

**SPEED Text Editor
(AOS and AOS/VS)**

User's Manual

SPEED Text Editor (AOS and AOS/VS)

User's Manual

093-000197-03

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085-series) supplied with the software.

Ordering No. 093-000197
© Data General Corporation, 1976, 1977, 1978, 1980
All Rights Reserved
Printed in the United States of America
Revision 03, December 1980
Licensed Material - Property of Data General Corporation

NOTICE

Data General Corporation (DGC) has prepared this manual for use by DGC personnel, licensees, and customers. The information contained herein is the property of DGC and shall not be reproduced in whole or in part without DGC prior written approval.

DGC reserves the right to make changes without notice in the specifications and materials contained herein and shall not be responsible for any damages (including consequential) caused by reliance on the materials presented, including but not limited to typographical, arithmetic, or listing errors.

SPEED Text Editor
(AOS and AOS/VS)
User's Manual
093-000197-03

Revision History:

Original Release - April 1976
First Revision - April 1977
Second Revision - June 1978
Third Revision - December 1980 AOS SPEED 3.20
AOS/VS SPEED 1.10

A vertical bar or an asterisk in the margin of the pages in Chapter 7, "A SPEED Dictionary," indicates substantive change or deletion, respectively, from revision 02.

The following are trademarks of Data General Corporation, Westboro, Massachusetts:

U.S. Registered Trademarks		Trademarks		
DATAPREP	NOVA	AZ-TEXT	ECLIPSE MV/8000	SWAT
ECLIPSE	SUPERNOVA	DASHER	microNOVA	XODIAC
INFOS		DG/L		

Preface

This manual shows you how to compose and edit text using the Text Editor SPEED. It assumes that you are ready to call SPEED with the Command Line Interpreter (CLI) of the Advanced Operating System (AOS) or Advanced Operating System/Virtual Storage (AOS/VS).

Related Manuals

If you have never worked at a terminal or used a text editor before, you may find it helpful to read the first four chapters of *Learning to Use Your Advanced Operating System (AOS)* (69-000018). If you are not familiar with the CLI, consult the *Command Line Interpreter (CLI) User's Manual (AOS and AOS/VS)* (093-000122).

How to Use This Manual

If you are a beginner, this manual takes you, chapter by chapter, through the steps you need to master SPEED. Each chapter contains explanations, examples, and exercises. Each chapter builds on the preceding ones. As you work, keep the manual at your terminal. Read it thoroughly, practice with the examples, and work through the exercises.

If you have experience programming or using a text editor, you may wish to use this manual as a reference work rather than as a tutorial. If so, turn directly to Chapter 7, *A SPEED Dictionary*; it contains information about SPEED in a compact form.

You will find change bars in the margins of some entries in Chapter 7. These bars show changes in functionality, including enhancements, from the previous version of SPEED.

About the Exercises

Your ability to work with SPEED will grow quickly if you keep practicing with SPEED. Reading through the exercises and examples helps less than working through them patiently at the terminal. Don't worry about making errors. You cannot harm the system, and *the more mistakes you make while learning SPEED, the fewer you will make later when you actually use it*. If the first exercise in a group puzzles you and, after several tries, you feel that you cannot work it, consult the answer and explanation at the end of the chapter.

Then go back, work through the first exercise, and try a second one.

About the Dictionary

Chapter 7, the dictionary, contains an entry for each SPEED command, console control key, switch, symbol, and template control key. As you work through this manual, consult these entries to confirm that you understand the relevant parts of each example or exercise. Don't worry if some parts of the entry at first make no sense to you. Things will fall into place as you learn more about SPEED. After you work through the manual, you can use Chapter 7 as a reference tool for looking up information quickly.

Organization of the Manual

Chapter 1 introduces you to SPEED. It describes the use of the terminal and presents the format of SPEED commands.

Chapter 2 shows you how to perform elementary editing tasks by forming simple command lines and executing them with the control key CTRL-D.

Chapter 3 shows you how to handle more than one text file and use more than one buffer at a time.

Chapter 4 shows you how to perform more complicated editing tasks using numbers, variables, and command modifiers. You also learn how to use control keys other than CTRL-D.

Chapter 5 shows you how to make a command or string of commands repeat a number of times and how to set up conditions for the execution of commands.

Chapter 6 shows you how to create and store your own commands and command strings for editing tasks that you are likely to encounter frequently.

Chapter 7 is your SPEED dictionary.

Appendix A exhibits the ASCII character set.

Appendix B interprets SPEED error messages.

Appendix C is your SPEED code graph.

Appendix D contains a functional analysis of SPEED commands.

Reader, Please Note:

We use these conventions for command formats in this manual:

COMMAND required *[optional]* ...

Where	Means
COMMAND	You must enter the command (or its accepted abbreviation) as shown.
required	You must enter some argument (such as a filename). Sometimes, we use: $\left. \begin{array}{l} \text{required}_1 \\ \text{required}_2 \end{array} \right\}$ which means you must enter <i>one</i> of the arguments. Don't enter the braces; they only set off the choice.
<i>[optional]</i>	You have the option of entering some argument. Don't enter the brackets; they only set off what is optional.
...	You may repeat the preceding entry or entries. The explanation will tell you exactly what you may repeat.

Additionally, we use certain symbols in special ways:

Symbol	Means
␣	Press the NEW LINE or RETURN key on your terminal's keyboard.
□	Be sure to put a space here. (We use this only when we must; normally, you can see where to put spaces.)
—	Press the TAB key on your terminal's keyboard.

All numbers are decimal unless we indicate otherwise; e.g., 35₈.

In the text and in examples, we distinguish between your entries and system responses. We use

THIS TYPEFACE TO SHOW YOUR ENTRY\$\$

THIS TYPEFACE FOR THE SYSTEM RESPONSE.

! is the SPEED prompt. It indicates that SPEED is ready to accept commands.

By convention, we cite alphabetic SPEED command names in uppercase letters in the examples. In fact, SPEED accepts command names both in uppercase and in lowercase. Use whichever is convenient.

End of Preface

Contents

Chapter 1 - An Introduction to SPEED

What Is SPEED?	1-1
What Can SPEED Do?	1-1
What Does SPEED Require?	1-1
The Terminal Display	1-1
Getting a Display	1-1
What SPEED Displays	1-1
The Prompt and the Character Pointer	1-2
The Terminal Keyboard	1-2
Uses of Keys	1-2
Console Control Keys	1-3
Distinguishing Keys	1-3
Text	1-4
Characters	1-4
Lines	1-5
Pages	1-5
Strings	1-5
Windows	1-5
Files	1-5
Buffers	1-5
Commands and Command Lines	1-5
Building a Command Line	1-6
The Ins and Outs of SPEED	1-7
Making Errors in SPEED: Don't Worry	1-7
Correcting Typing Errors	1-7
Cancelling a SPEED Command Line	1-7
Inadvertent Commands	1-7
Finding Your Status	1-7
Aborting SPEED	1-7
Exercise	1-7

Chapter 2 - Elementary Editing with SPEED

Entering SPEED	2-1
Opening a File	2-1
Reading in a Page	2-1
Displaying Text	2-1
Creating a File	2-1
Inserting Text	2-1
How to Insert Text	2-1
What to Count as a Character	2-2
Where to Insert Text	2-2

Correcting Text	2-2
Moving the CP from Line to Line	2-2
Moving the CP across Characters	2-3
Deleting Characters	2-3
Searching for Text	2-3
Changing Text	2-3
Killing Lines	2-3
Inserting Text	2-4
Checking the Status of the Buffer	2-4
Typing Out Text	2-4
Finding the Status of Lines and Characters	2-4
Checking the Status of Files	2-5
Ending a SPEED Session	2-5
Writing to a File	2-5
Closing Files	2-5
Exiting	2-5
Exercise	2-5

Chapter 3 - Using Files and Buffers in SPEED

Using Global Files	3-1
Opening Global Files	3-1
Reading from Files	3-1
Writing to Files	3-2
Closing Global Files	3-2
Using Multiple Buffers	3-2
Copying to Another Buffer	3-2
Switching to Another Buffer	3-3
Killing a Buffer	3-4
Using Local Files	3-4
Local File Commands	3-4
Manipulating Local and Global Files	3-4
Command Precedence	3-5
Checking the Status of Buffers	3-5
Exercises	3-5

Chapter 4 - Intermediate Editing with SPEED

Arithmetic in SPEED	4-1
Numerical Expressions as Text	4-1
Storing Numerical Expressions	4-1
Using Decimal Equivalentents of Characters	4-2
Boolean Functions in SPEED	4-2
Using Variables in Housekeeping	4-2
Modifying Commands	4-3
Conditionalizing the Next Command	4-3
Setting a Temporary Delimiter	4-3
Using the Alternate Radix	4-4
Choosing the SPEED Modes	4-4
Default Argument Mode	4-4
Case Control Mode	4-4
Display Mode	4-5
Window Mode	4-5
Position Mode	4-5
Alternate Radix	4-5
Shift Sensitive Mode	4-6

SPEED Control Characters and Templates	4-6
Expansions to Buffers and Files	4-6
Search Templates and Control Characters	4-7
Executing Commands Across Windows or Pages	4-8
Executing the CLI	4-9
Exercises	4-9

Chapter 5 - The Iteration and Flow of SPEED Commands

Command Loops	5-1
Numerical Control	5-1
Conditional Control	5-1
Nesting Command Loops	5-2
Conditional Execution	5-2
The Execution Conditions	5-2
Branch Over to Label	5-2
Conditional Iteration	5-3
Exercises	5-3

Chapter 6 - Advanced Editing with SPEED

Saving a Command Line	6-1
Tracing a Command Line	6-1
Interacting with SPEED	6-1
Writing SPEED Commands	6-1
Creating and Using SPEED Macros	6-2
Executing a Buffer	6-2
Executing Files	6-3
Using the /I= Switch	6-3
Exercises	6-3
Envoi	6-4

Chapter 7 - A SPEED Dictionary

How to Use this Dictionary	7-1
Entering SPEED	7-1
Entering without a Filename	7-1
Entering with a Filename	7-1
Permanence	7-1
Switches	7-2
Exiting from SPEED	7-2
The Exit Command	7-2
A Note about Line Printer Listings	7-2
Organization of the Entries	7-2
Format	7-2
Function	7-2
Numerical Arguments	7-2
Symbolic Modifiers	7-3
Characteristics	7-3
Precautions and Error Messages	7-3
Related Commands	7-3
Examples	7-3

The Structure of SPEED Commands	7-3
Numerical Arguments	7-3
Symbolic Modifiers	7-4
Command Name	7-4
Search Strings and Text Strings	7-4
Delimiters	7-4
Command Line Terminator	7-4
Convention on Capitals	7-4
A Note on Command Precedance	7-4
Functional Analysis of SPEED Commands	7-4
Entry Sequence in this Dictionary	7-4
A	7-5
BC	7-6
BFB	7-8
BFC	7-9
BFNR	7-10
BFNW	7-11
BFO	7-12
BFR	7-14
BFU	7-15
BFW	7-16
BG	7-17
BK	7-19
BS	7-20
BT	7-21
B?	7-23
C	7-24
D	7-27
/D	7-28
E	7-29
FB	7-30
FC	7-31
FNR	7-32
FNW	7-33
FO	7-34
FR	7-36
FU	7-37
FW	7-38
F?	7-38
H	7-39
I	7-40
/I=	7-42
J	7-43
K	7-44
L	7-46
M	7-47
N	7-48
O	7-49
P	7-51
PW	7-53
Q	7-54
R	7-56
S	7-57
T	7-59
V	7-61
VC	7-62
VD	7-63
VI	7-64
VL	7-65

VM	7-66
VN	7-67
VP	7-68
VS	7-69
WA	7-70
WC	7-71
WD	7-72
WM	7-73
WP	7-74
WR	7-75
WS	7-76
X	7-77
Y	7-79
Z	7-80
\$ (ESC)	7-80
!	7-81
!label!	7-82
n''Xcommand-string'	7-82
#	7-85
&	7-85
*	7-86
+	7-87
-	7-87
.	7-88
/	7-88
:	7-89
;	7-90
< x >	7-91
=	7-93
?	7-94
@	7-95
\	7-96
^*	7-97
^+	7-98
^-	7-98
^/	7-99
_x	7-99
CTRL- \ character-listCTRL- \	7-100
CTRL-Bbuffer-name	7-101
CTRL-C, CTRL-A	7-102
CTRL-C,CTRL-B	7-102
CTRL-D	7-103
CTRL-E	7-103
CTRL-Ffilename.	7-104
CTRL-G	7-105
CTRL-I	7-105
CTRL-J	7-106
CTRL-L	7-106
CTRL-M	7-107
CTRL-N	7-107
CTRL-Q	7-108
CTRL-S	7-108
CTRL-T	7-109
CTRL-U	7-109
CTRL-W	7-110
CTRL-X	7-110
CTRL-Y	7-111
CTRL-Z	7-111

Appendix A - ASCII Character Set

Appendix B - Errors in SPEED

SPEED Error Messages	B-1
--------------------------------	-----

Appendix C - SPEED Code Graph

Appendix D - Functional Analysis of SPEED Commands

Tables

Table Caption

3-1	Input and Output Functions	3-3
3-2	File Handling Commands	3-5
4-1	Boolean Functions.	4-3
7-1	Numerical Arguments to Commands	7-3

Illustrations

Figure Caption

1-1	Keyboard Differences.	1-4
1-2	English-SPEED Differences in Syntax.	1-6
7-1	Command Control.	7-50

Chapter 1

An Introduction to SPEED

Welcome to the SPEED Text Editor! This manual assumes that you are ready to enter SPEED under the Command Line Interpreter (CLI) of the Advanced Operating System (AOS) or the Advanced Operating System/Virtual Storage (AOS/VS). It also assumes, in the first six chapters, that you have little or no experience with programming or other text editors. (If you do have some experience, read the introductory remarks to Chapter 7; if they seem to make sense, try skipping the six tutorial chapters.)

