new entry (opposite of &DEF)

## **&TRACE**

preserves input field for trace-over

## **&OR**

Optional Response, allows null entries

## **&GET**

disallows null entries

## **&STEP**

generates automatic RETURN at end of input field

**&PDL (aexpr)**

Position DecimaL, sets decimal rounding to aexpr places

**&EXP (aexpr)**

sets the magnitude limit of input to 10 to the aexpr

**&ABS**

allows unsigned numeric entry only

**&SGN**

allows signed numeric entry

**&VAL**

allows actual numeric values only to be entered

**&FN**

allows simple computations to be entered

**&SIN**

allows all ScieNtific functions to be used in computation entry

**&ASC (string/svar)**

defines ASCII character classes allowable for string input

**&FOR (string/svar)**

allows input and output to be made through a FORmat mask

**&HOME**

80-column HOME command

**&HLIN \$expr**

puts output on 24th line, and save output during error messages

**&SAVE**

saves text screen to a buffer

**&RECALL**

recalls text screen from buffer

**&SCRN (aexpr)**

sets screen width to aexpr

**&PR# aexpr**

sets video output to slot aexpr

**&RND var [ $>$  aexpr]**

RouNDs a variable to current &PDL value or aexpr

**&&**

calls another & utility

**&CONT \$expr**

sets control character table to the characters in \$expr

**&ON aexpr GOTO linum**

jumps to linum if the aexpr'th control character is entered during &INPUT



**&ON aexpr GOSUB linum**

jumps to subroutine at linum if the aexpr'th control character is entered during &INPUT

**&ON aexpr RESTORE**

restarts input if the aexpr'th control character is entered during &INPUT

**&RETURN**

returns from an &ON..GOSUB at the following line

**&LIST aexpr1 TO aexpr2, varname1 [, varname2]**

sorts array varname1 from aexpr1 to aexpr2; optionally sorts integer array varname2 in same sequence

**CALL 896**

Disable RESET

**CALL 832**

Enable RESET

**CALL 768**

ONERR stack fix