Before you begin your first editing session, you need some general information about the SPEED Text Editor.

What Is SPEED?

SPEED is a character-oriented text editor that can help you edit pieces of text as various as poems and programs -- and you need not be a poet or a programmer to use it. (The author of this manual is neither!)

Editing a text includes

- adding new material
- changing material
- deleting old material
- making copies of the material

An editing task may be as simple as correcting a misspelling, or as complex as writing a manual like this one.

What Can SPEED Do?

SPEED allows you to open existing text files or create new ones. You can create copies or variants of existing files, combine them, or divide them. You can work on several files at the same time. SPEED can give you multiple workspaces or *buffers* so that you can carry out several different tasks at the same time, or one larger task that requires more than one working area. SPEED can undertake repetitive tasks automatically, and you can set conditions for SPEED to check before carrying out a task. With SPEED you can create your own editing programs for use during an editing session, and you can file them as SPEED *macros* for use later.

What Does SPEED Require?

You need very few tools before you start editing with SPEED. If you can type, the initial strangeness of using SPEED will wear off more quickly.

If you haven't done so already, familiarize yourself with the structure of AOS files and directories in the chapter "AOS Terms and Concepts," of the manual *Learning to Use Your Advanced Operating System (AOS)* (69-000018).

The best asset you can bring to your initial SPEED sessions is patience. You may think at first that a madman devised SPEED as an instrument of torture. With patience and practice, your frustrations will turn into an understanding of SPEED's psychology. SPEED was designed for the use of the expert -- one who makes few mistakes. It was also designed to require the minimum number of key strokes from the user, to be as efficient as possible. Therefore, SPEED is compact and unforgiving.

The Terminal Display

An editor needs to be able to look at text, and to change it. When you enter SPEED, it creates a *buffer* or workspace from which you can display the text you wish to edit.

Getting a Display

SPEED does not display text automatically. You must get a display either by issuing T commands or by entering SPEED with the display switch. You will find out how to do these things in your first editing session in the next chapter.

What SPEED Displays

SPEED displays two sorts of things. It shows you

- the *text* that you are currently editing, whether you keyed it in from the terminal or brought it into the buffer from an open file
- the *command line* you type in from the keyboard containing the changes you ask SPEED to make in the text

When both text and a command line are on the screen, the current command line always follows the text.

The Prompt and the Character Pointer

SPEED gives you two different signals as you work. One tells you that SPEED is ready for your commands, and the other tells you where in the text SPEED is ready to work.

The SPEED *prompt* has the form of an exclamation point. On video display terminals, SPEED follows the prompt with the *cursor*, which it represents as a blinking underline on models 6052 and 6053, and as a white rectangle on models D100 and D200. In this manual, we represent the cursor with an underline.

!_

SPEED gives you a prompt when it is ready to accept your commands. It tries to interpret whatever you type as a command line. SPEED *echoes* the line you type in after the exclamation point, moving the cursor to the right. If, for example, you ask SPEED to insert *two words*, SPEED displays

!I □two words_

before you execute the command by entering a CTRL-D.

The SPEED *Character Pointer* has two different forms. If you are working at a terminal model 6052, 6053, D100, or D200, the Character Pointer (CP) is a blinking asterisk. On other models, SPEED represents the CP with a caret in parentheses (^). In this manual, we represent the CP with an asterisk.

The CP shows the point in the text where SPEED tries to work. If, for example, it shows you

type* in

and you successfully ask SPEED to insert □two□words, the result will appear as

type two words* in

The CP lies between two characters, and not on a character. SPEED marks the location of the CP with the preceding character. The first position of the CP when you enter SPEED is 0, at the beginning of the buffer.

The relevant prompt, preceding your current command line, lies below the most recent CP in your text. Keep the distinction between the two constantly in mind: the prompt signals you *what you are doing*, the CP tells you *where you are doing it*.

The Terminal Keyboard

SPEED takes its commands from you by interpreting what you type at the keyboard. The terminal keyboard resembles that of a typewriter, but has additional keys and functions. A key on the alphanumeric keypad may have up to three uses in SPEED.

Uses of Keys

SPEED uses keys and sequences of keys as command names, text characters, and control keys:

- Keys serve as *characters* when you enter them in sequence after a search (C, N, Q, S), insertion (C, I, BG), or display (@T) command
- Keys serve as *command names* when you enter them alone or in sequence and they do not follow a search, insertion, or display command, or occur in a label
- Keys serve as *control keys* when you depress CTRL and another key at the same time. A control key may have one of two uses. When you use it within a search command, it may function as a *template* for portions of the search string. When you use it elsewhere, the control key has its standard CLI function

For example, the N key

<i>Character</i>	is a text key in a command to change the next <i>N</i> in the text to <i>M</i>
CN\$M\$	
<i>Command</i>	is the command name in a command to search throughout the file for the next occurrence in the text of <i>I</i>
NI\$	
<i>Template</i>	is a template control key in a command to find the next character in the text that is <i>not</i> a <i>P</i>
SpNP\$	

Console Control Keys

Control characters allow you to alter the activity of your terminal. To enter a control character, press and hold the CTRL key, then press the other key. In this manual we mention a control character with a preceding CTRL (CTRL-A) and show its echo as ↑A.

CTRL-C, CTRL-A When you key in a ↑C,↑A sequence, you abort the entire current command line, even if it contains NEW LINES.

CTRL-C, CTRL-B When you key in a ↑C,↑B sequence, you abort the entire SPEED editing session. Your input file, if you have one, remains unchanged, but you lose any editing carried out during the session. This is a fast way to kill bad command execution.

CTRL-D This control key echoes as \$\$ rather than ↑D. It terminates and attempts to execute your current SPEED command line.

CTRL-I Keying in ↑I or TAB is the same as issuing an I command followed by a TAB.

CTRL-J Keying in ↑J in an I command equates to inserting a NEW LINE.

CTRL-L Keying in ↑L in an I command equates to inserting a form feed.

CTRL-M Keying in ↑M in an I command equates to inserting a carriage return.

CTRL-O When you key in ↑O, you discard information written to the terminal. Cancel a CTRL-O with a CTRL-Q.

CTRL-P When you key in ↑P, you tell SPEED not to interpret the next character you type. (In search commands, use the sequence CTRL-P, CTRL-T to echo ↑T.)

CTRL-Q When you key in ↑Q, you unfreeze the terminal if you have frozen it beforehand.

CTRL-S When you key in ↑S, you freeze the current display until you key in ↑Q. You do not lock the keyboard, so you may type "ahead."

CTRL-U When you key in ↑U, you cancel that part of the command line following the last NEW LINE.

Distinguishing Keys

You must distinguish certain keys that look alike or appear to function alike. (If you are an experienced typist, this may mean breaking some habits you have acquired on other machines.) You should never substitute for each other

- zero and the letter O
- one and the letters l or I
- the standard dollar sign (\$) and the dollar sign that SPEED echoes the ESC key with
- the caret (^) and the circumflex that SPEED echoes the CTRL key with
- the NEW LINE key, which causes a carriage return and a line feed, creating a new line, and the carriage return key, CR, which merely takes you back to the beginning of the current line

The ALPHA LOCK key acts as a toggle: if you key it in while in lowercase, it shifts you to upper; rekeying it shifts you back to lowercase.

You may use either the numeric keypad or the numbers on the alphanumeric keypad while you are in SPEED. In the console control keypad of terminal models 6052 and 6053, only the TAB and repeat (REPT) keys function (but SPEED has no need of them). SPEED makes no use of the function keypad(s). Figure 1-1 highlights crucial aspects of the keyboard models.

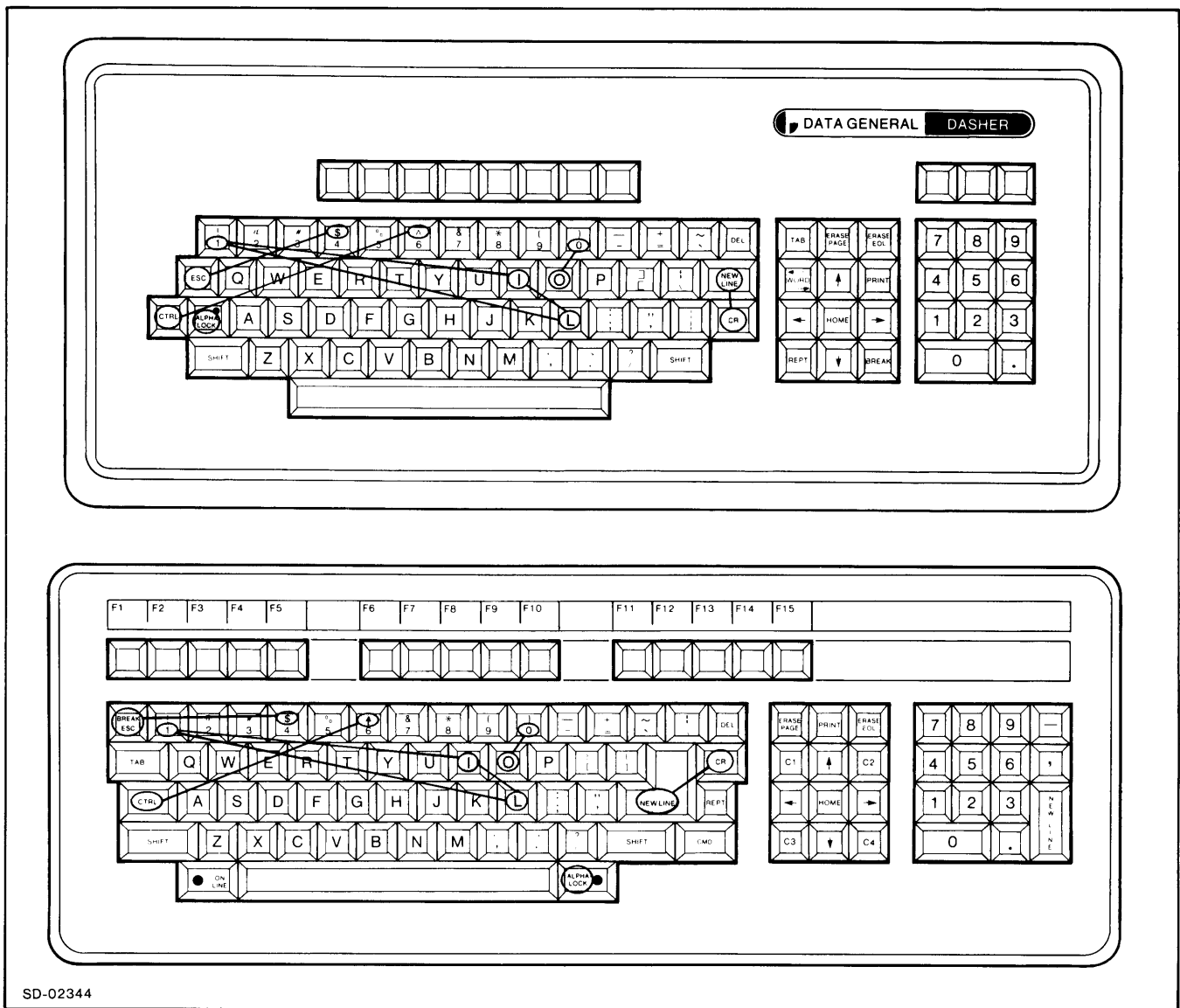


Figure 1-1. Keyboard Differences

Text

For SPEED, text comes in units of different types and sizes.

Characters

Some SPEED commands operate on *characters*. Characters include, as you might expect, letters of the alphabet, the digits 0 through 9, and units of punctuation. SPEED also counts as a single character a

- space
- tab

- carriage return (CR)

- carriage return/line feed, which you insert with a NEW LINE

- form feed, or page break, which you insert with a CTRL-L

Each text character in SPEED has a position. Character positions begin with 0.

Lines

A *line* is a sequence of characters ending with NEW LINE. Each line of text that you type in or SPEED displays echoes as a separate line on the terminal screen.

A sequence of characters followed by a carriage return (CR) does *not* count as a separate line, and SPEED does not display it separately.

Each line in SPEED has a number. Line numbers begin with 1, not 0.

Pages

A *page* is a sequence of characters extending from form feed (CTRL-L) to form feed, or from a form feed to the end of the file. You may think of a SPEED page as the counterpart of the black and white variety, if you wish, but SPEED pages are more supple: you may change their length, split them in two, or combine them by inserting and deleting form feeds.

If you open a paged input file and do not reset your Window Mode, SPEED reads the file into the buffer one page at a time.

Strings

A *string* is any sequence of characters that has as its last character a standard or temporary delimiter. (The standard delimiter is ESC, and its echo is the dollar sign (\$).) A string may contain spaces, tabs, NEW LINES, and form feeds. SPEED uses strings in two ways:

- SPEED tries to match the *search strings* you type into a command line with equivalent strings of text in the current buffer. When it finds one, it repositions the CP after the string and carries out subsequent commands from the new CP position
- SPEED inserts the *text strings* you type in as part of a command line into the text at the position of the CP

When you type in a delimiter, or the CTRL-D command line terminator, you end the string.

Windows

Whether or not your input file is paged, you may read it into the buffer in *windows*, or a specific number of lines at a time that you select with a WM command. When you select a window length, SPEED ignores the page breaks in the file, displaying in the place of each the CTRL-L (␣) you inserted.

If you do not issue a WM command, SPEED reads a file in one page at a time.

You may find certain window modes useful: SPEED can display up to 23 lines on many screens. You can obtain an automatic 20-line display if you enter SPEED with the /D switch. You can simulate a standard 8 1/2- by 11-inch typewriter page with 66 lines. A standard line printer page can hold 63 lines at six lines per inch and not print across the perforations.

Files

Text *files* contain the text you edit. To edit files in SPEED, you must *open* or *create* them. *Input files* contain the text you wish to edit. If you wish to retain the editing you do during a SPEED session, you place your results in one or more *output files*. SPEED has a procedure for using a single file for both input and output, and you may have more than one input and output file open at the same time.

You may have input files open without open output files and vice versa. In fact, you can insert material into the buffer from the terminal with no files open at all. If you have no output file open, of course, you will retain nothing of the editing session after you exit from SPEED.

Buffers

SPEED can, if you wish, create more than one workspace, or *buffer*. You may place different pieces of text in each buffer, and you can copy text from one buffer to another.

You retain access to input and output files from various buffers by opening them *globally*. By using *local* file-opening and file-closing commands, you may use additional files in each buffer.

SPEED can activate, open files, and store text in 36 buffers.

Commands and Command Lines

The sequence of elements in a SPEED command does not always correspond closely to the sequence of words you would use to translate it into English, and you must take precautions in stringing commands together that are peculiar to SPEED. For example,

```
!3:Sform$* 1T$$
```

corresponds to

Search for form in the next three lines, and if you find it, type out the rest of the current line.

See Figure 1-2, which explodes the command string following the SPEED prompt (!).

Because of such syntactic differences, you must learn the structure of SPEED commands very carefully.

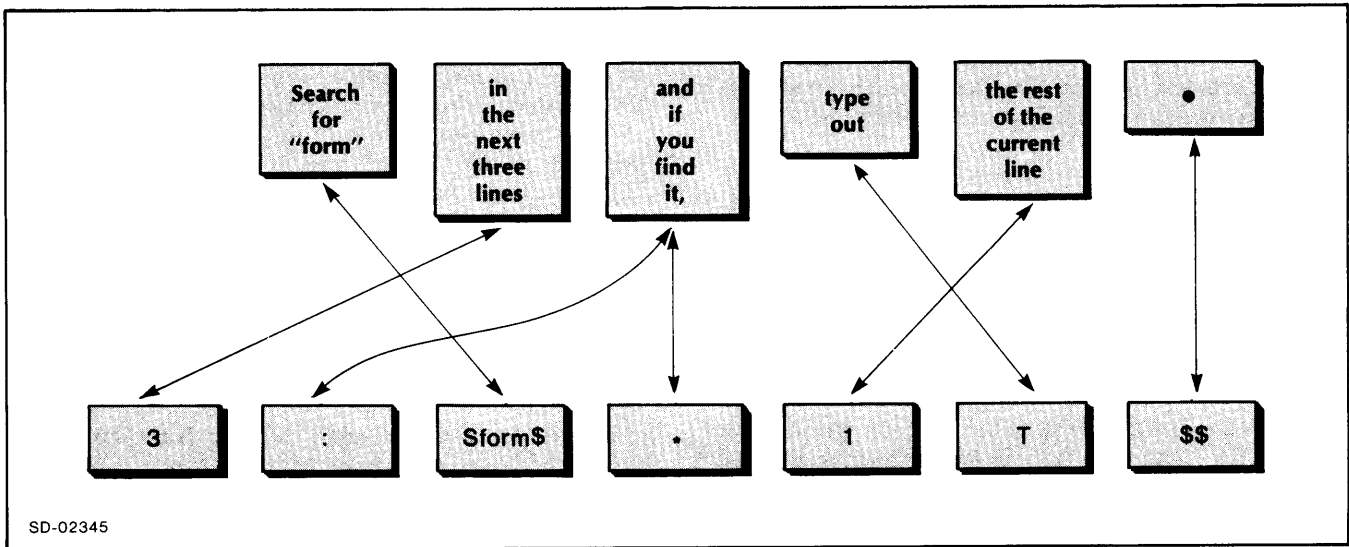


Figure 1-2. English-SPEED Differences in Syntax

Building a Command Line

A SPEED *command line* consists of one or more *commands* and a command line *terminator*. You can construct and execute a SPEED command line by following these rules:

- Rule 1. If the command takes a numerical argument, type it in. Numerical arguments may be positive integers, negative integers, or pairs of integers. SPEED accepts as a numerical argument any numerical expression that it can evaluate to an integer or pair of integers. In SPEED, 0 counts as a positive integer.
- Rule 2. If the command takes symbolic modifiers (@ : &), type them in.
- Rule 3. You may apply rules 1 and 2 in the opposite order, if you wish.
- Rule 4. The command *must* have a command name. Type it in next.

- Rule 5. If the command takes a character string, type it in next. The character string may be either a search string or a text string. If this command is not to be the last one, or if the string is the search string of a C command, enter the ESC delimiter (which echoes as \$) as the final character of the string.
- Rule 6. If you typed in a search string for a C command, and you wish to insert a text string, type it in. If this command is not to be the last one, enter the ESC delimiter to end the string.
- Rule 7. If you are ready for another command, go back to rule 1.
- Rule 8. If you have typed in all the commands you want SPEED to execute, type in the command line terminator - depress CTRL and type D before releasing CTRL.

Most SPEED commands obey these rules exactly. The dictionary in Chapter 7 describes the few commands that these rules do not cover in the section "The Structure of SPEED Commands" and in the individual entries.

The Ins and Outs of SPEED

You may enter SPEED with or without a filename. Type

) X SPEED)

or, if you have a specific file to edit in mind,

) X SPEED filename)

When SPEED shows you its revision number and displays the prompt, you are ready to begin editing.

When you wish to end your editing session, type

!H\$\$ (That is, type H, depress CTRL, type D.)

If, in response, SPEED asks you

Confirm?

then you have text in the buffer, an open file, or both. Type y) to exit from SPEED; type any other character to remain in the editing session.

Making Errors in SPEED: Don't Worry

In your first practice sessions with SPEED, you will make many errors of several different kinds. Go ahead and make them! It is better to make them during practice sessions and learn what to expect than it is to make them later during serious editing sessions. Use these tips to reduce the seriousness of your errors.

Correcting Typing Errors

Erase the preceding character by pressing the DEL or RUBOUT key. On video display terminals, the last character disappears and the CP moves left one space. Hardcopy terminals echo a backarrow or underscore.

You may use the DEL and REPT keys jointly to delete a sequence of characters.

Cancelling a SPEED Command Line

You can cancel an entire command line by issuing a CTRL-U (depress CTRL and type in a U).

If your unexecuted commands range over more than one line, and you wish to cancel them, you may do so by issuing a CTRL-C, CTRL-A sequence. SPEED deletes the command sequence and restores your prompt.

Inadvertent Commands

If, during an editing session, you lose control of the terminal and "nothing seems to work," you may have inadvertently struck a CTRL-S, which freezes the terminal. (It is easy to do this, since S lies next to D,

which you must use constantly to execute command lines.) To regain control of the terminal, issue a CTRL-Q.

It is easy, especially if you are an experienced typist, to forget the I command name preceding the text you wish to insert in the buffer. When you do this, SPEED tries to interpret the line you type in as a command line. If your line begins with H, Q, or Y, SPEED's response may surprise you: SPEED will ask you

Confirm?

Confirm (Q-command)?

Confirm (Y-command)?

when you already have text in the buffer, because you have inadvertently told SPEED to terminate the SPEED session (H) or to clear the buffer (discard its contents) and read in the next page of your input file. Type in any other characters but y) to remain in the session. You may then type in the I command name and begin your insertion anew.

Finding Your Status

It is easy, especially at first, to lose track of your SPEED editing session and forget which buffers you have stored text in and which files remain open. You may find it useful to check on the status of your buffers and files before issuing the H exit command. You can do this quickly by issuing the command line

!F?B?\$\$

SPEED tells you which files are open in the current buffer, and tells you which buffer you are in and which other buffers are active.

Aborting SPEED

If you need to leave the SPEED session and no other method seems to work, you can abort SPEED and return control to its parent process by issuing a CTRL-C, CTRL-B sequence. *You lose the results of your editing session when you do this.* Do not issue this sequence in a panic; try first to think your situation through. You harm nothing by letting SPEED "idle" as you do so.

Exercise

From time to time in this manual, we offer an exercise so that you can practice your SPEED skills. Here's the first one:

Exercise 1-1.

Enter SPEED. Make a note of the revision number.
Check your file and buffer status. Exit from SPEED.

```
) X SPEED)
SPEED REVmm.nn
!F?B?$$
Global:
    Input File - None
    Output File - None
Local:
    Input File - None
    Output File - None
=> Buffer 0 - 0
!H$$
)_
```

SD-02377

Answer 1-1.

End of Chapter

Chapter 2

Elementary Editing with SPEED

Now that you have a grasp of the basic elements of SPEED, you can undertake a simple editing session. If you have little experience with SPEED, please *work through the examples* as you read. As you use each command, look up the entry for it in Chapter 7, "A SPEED Dictionary."

Entering SPEED

Before you enter SPEED, please create, or ask your supervisor to create, a file to practice with. Call it MYFILE. Place some text in it using the CLI, perhaps

Here is a sentence to work with.)

If you're ready, let's begin.

Enter SPEED from the CLI by typing after the CLI prompt

```
) X SPEED)
```

Your screen then displays the SPEED revision number and gives you its prompt.

```
SPEED REV mm.nn  
!
```

SPEED is ready for your first command. Let's create a new file using the contents of the one you already have. MYFILE will be the *input* file, and the file we create will be the *output* file.

Opening a File

To open MYFILE for reading, issue an FR (File Read) command

```
!FRmyfile$$
```

Remember *not* to place a space between the command name FR and the filename, and remember to terminate the command with ESC or with CTRL-D--depress CTRL, type in D, and observe the \$\$ echo.

Reading in a Page

When SPEED restores your prompt, it has carried out your command and opened the file. But it has not yet read any contents of the file into the buffer. Append the first page of the file (in the present case, the entire file)

to the buffer with the A (Append) command. Since your file contains lowercase letters, you will see

```
!A$$
```

```
** Lower case input encountered **
```

SPEED has now appended the first page of the file to the buffer. In the present case, since the buffer was empty, its current contents equal the contents of your file.

Displaying Text

To verify that SPEED has done what you asked it to do, call for a type out with the a T or 1T (Type) command. The CP is at the beginning of the first line of the buffer. Either of the T commands will display everything on that line to the right of the CP. If you have been successful, you will see

```
!T$$
```

```
*Here is a sentence to work with.  
!_
```

Creating a File

Now that you have brought text into the buffer, create the output file that you wish to copy it to. Name the output file NEWFILE with the FW (File Write) command

```
!FWnewfile$$
```

We now have an input file, an output file, and text in the buffer. You are ready to begin editing with SPEED.

Inserting Text

Let's put another sentence in the buffer, so that the contents of the buffer will differ from the contents of the input file. (We are altering the buffer, but not the input file itself.)

How to Insert Text

You make text insertions by typing in the I (Insert) command and the text you want to insert. You follow the text with a *delimiter* (ESC, which echoes as \$,) if you wish to continue the command line. You follow the text with a CTRL-D if you wish to terminate and execute the command line.

Let's type in the sentence and ask SPEED to show us the contents of the buffer at the same time. The symbol # abbreviates the pair of arguments (O,Z), where O stands for the beginning of the buffer, and Z stands for the last character in the buffer. Since T is the Type Out command for a display, a #T command asks SPEED to display the entire buffer. Try, for example,

```
!!Here is a second sentence to type in.)  
$#T$$
```

What to Count as a Character

You typed a NEW LINE (␣) just as you did every other character. SPEED counts as characters not only letters of the alphabet, digits, and punctuation symbols, but also spaces, tabs, NEW LINES, carriage returns, and form feeds. Each of these, even the tab, counts as a *single* character.

Where to Insert Text

The result of your insertion, alas, is

```
Here is a second sentence to type in.  
Here is a sentence to work with.
```

with the sentences in the wrong order. Type out a another display with a T command to see where the CP is and then insert another sentence.

```
!!This is my third sentence in SPEED.)  
$$
```

When you issue another #T command, you see

```
Here is a second sentence to type in.  
This is my third sentence in SPEED.  
Here is a sentence to work with.
```

SPEED made the insertions at the position of the CP. At the time of the first insertion, the CP was at the head of the buffer. At the end of that insertion, it followed the NEW LINE, the last character you inserted. The general rule is that SPEED carries out its operations (insertions, deletions, movements) at the position of the CP, and the CP follows inserted material.

Correcting Text

By now, you are probably wondering how to correct the errors you made when you typed in your insertions. On the off chance that you made none, please insert the following text in the buffer, preserving the intentional errors we have made in it. Don't omit the initial I Insertion command, and don't forget to type in a NEW LINE after each of the periods.

```
Hgere is a setnece fo you tocrrect.  
Please put thge missing wrod.  
Womethg iswrong on this lin.
```

Use a #T (Type) command to verify that buffer now looks something like this:

```
Here is a second sentence to type in.  
This is my third sentence in SPEED.  
Hgere is a setnece fo you tocrrect.  
Please put thge missing wrod.  
Womethg iswrong on this lin.  
Here is a sentence to work with.
```

Your CP should be at the beginning of the last line (use a 1T command to verify that it is). Don't worry if your actual text contains different or additional errors; you're going to learn to correct them all in this editing session.

During the rest of this session, enter #T and T commands at will, in order to see exactly what you are doing as you do it.

Moving the CP from Line to Line

You are eager to correct your errors, but you are not in a position to do so. You need to move your CP so that you can work in the appropriate place. To move to a different line, use the L (Line) command

```
!-2L$$
```

and check the result. You should be at the beginning of the line that is missing a word. L (Line) commands move the CP across the number of NEW LINES you specify. When you give the command name a positive numerical argument, it moves the CP forward in the buffer (down in the display) from its previous position. When you give it a negative argument, it moves the CP backward in the buffer (up in the display) from its previous position. So your command moved the CP to the beginning of the second line preceding your previous position.

If you know which line from the beginning of the buffer you wish to work on, you can use a J (Jump) command instead. Enter a 3J command and check the result. Line 3 in the buffer contains the first of the intentionally incorrect sentences, and you should be at the beginning of that line.

A -1L command takes you to the beginning of the preceding line. A 1L command puts you at the beginning of the following line. An OL command puts you at the beginning of the current line. Since zero is the default value for L commands, you need not type the zero if you want to be at the beginning of the current line; L alone has the same effect.

An OJ, 1J or J command places you at the beginning of the buffer. A ZJ command places you at the end of the buffer.

Moving the CP across Characters

You can move across characters with M commands. Enter a 1M (Move) command and check the result. You should be in front of a spurious letter, g. As with L, positive numerical arguments to the M command carry you forward in the buffer, and negative ones carry you backward.

Deleting Characters

You have now moved to a position from which you can use the D (Delete) command to delete characters. This time you wish to delete one character to the right of the CP, so enter a positive 1D command and check the result. You should see

*H*ere is a setnece fo you tocrrect.*

You can delete characters preceding the CP with -nD commands. You can delete all material preceding the CP with a -D or -ZD command.

Searching for Text

You do not need to specify any specific number of characters or lines in order to reposition the CP. Instead, you can use an S (Search) command. From your present position you can see that you need to make an insertion in the fourth word. Type in the S (Search) command and enough text as a search string to distinguish the position you wish to move to, and check the result, for example

!Sse\$T\$\$

The result should be

*Here is a se*tnece fo you tocrrect.*

An nS command searches within the next n lines. A -nS command searches within the preceding lines. A m,nS command searches from the character following the mth character in the buffer to the nth character. The default value of this command is from the CP forward to the end of the buffer.

An OS command searches from the beginning of the line up to the CP. An 1S command searches from the CP to the end of the line. A #S command searches the entire buffer.

If you do not follow the search string in the command line with a CTRL-D, you must end the string with a delimiter, as you did above.

If you issue the command without a search string argument, SPEED will attempt to use the search string you used in the last search (C, N, Q, S) command. Try it yourself. Enter

!S\$T\$\$

The result should be

Please put thge missing wrod.*

since your preceding search was for se.

Changing text

You can see letters in the wrong order in the last word in this sentence. You can change it by issuing a C (Change) command. A quick glance shows that the sequence of letters you want to change is the first and only such sequence. Type in the C (Change) command, the search string (or string you want to correct), a delimiter, and the text string (or string you want to change it to), and a final delimiter. If you type

!Cro\$or\$T\$\$

you will see

*Please put thge missing wor*d.*

The C command takes numerical arguments in the same way as the S command. If you issue the command without a search string as

!C\$string\$\$

it will try to find another instance of the last thing you searched for with a search command, and change that. If you issue the command without a text string as

!Cstring\$\$\$

it will try to delete the next instance it finds of your searchstring. Try it: tell SPEED to change the only g in the preceding part of the line. If you enter the Change and Type commands

!OCg\$T\$\$

(with two delimiters following g) you should see

*Please put th*e missing word.*

Killing Lines

You may delete entire lines with a K (Kill) command, regardless of how many characters each contains. For example, enter

!2J1K\$\$

and delete the second line, including the NEW LINE character that followed it.

The K (Kill) command takes numerical arguments in the same way as the S and C commands. The #K command kills the entire buffer.

Inserting Text

You insert text into the buffer from the terminal with I (Insert) commands. Like search commands, the I command requires either a CTRL-D or a delimiter at the end of the text string. If you omit the delimiter, SPEED treats the keying you intend for a command line as a continuation of the character string of the insertion.

You are missing a word in the third line. Have SPEED conduct a search for *put* and insert *in*.

```
!Sput$lin$T$$
```

The result will be

```
Please putin* the missing word.
```

which contains an unfortunate error. Move two characters backward and insert the missing space by entering

```
!-2M$!□$T$$
```

```
Please put *in the missing word.
```

You may want to follow these hints to make SPEED insertions more easily:

- Always check the screen when making an insertion to ensure that you typed in the I command name first.
- Decide on a maximum line length and type a NEW LINE as you approach that length. (Screen wrap-around may otherwise mislead you to think that you have distinct lines when you don't.) Or type in a paragraph as a single line and go to the front of the line. Looking at the screen, you will see how many lines to create. For example,

```
!OL8<70MC□$!  
$>$$
```

tells SPEED to create 8 lines that are at least 70 characters long, starting at the beginning of the paragraph, by replacing the first space after each 70th character with a NEW LINE.

- Use DEL or RUBOUT within an insertion just as you would in the rest of a command line.
- Make several shorter insertions rather than one longer one in order to limit the consequences of an incorrect insertion.

Checking the Status of the Buffer

After you have spent some time in an editing session, carrying out various tasks, you may wish to verify in several ways where you are and what you have done.

Typing Out Text

You have already used a couple of variants of the T (Type) command. nT displays lines following the CP, and -nT displays lines preceding the CP. 1T displays the current line to the right of the CP, OT displays the current line to the left of the CP. The default value for T is the entire current line.

m,nT displays text from the $m+1^{st}$ to the n^{th} character in the buffer.

#T displays the entire buffer. (If the buffer is longer than the maximum that the video display allows, the display will roll.) Enter

```
!#T$$
```

You should see

```
Here is a second sentence to type in.  
Here is a setnece fo you tocrrect.  
Please put in the missing word.  
Womethg iswrong on this lin.  
Here is a sentence to work with.
```

In subsequent SPEED editing sessions, you may find it convenient to set an automatic 20-line display, in order to avoid entering a stream of T commands. You do this by using the /D Display switch.

```
) X SPEED /D [filename] )
```

Finding the Status of Lines and Characters

You may check on the status of your SPEED editing session in other ways by using the = Equals command and some SPEED housekeeping variables.

To find out which line your CP is on, interrogate the Value Line variable

```
!VL=$$
```

To find out the number of lines you have in the buffer, interrogate the Value Number variable

```
!VN=$$
```

To find out how many characters the CP has moved from the beginning of the line, interrogate the Value Moved variable

```
!VM=$$
```

To find out how many characters precede the CP, interrogate the Current CP Position variable

```
!.=$$
```

And to find out how many characters there are in the buffer, interrogate the Z Last Character variable

```
!Z=$$
```


For example, if you enter

```
!VN=Z=$$
```

you should see

```
5  
168
```

which tells you that you have 5 lines and 168 characters in the buffer.

Checking the Status of Files

Let's suppose at this point that you are ready to end this SPEED editing session even though you haven't finished filing everything. Do you remember which files you have open?

If you don't recall, enter a F? (Files?) command:

```
F?$$
```

You should see

Global:

Input File - :UDD:YOURDIR:MYFILE

Output File - NEWFILE

Local:

Input File - None

Output File - None

which tells you that your input file is MYFILE and gives you its path. It also tells you that your output file is NEWFILE. You created it a little earlier in this session, remember? (We will discuss the distinction between global and local files in Chapter 3.)

Ending a SPEED Session

To save the editing you have done for use after this session, you must place it in an output file.

Writing to a File

You can put all or some of the contents of the buffer into the output file with one of the P, :P, PW, :PW commands. To put the entire buffer into the output file, enter the P (Put) command without a numerical argument as

```
!P$$
```

These commands without the symbolic colon modifier (:) put text in the output file but do not clear the buffer. The commands with the colon modifier clear from the buffer the material they put in the output file. The P commands append a form feed to the end of the output material, creating pages out of the successive buffer contents that you put into the file. The PW commands do not append a form feed, so that you may continue building a single page with successive buffer contents.

All the P commands, like the C, K, S, and T commands, take positive, negative, and paired numerical arguments: nP copies lines forward from the CP, -nP copies lines preceding the CP, and m,nP copies text from the m + 1st through to the nth character to the output file.

Closing Files

Although you have now written to the output file, you have not closed it: it is still open for further copying of text from the buffer. (Your input file is still open, too. If it had another page, we could read it into the buffer and do some more editing.)

To close both files, issue an FC (File Close) command

```
!FC$$
```

If you were to issue another F? (Files?) command, you would now see that all entries are *None*.

Exiting

You may now ask SPEED to end the SPEED editing session with an H Halt command:

```
!H$$
```

SPEED asks you this time

Confirm?

because, although you have closed your files, you still have text in the buffer. (You did not use the colon with the P command, remember?) Since you are willing to discard the buffer contents, go ahead and type in a y). Shortly you will receive the prompt of your parent process, usually the CLI.

SPEED will query you about exiting only if you have text in the buffer or open files. If you decide not to exit, type any other character but y) and proceed with your editing. If you decide to exit, SPEED will kill the buffer and close any open files, *but it will not copy the contents of the buffer to an output file*.

You have now worked through your first SPEED editing session.

Exercise

Try this exercise before reading through one possible answer on the next page.

Exercise 2-1.

Enter SPEED. Open NEWFILE and edit the rest of its contents. Put the edited text in an output file, RIGHTFILE. Exit SPEED.

Answer to Exercise 2.

) X SPEED / D)
!FRnewfile\$A\$\$

**Here is a second sentence to type in.
Here is a setnece fo you tocrrect.
Please put in the missing word.
Womethg iswrong on this lin.
Here is a sentence to work with.*

SD-02378

!Ctne\$nten\$\$

*Here is a second sentence to type in.
Here is a senten*ce fo you tocrrect.
Please put in the missing word.
Womethg iswrong on this lin.
Here is a sentence to work with.*

SD-02379

!Sfo\$lr\$Sto\$!□\$\$

*Here is a second sentence to type in.
Here is a sentence for you to *crrrect.
Please put in the missing word.
Womethg iswrong on this lin.
Here is a sentence to work with.*

SD-02380

!Cr\$0\$2LCw\$S\$\$\$

*Here is a second sentence to type in.
Here is a sentence for you to correct
Please put in the missing word.
S*omethg iswrong on this lin.
Here is a sentence to work with.*

SD-02381

!Sth\$lin\$4MI□\$\$

*Here is a second sentence to type in.
Here is a sentence for you to correct.
Please put in the missing word.
Something is *wrong on this lin.
Here is a sentence to work with.*

SD-02382

!C.\$e.\$\$

*Here is a second sentence to type in.
Here is a sentence for you to correct.
Please put in the missing word.
Some thing is wrong on this line.*
Here is a sentence to work with.*

SD-02383

!FWrightfile\$:PWFCH\$\$
) –

End of Chapter

Chapter 3

Using Files and Buffers in SPEED

You can manipulate multiple files and buffers in SPEED in order to carry out complex editing tasks. SPEED can create up to 36 buffers for you to use as workspaces. During an editing session, you can move from one buffer to another, and you can copy material from one buffer to another. You can open different input and output files for each buffer. If you want access to a file *from every buffer*, you open it *globally* with one of the **Fx** commands. If you need access to a file only *from a single buffer*, you open it *locally* with one of the **BFx** commands. You may have only one pair of input and output files open globally at a time, and each buffer may have only one pair of input and output files open locally.

Using Global Files

When you use global files, you have access to them from every buffer. You could, for example, read successive pages of a global input file to different buffers, and then copy the buffers in a different order to a global output file, so that the two files would differ only in the order of their pages.

Opening Global Files

You have already learned two ways of opening global files. The **FR** (File Read) command opens an already existing file for input. You must enter a filename argument with the **FR** command, and either the **ESC** delimiter or a **CTRL-D**. The editing session will not change the contents of a file opened with the **FR** command.

You have also already learned how to create a global output file with the **FW** (File Write) command. You must enter a filename argument with this command, and the filename must be new to the current directory. You create contents for this file by copying to it from the buffer.

If you finish with the input file **ALPHA** and want to open a new input file **BETA**, you may close the current input file **ALPHA** and open a new file with a **FNRbeta\$** (File New Read) command. (If you do not specify a filename argument, the command merely closes the current input file.)

You can work with a succession of output files in the same way. If you have copied everything you want to the

file **OMICRON**, you can create the new output file **OMEGA** and close the current file with a **FNWomega\$** (File New Write) command. (If you do not specify a filename argument, the command merely closes the current output file.)

SPEED has a strategy for updating already existing files. Instead of entering an **FR** command, open a file for updating with an **FOfilename\$** (File Open) command. When you do this, SPEED creates an output file for you, with the same name as the input file and a **.TM** extension. When you later update your file with an **FU** command, SPEED writes your edited copy to **FILENAME.TM**, deletes the file **FILENAME**, and renames **FILENAME.TM** to **FILENAME**. As a result, your edited material has the old filename. When a file is open for updating, no one else can have access to it, nor can you gain a second access to it. SPEED automatically clears the buffer and yanks the first page of a file opened for updating into the buffer.

You may also open a file by entering SPEED with a filename argument. If the file exists, SPEED opens it for updating. If the file does not exist, SPEED asks you if you want to create it. If you type in **y**, SPEED creates an output file with that name.

Reading from Files

If your file is not paged (you have selected a nonzero window mode) SPEED reads only the number of lines you specify; otherwise (page mode) it reads up to the first form feed. When you open an input file for updating, SPEED reads the first page or window into the buffer.

If you open an input file with an **FR** or **FNR** command, you must bring text into the buffer with an explicit command. You have already learned one of these: the **A** (Append) command reads the first page or window of the file into the buffer. If there is already material in the buffer, the material you append follows it. If your **CP** is at the bottom of the buffer before you append, the **CP** precedes the appended material.

If you are ready to *discard* the current contents of the buffer, and wish to bring the next page or window of the input file into the buffer for editing, issue a **Y** (Yank) command. With this command, you lose the current buffer permanently. If Update Mode is on (if you have a

file open for updating), there are characters in the buffer, and the Y command is not in a Command Loop,

SPEED will query you

Confirm (Y-command) ?

before it yanks a page or window. If Update Mode is *not* on, or there are no characters in the buffer, or the Y command is in a Command Loop, SPEED carries out the yank without querying first.

A good rule of thumb is never to issue a Y command until you have considered the alternatives to it.

If you are done editing the current buffer, and wish to write to an output file, clear the buffer, and read in another page of the input file, you may do so in a single command. Enter an R (Read) command. This command does the work of a PY sequence.

The R command takes a positive numerical argument. An nR command reads pages or windows out and in n times.

Writing to Files

You already know of two ways to write to output files. You learned of the P (Put) and PW (Put Without Formfeed) commands during the editing session of Chapter 2. These commands copy the buffer to an output file. They have no input functions. They clear the buffer if preceded by a colon (:).

In the “Reading from Files” section of this chapter we sketched the R (Read) command. That command writes the contents of the current buffer to an output file and yanks the next window or page of the input file into the buffer.

If you are content with the current buffer text, wish to make no further changes in the rest of the windows or pages of the input file, but wish to copy the rest of the input file to the output file, issue an E (End) command. The E command also clears the buffer.

Closing Global Files

You have already learned of some ways to close files. In Chapter 2, you used the FC (File Close) command. This command does nothing more than close any open global file: it does not write to a file, and it does not clear the buffer.

In the “Opening Global Files” section of this chapter, you learned that the FNR (File New Read) and FNW (File New Write) commands close current input and output files as they open new ones. Those commands also do not clear the buffer or write to files.

If you have opened a file for updating, you may close it with an FU (File Update) command. This command copies the buffer and the rest of the input file to the output file. It also clears the buffer. If the input file is FILENAME and the output file is FILENAME.TM (which SPEED created when you issued an FO (File Open) command or entered SPEED with a filename argument), the command deletes FILENAME and renames FILENAME.TM to FILENAME. The result is a closed, updated file.

If you wish to retain both the updated and the original versions of your file for future reference, issue a FB (File Backup) command. This command also copies the buffer and the rest of the input file to the output file before it closes the files. It also clears the buffer. If the output file is FILENAME.TM, as above, the FB command renames FILENAME to FILENAME.BU, and then renames FILENAME.TM to FILENAME.

Table 3-1 summarizes file opening, reading, writing, and closing possibilities.

Using Multiple Buffers

In Chapter 2, you learned how to edit text in a single buffer. You can use other buffers to simplify otherwise tedious editing tasks. Please open your file RIGHTFILE for updating by entering SPEED and issuing an FOrightfile\$ command. When you do, the buffer should look something like:

**Here is a second sentence to type in.
Here is a sentence for you to correct.
Please put in the missing word.
Something is wrong on this line.
Here is a sentence to work with.*

Let's try to make the sentence containing the phrase *second sentence* the second sentence -- without retyping it.

Copying to Another Buffer

There are two ways to copy material to another buffer. One way leaves the original material in the current buffer, and the other deletes the original material.

To retain a copy of the entire current buffer, which is Buffer 0, issue a BC (Buffer Copy) command for any other buffer name, for instance,

```
!BCA$$
```

The only legal buffer names are the digits 0 through 9 and single letters of the alphabet.

Table 3-1. Input and Output Functions

	Appends buffer to output file	Clears buffer	Appends one window of input file to buffer	Copies remainder of input file to output file	Clears buffer
FO	-	+	+	-	-
A	-	-	+	-	-
Y	-	+	+	-	-
R	+	+	+	-	-
E	+	+	-	+	+
P, PW	+	-	-	-	-
:P, :PW	+	+ ¹	-	-	-
FU	+	+	-	+	-
FB	+	+	-	+	-

¹ May not clear entire buffer if command has numerical arguments

Legend: + yes - no

Like the S and C commands, the BC command takes all three kinds of numerical arguments. An nBCx command copies the n lines following the CP to Buffer x, a -nBCx command copies the n lines preceding the CP, and an m,nBCx command copies from the m+1st through the nth character. The default value, which you have just used, is the entire buffer. (The value is independent of the CP position.)

The BT (Buffer Take) command behaves like the BC command except that it deletes from the current buffer the material it copies to the destination buffer. Remember to give the command a numerical argument, or you will delete the whole buffer! To move the first sentence down a line, we first transfer it to another buffer, say Buffer 2.

!1BT2\$\$

Your buffer should now look like this.

**Here is a sentence for you to correct.
Please put in the missing word.
Something is wrong on this line.
Here is a sentence to work with.*

Place your CP at the point where you want to reinsert the sentence with a 1L command. To insert one buffer in another, you must issue an I command with a *buffer expansion*. (We will discuss expansions in general in Chapter 4.) The expansion consists of CTRL-B and the

buffer name (you depress CTRL and key in B and the buffer name). For example, enter

!!|B2\$\$

Your buffer should now look like this.

*Here is a sentence for you to correct.
Here is a second sentence to type in.
*Please put in the missing word.
Something is wrong on this line.
Here is a sentence to work with.*

And you've done the trick. A word of caution: both the BC and BT commands delete the current contents of the buffer to which you copy. If a buffer contains text you want to save, do not issue a BC or BT command to that buffer.

Switching to Another Buffer

When you enter SPEED, your current buffer is Buffer 0. You don't have to stay there. You can switch to any of the other 35 buffers to continue editing. For example, let's switch to Buffer 2 by entering a BS2\$\$ (Buffer Set) command:

!BS2\$\$

Your current buffer now holds

**Here is a second sentence to type in.*

since that is what you transferred to this buffer a moment ago. You can edit in this buffer if you want to, by entering

!Csecond\$third\$\$

To return to your original buffer, enter

!BSO\$\$

and reinsert the fruits of your labors by entering

!!|B2\$\$

with this end result.

Here is a sentence for you to correct.

Here is a second sentence to type in.

Here is a third sentence to type in.

**Please put in the missing word.*

Something is wrong on this line.

Here is a sentence to work with.

These examples will, no doubt, suggest experiments of your own.

Killing a Buffer

You may *kill*, or delete a buffer with a BK (Buffer Kill) command. It requires a buffer name argument. Let's say we're done with Buffer 2. Go ahead and enter

!BK2\$\$

You cannot kill the current buffer with a BK command.

Using Local Files

At the moment, you have a *global* file, RIGHTFILE, open for updating. You have access to RIGHTFILE from any buffer. For example, you could put the current buffer in RIGHTFILE with a P command, switch to another active buffer, and put its contents in RIGHTFILE. But suppose, instead, that you want to put the contents of Buffer A in a different file without closing RIGHTFILE (since you want to continue editing it). Can you do that?

Local File Commands

Yes, you can open another output file without closing RIGHTFILE. Switch to Buffer A, where we placed a copy of the contents of RIGHTFILE for safekeeping. You may create a *local* output file for Buffer A and write the buffer to it. For example,

!BFWcopyfile\$P\$\$

In doing this, you do not disturb the contents of other buffers, or the contents of any open global files.

If you check your file status now, while you are in Buffer A, you see

Global:

Input File - :UDD:YOURDIR:RIGHTFILE

Output File - :UDD:YOURDIR:RIGHTFILE.TM

Update Mode On

Local:

Input File - None

Output file - COPYFILE

In order to close your local file, you will issue a BFC (Buffer File Close) command.

In the sections "Opening Global Files" and "Closing Global Files" we sketched file opening and closing maneuvers. To each global file opening and closing command, there corresponds a local file opening or closing command, which you can construct simply by prefixing a B to the Fx command. See Table 3-2.

A word of caution: do not attempt to execute both FO and BFO commands in the same buffer without an intervening file closing command; you may lose material from the file you open first.

Manipulating Local and Global Files

How might you split a file into two files, using SPEED buffers and local file commands? Something like this would do the trick, assuming the file CHICKEN contains only one page.

- Open CHICKEN for updating in, say, Buffer 0 with an FO command
- Locate the CP where you want the split to occur with L, J, and M commands
- Copy the material following the CP to, say, Buffer A, and delete the original, with a ZBTA command
- Switch to Buffer A
- Create an output file EGG with the local file command BFWegg\$
- Write the contents of Buffer A to EGG with a P command

After any editing of EGG, you would close it with a BFC command. You could then return to Buffer 0 and continue editing CHICKEN. If you do no further editing on CHICKEN or EGG, the result of the steps above is two files: a truncated CHICKEN file, and an EGG file that contains the material you deleted from CHICKEN.

Table 3-2. File Handling Commands

<i>Global Opening Commands</i>	<i>Local Opening Commands</i>	<i>Global Closing Commands</i>	<i>Local Closing Commands</i>
FNR	BFNR	FB	BFB
FNW	BFNW	FC	BFC
FO	BFO	FU	BFU
FR	BFR		
FW	BFW		

How might you combine two files into a single one? Assuming, again, that your files consist of single pages, you might proceed this way.

- Open DOUBLE for updating with an FO command in Buffer 0
- Switch to Buffer A and open ORNONE for updating with a local BFO command
- Switch back to Buffer 0
- Locate the CP where you want to insert the contents of ORNONE
- Enter an I|BA\$\$ command

You can then continue editing in either buffer or close the files. If you do no other editing, your file DOUBLE now contains its previous contents and the contents of ORNONE.

Command Precedence

When you open files both locally and globally in the same buffer, subsequent commands will apply *to the local file first*. Only after you close the local file will commands apply to the global file. For example, if you enter the command line

```
!FRwait$BFRgrab$A$$
```

SPEED opens WAIT globally and GRAB locally. The A command applies to GRAB, and SPEED reads into the buffer the first page of GRAB, not WAIT.

Checking the Status of Buffers

It is easy to lose track of which buffers you have activated during an editing session. You may need to check on them, especially toward the end of the session,

if you plan to write to various output files from different buffers. The B? (Buffers?) command allows you to do this.

During this session, we activated three buffers: Buffer 0 when we entered SPEED, Buffer A when we copied Buffer 0 to it, and Buffer 2 when we decided to rearrange the lines of Buffer 0. Later, we killed Buffer 2, so if you now enter

```
!B?$$
```

you should see

```
= >Buffer 0 - 212
    Buffer A - 172
```

The arrow points to your current buffer. The figure following the buffer name gives you the number of characters in that buffer.

If you are interested only in the status of a specific buffer, Buffer Q, you may enter the command with a buffer name argument

```
!B?Q$$
```

Exercises

Exercise 3-1.

Duplicate a file within itself.

Exercise 3-2.

Type in five lines. Reverse their order.

Exercise 3-3.

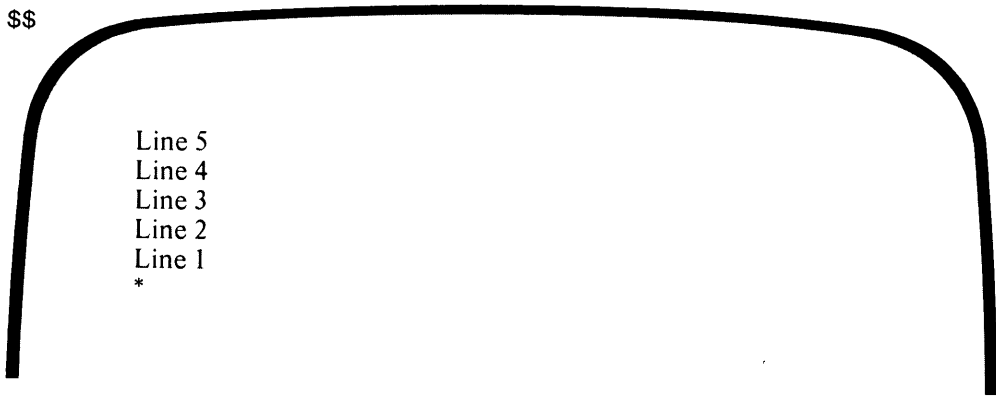
Open a three-page input file. Create a new file for each page. Create an output file with the pages in reverse order.

Answer 3-1.

!FOdoublefun\$\$
!BCR\$\$
!!|BR\$\$
!FU\$\$

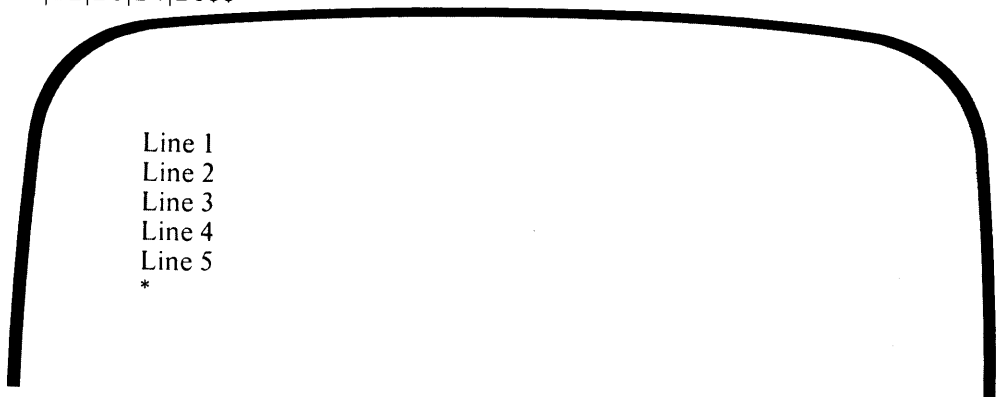
ANSWER 3-2.

!!Line 5)
Line 4)
Line 3)
Line 2)
Line 1)
\$\$



SD-02384

!J\$1BT5\$1BT4\$1BT3\$1BT2\$\$
!1L|B2|B3|B4|B5\$\$



SD-02385

Answer 3-3.

!FRinfile\$\$
!Y\$BCA\$\$
!Y\$BCB\$\$
!Y\$BCC\$\$
!FC\$FWoutfile\$\$
!BSC\$BFWfile.c\$P\$\$
!BFC\$P\$\$
!BSB\$BFWfile.b\$P\$\$
!BFC\$P\$\$
!BSA\$BFWfile.a\$P\$\$
!BFC\$P\$\$
!FC\$\$

End of Chapter

Chapter 4

Intermediate Editing with SPEED

This chapter explores the SPEED environment, showing the use of numerical expressions and symbolic modifiers. It acquaints you with the various variable and mode commands that make SPEED housekeeping easier. It shows you how to use the control keys as expansions and templates. And it introduces you to commands that work across windows or pages.

Arithmetic in SPEED

You can carry out simple arithmetic tasks in SPEED, and you can use the evaluations of numerical expressions as arguments to other commands.

add m to n	$m+n$
subtract n from m	$m-n$
multiply m by n	$m*n$
divide m by n	m/n
negate m	$-m$

To display the value that SPEED returns for a numerical expression, follow it with the = (Equals) command.

SPEED evaluates numerical expressions from left to right. That is, SPEED first adds, then multiplies, then divides, then subtracts in the following example to get the result

```
!9+3*4/2-3=$$
```

```
21
```

Parentheses are illegal in SPEED numerical expressions. SPEED performs only integral arithmetic, truncating after division. SPEED commands that take both positive and negative numerical arguments are restricted to the range -32768 through +32767. Commands which take double or positive arguments range from 0 through +65535.

Numerical Expressions as Text

You can ask SPEED to perform simple computations for you, and then insert the result in the text. For example, if you want to include the correct answer in the text as part of the conjecture "I wonder whether $144*144=$ ___ or not?" you need not make the calculation. You can use the \ (Backslash) command instead to insert the result as follows.

```
!!!\wonder\whether\144*144=$$  
!144*144\$$  
!!\or\not?$$
```

```
I wonder whether 144*144=20736 or not?*
```

Storing Numerical Expressions

SPEED uses 10 variables, V0 through V9, for storing the evaluations of numerical expressions. You can display or insert the values of the variables by using them as arguments to = and \ commands.

You may set a variable to a numerical result with a VS (Value Set) command, you can increment a variable by one with a VI command, and you can decrement a variable by one with a VD command. You can set and insert, or set and display, a variable in one command. For example,

```
!13*69VS0\$$
```

```
897*
```

```
!V0/3VS0=$$
```

```
299
```

```
!V10=$$
```

```
300
```

You may also store the values of the SPEED pseudo-variables in variables. For example, if there are 117 lines in the buffer, then the command VNVS0 stores the number 117 in Variable 0.

You can perform arithmetic operations on both variables and pseudo-variables. For example, if you want to know your average line length, you can ask SPEED to divide Z, the number of characters in the buffer, by VN, the number of lines in the buffer, with a `Z/VN=$$` command.

Using Decimal Equivalents of Characters

Some characters, such as NEW LINE or form feed, are awkward to type in on occasion, for instance, when you are composing a command line for later use. SPEED allows you to insert characters by typing in the ASCII decimal value of the character as a numerical argument to the I command. Since 10 is the ASCII decimal for NEW LINE, you can type

```
!S.$10!T$$
```

and get the same result as you would with

```
!S.$!)  
$T$$
```

(Of course, if you want to save key strokes, you will use the second option. Use the nl command when there is no other way to insert a character, such as]D.)

In the example above, SPEED evaluates the numerical expression before making the insertion, so `15/3*2!$` also inserts a NEW LINE.

Appendix A gives the ASCII character set.

Boolean Functions in SPEED

SPEED can perform Boolean operations on binary numbers. SPEED represents them as in Table 4-1.

where *x*, *y* range from 0 to 65536, or from -32768 to 32767. (If you are not a programmer or do not know what a Boolean function is, you may disregard this section entirely.) Please consult each Boolean operator under its entry in Chapter 7 for details.

Using Variables in Housekeeping

Suppose that you are working on a certain line of text and recall an error that you made elsewhere that you want to correct before you forget it. You don't know or care what line you are on, but you want to return to it when the interruption is over. You store the line number, VL, in a free variable (one you don't need for anything else), say 6, by entering a `VLVS6$$` command. You go hunting for the location of the error, make the correction, and are ready to return to your previous editing site. To do so, you merely enter a `V6J$$` command. SPEED substitutes the value you put in V6 and carries out the J command with that line number. Now you're back on the main job!

The example shows that you can use the values of pseudo-variables to simplify your editing tasks. You can display or insert them with = and \ commands, and you can use their values as numerical arguments to other commands as well.

These SPEED pseudo-variables are available as numerical arguments.

- VC Value Character gives the ASCII¹⁰ value of the character to the right of the CP or zero when CP is at the end of the buffer
- VL Value Line gives the current line number
- VM Value Moved gives the number of single moves between the beginning of the line and the CP
- VN Value Number gives the total number of lines in the buffer
- VP Value Previous gives the previous CP position at start of the last search in number of characters from beginning of the buffer
- Z Z Last Character gives the total number of characters in the buffer
- Current CP Position gives the number of characters between the beginning of the buffer and the CP

Table 4-1. Boolean Functions

Logical Function	or Boolean Operation	Representation
AND	Intersection	$x \wedge y$
inclusive OR	Union	$x \vee y$
exclusive OR	Symmetric Difference	$x \oplus y$
NOT	Negation	$\neg x$

Modifying Commands

SPEED uses three nonnumeric *symbolic modifiers* to adjust the functions of SPEED commands: the colon (:) modifier, the commercial at (@) modifier, and the ampersand (&) modifier. Each *precedes* the command name. When you use more than one, they can occur in any order before the command name.

Conditionalizing the Next Command

Suppose you want to issue a D (Delete) command only if an S (Search) command is successful. You can do this with the colon modifier preceding the search (C, N, Q, S) commands. Using the colon modifier, SPEED returns a +1 if the command succeeds, and a 0 if the command fails. You may use this value as part of a numerical expression serving as a numerical argument to the next command.

For example, suppose you want to get rid of some, but not all, instances of *ue* (the one in *catalogue*, but not the one in *value* or *Tuesday*). You might type in

```
!:Slog$*2D$$
```

SPEED searches for *log*. When it finds the string, it returns a value of 1 to carry out the multiplication on. That gives 2 as the numerical argument for the D command, which then deletes two characters. If SPEED fails to find *log*, it returns a zero. SPEED does not display an error message, but since $0*2=0$, SPEED deletes nothing.

Inclusion of a search command in an iteration automatically simulates the : colon modifier.

The colon modifier has several other uses, too. (You learned one with the P command earlier; using the colon there clears the buffer.) Please consult the entry for the colon symbol in Chapter 7.

Setting a Temporary Delimiter

Suppose you are reading a SPEED command file of your friend Nick's. (You will learn about command files in Chapter 6.) You see the command line

```
C!$0ZJ$$
```

and are certain from the context that Nick meant to say "Change 1 to 0 and jump to the end of the buffer" rather than "Change 1 to 0ZJ." Nick is missing a delimiter. You would like to help Nick out, but if you try to insert a delimiter with something like

```
!COZJ$0$ZJ$$
```

you will merely delete the *ZJ* in Nick's text and jump yourself to the bottom of the buffer. SPEED cannot treat the delimiter as text without special provisions. Instead, you must type

```
!S0$@!/$/$$
```

When you type in the commercial at (@) modifier before the I (Insert) command, you tell SPEED to accept the character following the command name as a temporary delimiter. This time you choose the slash sign (/) for a temporary delimiter. SPEED searches for 0. Next, SPEED inserts an ESC (\$) since you typed it in between the temporary delimiters you selected. SPEED restores the standard delimiter to its function as soon as it executes the command.

You may set a temporary delimiter for any of the insertion and search commands. Please consult the entry for this symbol in Chapter 7 for more details.

Using the Alternate Radix

SPEED allows you to do not only decimal arithmetic, but arithmetic in an alternate radix from 2 to 36. (Your default alternate radix is octal.) You use the alternate radix by prefixing the appropriate number or command with the ampersand (&) symbol.

An ampersand before the command name, for instance = or \, interprets the result of the command in the alternate radix. An ampersand before the numerical arguments to the command interprets those arguments in the alternate radix. For instance, for the octal radix,

```
10*10&=144
&10*&10=64
&10*&10&=100
```

Digits following the ampersand must be valid digits in the alternate radix: 8 and 9, for example, are not valid digits in the octal radix. Please consult the entry for this symbol in Chapter 7.

Choosing the SPEED Modes

When you enter SPEED, default values for some of SPEED's parameters are set. You can alter them for ease and efficiency. (You have already learned to adjust one, display mode, by entering SPEED with the D switch.) For each mode except WR, the default value is 0. By changing that value, you change a SPEED characteristic.

Default Argument Mode

The setting of the default argument mode determines how SPEED treats certain commands when you issue them without explicit numerical arguments.

If you issue no WA Window Advance command, or reset the mode with a OWA command, SPEED treats certain commands without numerical arguments as if they had an argument of 0. A D or M command without an argument has no effect, a J command sends you to the beginning of the buffer, and an L command takes you to the beginning of the current line.

If you issue a 1WA Window Advance command, SPEED treats certain commands without numerical arguments as if they had an argument of 1. A D command deletes one character to the right, and an M command moves you one character to the right. A J command still sends you to the beginning of the buffer (because lines are numbered from 1), and an L command takes you forward to the beginning of the next line.

You should be aware, however, that many variable expressions and operations automatically provide a numeric argument for the following command. When in doubt, it is safer to add 0 before the command, particularly in command files that run blind (see the /1= Invocation switch). Suppose, for example, that you set WA to 0, and later issue a Command Loop of the form

```
<...:Stext$L...>
```

in the expectation that a successful search will return you to the beginning of the line. Instead, however, the successful search furnishes the next command with an argument of 1 and takes you down a line.

Case Control Mode

The case control mode allows you to enter both lowercase and uppercase text from a terminal that has only uppercase characters.

A OWC Window Case command turns case control off. The terminal reads characters exactly as you type them, with no translation from uppercase to lowercase.

A 1WCx\$ Window Case command turns case control on and allows you to designate x as the shift-up character. SPEED treats the next immediate alphabetic character following an x as uppercase, and all others as lowercase.

A -1WCx\$ Window Case command turns case control on and allows you to designate x as the shift-down character. SPEED treats the next immediate alphabetic character following an x as lowercase, and all others as uppercase.

In addition, the 1WCxy or -1WCxy Window Case command lets you designate a second character y as the shift-lock toggle. SPEED treats all alphabetic characters following the shift-lock as the same case (upper for 1 and lower for -1) until it encounters another shift-lock or a CTRL-D.

To *insert* the shift or shift-lock character as text, rather than use it, precede it with the shift character.

To see the effect of this command at an uppercase terminal, turn the ALPHA LOCK toggle on. Set Window Case Mode to 1, selecting / as the shift character and \ as the shift-lock toggle

```
!1WC/\$$
```

Now make an insertion, perhaps

```
!!/SET /CASE /CONTROL TO 1 AND  
DISPLAY \THIS.$OT$$
```

SPEED then displays

```
/SET /CASE /CONTROL TO 1 AND  
DISPLAY /T/H/I/S.
```

Then, when printed on an upper- and lowercase printer, the line is

```
Set Case Control to 1 and  
display THIS.
```

Display Mode

The display mode allows you to specify how much automatic screen display you want.

The OWD (Window Display) command turns display mode off, giving you no automatic display of text lines. For displays, you must enter appropriate T commands individually.

The nWD (Window Display) command, where n ranges from 1 to 10, specifies the n lines of text on either side of the CP you want SPEED to display after executing each command line, and before restoring your prompt.

The mode value is 10 if you enter SPEED with the /D switch. The default value is 0. This mode treats a value greater than 10 or less than 0 as 10.

Window Mode

The window mode allows you to specify how much text you wish SPEED to read into the buffer at one time.

The OWM (Window Mode) command sets SPEED to read in the text a page at a time, from form feed to form feed. If your file contains no page breaks (form feeds), SPEED will attempt to bring in the entire file. With the OWM value, SPEED does not display the original form feeds (⌊L), and you cannot delete them except by using the A (or Y) command.

An nWM (Window Mode) command sets SPEED to read in n lines at a time, where n is positive. With these values, SPEED enters and displays form feeds as ⌊L, and you can search for and delete them.

Position Mode

The position mode allows you to determine the way SPEED repositions the CP after an unsuccessful C or S search command.

The OWP (Window Position) command specifies the beginning of the buffer for default search commands. It specifies the previous position of the CP followed by n lines for positive nC and nS commands, the position before the search for negative -nC and -nS commands, and the position after the nth character for m,nC and m,nS commands. That is, it repositions the CP where the search ends.

The 1WP (Window Position) command specifies the position of the CP before the search for default commands, the position before the search for positive nC and nS commands, n lines before the previous position for negative -nC and -nS commands, and the position following the mth character for m,nC and m,nS commands. That is, it repositions the CP where the search began.

Alternate Radix

The alternate radix command allows you to specify an alternate radix from 2 to 36 for SPEED arithmetic. The default alternate radix is 8 (octal). The standard radix is always decimal.

Enter an nWR (Window Radix) command, where n ranges from 2 to 36 and is the value of the radix you wish to use.

To use the alternate radix, you must key in appropriate instances of the ampersand (&) character, discussed in the section "Modifying SPEED Commands." For example, if you set the alternate radix to hexadecimal and issue some = (Equals) commands

```
!16WR100& = &100 = $$
```

SPEED tells you

```
64  
256
```

Shift Sensitive Mode

This command allows you to specify whether SPEED is to ignore or take into account the case of alphabetic characters when conducting searches.

The OWS (Window Shifts) command sets case-independent searches: U matches *U* and *u*, and c matches *C* and *c*.

The IWS (Window Shifts) command sets case-dependent searches: U matches only *U* and not *u*, and c matches only *c* and not *C*.

SPEED Control Characters and Templates

SPEED provides you with certain control characters that make it easier for you to

- insert the contents of other buffers or files
- execute command lines contained in other buffers or files
- conduct modified or generalized searches

Where A is an arbitrary control key, we mention it with CTRL-A, and designate it by]A in a command line. When you wish to key in CTRL-A, depress the CTRL key and, while holding it down, type A.

Expansions to Buffers and Files

CTRL-Bbuffer-name In a command string, where *x* is a legal name of an active but noncurrent buffer, SPEED expands the current command string at]B*x* to include the data stored in Buffer *x*, if there is any. If Buffer C contains

Cour\$or

and your location in the text is

**a colour programme*

the command

!]BC\$\$

will respell *colour* to *color*.

CTRL-Bbuffer-name
(continued)

In a search or text string, where *x* is a legal name of an active but noncurrent buffer,]B*x* represents the contents of Buffer *x*. SPEED tries to match

pass]B*x*word

with *pass*, the contents of Buffer *x*, and *word*. For example, if Buffer 3 contains

alongthis

and you tell SPEED

!]pass]B3word\$\$

SPEED inserts

passalongthisword

in the current buffer.

To *insert*]B in a command line, instead of referring to a buffer's contents, use]B]B,]F]B, or 2]\$.

CTRL-Ffilename\$

In a command string, where *x* is the name of a file (to which you have access), SPEED expands the current command string at]F*x*\$ to include the data in file *x*, if there is any. If file BRIT contains just

Camme\$am

and your location in the text is

a color programme*

the command string

!]Fbrit\$\$

will correct *programme* to *program*.

CTRL-Ffilename\$
(continued)

In a search command, where *x* is the name of a file (to which you have access), SPEED tries to match `pass]Fx$word` with *pass*, the entire contents of the file, and *word*. You must follow the filename with the ESC delimiter or CTRL-D; if you do not, you will get the wrong file or no file at all.

In an insertion command, if you wish SPEED to treat what follows the filename as text, type in a single ESC. If instead you wish SPEED to treat what follows the filename as a command, you must type in *two* delimiters. If your file WORD contains exactly

me the next

and you tell SPEED

`!]pass]Fword$]sword$$`

SPEED inserts

pass me the next sword

but if you tell SPEED instead

`!]pass]Fword$$]sword$$`

(where the first \$\$ echoes two delimiters and the second a CTRL-D) SPEED inserts only

pass me the next

and undertakes a search for the next occurrence of *word*.

To *insert*]F in a command line, instead of referring to a file's contents, use]F]F,]B]F, or a 6!\$ command.

Search Templates and Control Characters

CTRL-E

In a search string,]E represents one or more tabs or spaces. SPEED tries to match `pass]Eword` with *pass*, one or more tabs and spaces, and *word*. So it would find

pass □ □ □ *word*

To match *zero* or more tabs and spaces, use]T instead.

CTRL-G

SPEED tries to match the search string. If it finds the string, it positions the CP at the position in the string at which you inserted]G. SPEED tries to match `pass]Gword` with *password* and position the CP as in *pass*word*. On models other than 6052, 6053, D100, or D200, you may use]] instead of]G. Insert it with a 7!\$ command.

CTRL-N

In a search string, SPEED accepts as a match in the position following]N any character *except* the character (or one character from the following]\...]\ list) following]N. SPEED tries to match `pass]Nsword` with *pass*, any next character except *s*, and *word*. For example, it would match

pass □ *word*

A sequence of CTRL-Ns has the same effect as a single one. (SPEED bypasses only one character.)

CTRL-T

To enter this control character, type in the sequence CTRL-P, CTRL-T. In a search string,]T represents zero or more tabs or spaces. SPEED tries to match `pass]Tword` with *pass*, zero or more tabs and spaces, and *word*. To match *one* or more tabs and spaces, use]E instead.

CTRL-W

In a search string, SPEED interprets the character following `]W` literally. SPEED tries to match `pass]W]word` with *pass*, an occurrence of `]E`, and *word*, rather than with *pass*, one or more tabs and spaces, and *word*. This template is, of course, especially useful when editing SPEED command lines. On models other than 6052, 6053, D100, or D200, you may use `]_` instead of `]W`. Insert it with a `23I$` command.

CTRL-X

In a search string, `]X` represents zero or more occurrences of the next character (or one character from the following `]...\]` list). SPEED tries to match `pass]Xsword` with *pass*, zero or more occurrences of the next character, and *word*. Thus it would match both the nonword *password* and the word *password*. To match one or more occurrences of the next character, use `]Y` instead.

CTRL-Y

In a search string, `]Y` represents one or more occurrences of the next character (or the character from the following `]...\]` list). SPEED tries to match `pass]Yword` with *pass*, one or more occurrences of the next character, and *word*. Thus it would match both *password* and the nonword *password*. To match zero or more occurrences of the next character, use `]X` instead.

CTRL-Z

In a search string, `]Z` represents any next single character. SPEED tries to match `pass]Zword` with *pass*, any single character, and *word*. Thus `pass]Zword` would match any of *pass word*, *pass-word*, or *password*, but not *password*.

CTRL- \ list CTRL- \

In a search string, these paired control characters enclose a list of single characters that SPEED may use as a match in that position. SPEED tries to match `pass]\xyz]\word` with *pass*, any one of *x*, *y*, or *z*, and *word*. Thus it would match *passxword*, *passyword*, or *passzword*, but not *passxyzword* or *password*. To place any control character other than `]W`, `]TAB`, `]NEW LINE`, `]form feed`, `]CR`, `]K`, or `]_` in a `]...\]` list, precede it with `]W`.

You may combine the templates for additional search strategies. For example, SPEED tries to match `pass]G]N]\xyz]\word` with *pass*, any one character that is not an *x*, *y*, or *z*, and *word*, and position the CP following *pass*.

In search strings, SPEED flags any control character other than `]B` (expansion to a buffer), `]F` (expansion to a file), `]TAB` (`]I`), `]NEW LINE` (`]J`), `]vertical tab` (`]K`), `]form feed` (`]L`), or `]CR` (`]M`) as an error unless you precede it with the control characters `]G` or `]W` (or with `]I` or `]_` on models other than 6052, 6053, D100, or D200).

Executing Commands across Windows or Pages

You may move forward (but never backward) from window to window with some SPEED commands. You have already learned that the `R` command reads the current buffer into an output file, clears the buffer, and reads the next page or window of the input file into the buffer.

You have also already learned that the `Y` command clears the buffer (without writing to an output file) and reads in the next page or window of the input file.

The `N` Nonstop search command resembles the `S` command in that it looks for an occurrence of the search string in the current buffer. If it doesn't find one, however, instead of giving you an error message, it writes the buffer to an output file, clears the buffer, reads in a new page or window from the input file and continues trying to match the search string. The command keeps reading pages in and out until it either matches the search string or exhausts the file. The `N` Nonstop search command acts as an abbreviation for a series of `S` Search and `R` Read commands.

Similarly, the Q Quick search command also looks for an occurrence of the search string in the current buffer. If it doesn't find one, it clears the buffer (*without* writing to an output file) and reads in the page or window of the input file. The command keeps reading pages in and discarding them until it either matches the search string or exhausts the file. If you ask SPEED to conduct a Quick when you have a file open for updating, the buffer is not empty, local or global Update Mode is on, and the Q Quick search command is not in a Command Loop, SPEED will query

Confirm (Q-Command) ?

Type y) if you are willing to lose file contents, and any other character if you're not. The Q Quick command acts as an abbreviation for a series of S Search and Y Yank commands.

Executing the CLI

On occasion, you may need to execute a CLI command without leaving the SPEED editing session. For example, you might want a printout of your editing after you close your files, and at the same time want to retain the contents of your buffers for further editing.

You can execute one or more CLI commands without leaving SPEED by prefixing them with the SPEED X command and terminating them with a CTRL-D (instead of a NEW LINE). For example, without leaving SPEED, you can ask for

```
!XQPRINT □ rightfile $$
```

The :X command executes a copy of the parent program, usually the CLI. If you want to execute a series of parent commands, issue a :X\$\$ command. You will get the parent prompt. Assuming the CLI is the parent, you must issue a BYE) to return to SPEED.

The :Xprogram.PR\$ command is a quick way of executing a program from SPEED when the program does not require arguments from a command line.

The :X command does not actually return you to your parent process. Instead, it creates a new son process. For that reason, you cannot change parent CLI characteristics from SPEED. See the entry for the X command in Chapter 7 for details.

If you have the privilege of creating at least two son processes, you may create a secondary SPEED process with its own buffers and open files. Issue the appropriate SPEED entry command with a prefixed X command name

```
!XX □ SPEED □ [filename] $$
```

Exercises

Exercise 4-1.

Open RIGHTFILE for updating. Without typing any digits, insert on each line its line number, a period, and a tab. At the bottom of the file, again without typing any digits, make a record of its length in characters and in lines.

Exercise 4-2.

With the file still open, jump to the beginning of the buffer. Ask SPEED to look for *for you* and, only if it finds the phrase, delete it.

Exercise 4-3.

With the file still open, correct the character count. Do not issue any M or D commands, and do not insert any numbers by hand.

Since solutions are not unique, your answers may differ from these.

Answer 4-1.

!FOrightfile\$\$

*Here is a sentence for you to correct.
Here is second sentence to type in.
Here is a third sentence to type in.
Please put in the missing wrod.
Something is wrong on this line.
Here is a sentence to work with.

SD-02386

!VL\I.—\$1L\$\$

.
.

(six times)

!VL\I.—\$1L\$\$

1. Here is a sentence for you to correct.
2. Here is a second sentence to type in.
3. Here is a third sentence to type in.
4. Please put in the missing wrod.
5. Something is wrong on this line.
6. Here is a sentence to work with.

*

SD-02387

!!File□contains□\$Z\I□characters.)

!File□contains□\$VN\I□lines.)

\$\$

1. Here is a sentence for you to correct.
 2. Here is a second sentence for you to type in.
 3. Here is a third sentence to type in.
 4. Please put in the missing wrod.
 5. Something is wrong on this line.
 6. Here is a sentence to work with.
- File contains 244 characters.
File contains 7 lines.

SD-02388

(Of course, SPEED includes the characters and lines it is inserting in the count.)

Answer 4-2.

!J\$\$
!:S|G□for□you\$ *8D\$\$

1. Here is a sentence* to correct.
 2. Here is a second sentence to type in.
 3. Here is a third sentence to type in.
 4. Please put in the missing wrod.
 5. Something is wrong on this line.
 6. Here is a sentence to work with.
- File contains 244 characters.
File contains 7 lines.

SD-02389

Answer 4-3.

!S|G244\$\$
!C244\$\$Z + 3\\$\$

1. Here is a sentence to correct.
 2. Here is a second sentence to type in.
 3. Here is a third sentence to type in.
 4. Please put in the missing wrod.
 5. Something is wrong on this line.
 6. Here is a sentence to work with.
- File contains 272 characters
File contains 7 lines.

SD-02390

End of Chapter

Chapter 5

The Iteration and Flow of SPEED Commands

You can ask SPEED to execute a command string several times. You can set conditions under which SPEED stops execution or continues execution of a command string. You can arrange for SPEED to skip forward in a command string, or return to an earlier point in the string for further execution of commands.

Command Loops

A *Command Loop* is a SPEED command string enclosed in angle brackets. You set a Command Loop when you wish to execute a command string automatically several times. You control the execution of the Loop in one of two ways: either you prefix the Loop with a numerical argument, or you include a ; Conditional Termination command in an appropriate place in the command string inside the Loop. If you do not successfully control the Loop in one of these ways, the Loop may attempt to cycle endlessly.

If a search or insertion command is the last command in the Loop, remember to include the final delimiter *before* the closing bracket.

Numerical Control

You may set a Loop with a specific number in mind. For example, suppose you want to insert the word *Week* and a NEW LINE in the text four times. If you tell SPEED

```
!4<|Week|  
$>$$
```

SPEED will automatically insert

```
Week  
Week  
Week  
Week
```

and if subsequently you tell SPEED

```
!4<—Month$>$$
```

SPEED produces

```
Week  
Week  
Week  
Week  
Month Month Month Month
```

to give you a form you might use for scheduling.

You might not know a given value, but still use it. Let's automate the line-numbering task we undertook earlier. We can do that by using VN, the number of lines in the buffer as the controlling numerical argument:

```
!VN<VL\$.—$1L>$$
```

Or suppose your buffer contains just

Trapper Jeff

You could ask SPEED to *encode* your text with

```
!Z<VC\$. $1D>$$
```

where Z is the number of characters in the buffer, and VC is the ASCII digital value of a character. (You happen not to recall the value of either of them, but you can use them anyway.) SPEED responds with

```
84.114.97.112.112.101.114.32.74.101.102.102.*
```

As usual, SPEED evaluates numerical expressions used as numerical arguments. If you tell SPEED

```
!2*4<|board$>$$
```

it will insert *board* eight times.

Conditional Control

You may want SPEED to carry out a task several times, even though you can't specify the number in any way. For example, you might want SPEED to find out how many times you have used a given word. To do this, you want SPEED to search for the word as many times as it takes for the search to fail. But you can't tell SPEED how many times to look, because that's the figure you want to know. You can conduct such searches by setting the Command Loop omitting a numerical argument, but using a ; (Conditional Termination) command following the final delimiter of a search command. For example, if you had this chapter in the buffer, and wanted to know how many times a comma occurs in this page or window, you could tell SPEED

```
!OVS0$J<S,$;VIO$>VO=$$
```

That is, jump to the beginning of the buffer and enter the following Loop: search for a comma; each time you find it, increment Variable 0 by one. When the search fails, exit the Loop and tell me the current value of Variable 0. SPEED does your bidding and responds with

16

You can reverse the condition for exiting the Loop by prefixing the ; command with the colon modifier (:). In that case, SPEED exits the Loop on the first *successful* search, successful R, Y, or A command, or positive argument (n:).

Inclusion of a search command in an iteration automatically simulates the : colon modifier.

Nesting Command Loops

You can *nest* or place one Command Loop within another. For example, if you tell SPEED

```
!4<5<1.$>!|
$>$$
```

it gives you a four by five array of periods.

```
.....
.....
.....
.....
```

You may nest Loops to a depth of ten.

Conditional Execution

You can set conditions for the execution of a command line with the Conditional Execution command. You do this by having SPEED compare a specified number with zero; depending on the outcome, SPEED executes the command line or not. A Conditional Execution contains

- a numerical expression you specify
- a *quotation mark* (“)
- one of the four condition codes G, L, E, N
- the command string you want to execute conditionally
- an *apostrophe* (’)

The Execution Conditions

SPEED checks your numerical argument against one of the four conditions. If the condition holds, SPEED executes the command string between the condition code and the apostrophe. If the condition fails, SPEED skips to the apostrophe and continues execution forward from there.

SPEED compares your numerical argument to zero. Given the

Condition code	SPEED executes the string
G	if <i>n</i> is <i>greater</i> than zero
L	if <i>n</i> is <i>less</i> than zero
E	if <i>n</i> is <i>equal</i> to zero
N	if <i>n</i> is <i>not equal</i> to zero

Two arithmetically trivial examples will show how the command operates. If you tell SPEED

```
!3"G|perhaps|'$!true$$
```

SPEED makes both insertions

```
perhaps true
```

since three is greater than zero. If you tell SPEED

```
!3"L|perhaps|'$!false$$
```

SPEED skips to the apostrophe and inserts only

```
false
```

since three is not less than zero.

The Conditional Execution command may look like a lot of machinery to do very little in these examples. In fact, it is a powerful editing tool, especially if you use it in conjunction with the O (Over) command and numerical expressions that *change* while SPEED executes the command line.

Branch Over to Label

The O (Over) command allows you to skip forward in a command line, or return to an earlier point in it. The O command requires a string of 32 or fewer characters and an ESC delimiter. (It ignores the rest of a longer string.) The string you assign to the O command must be identical to a string, somewhere in the command line, that you place between exclamation points, which is called a *label*.

When SPEED encounters the O command in a command string, it scans the command line for an instance of the label. Given

```
string!label!string2$Olabel$string3
```

with the label preceding the O command, SPEED returns to the label, and begins execution of *string2* again. (Under certain conditions, which you will want to avoid, SPEED will loop continuously between *label*!

and *Olabel\$*, and never manage to execute *string3*. To avoid this, you might, for example, place an *O* command with a different label in *string2*, to transfer execution to *string3*.)

Given instead

```
string1$Olabel$string2!!label!string3
```

with the label following the *O* command, *SPEED* skips the commands in *string2* and begins execution of *string3*. (Under certain conditions, which you may want to avoid, *SPEED* may never manage to execute *string2*. To avoid this, you might, for example, place an *O* command with a different label in *string3*, to transfer execution back to *string2*.)

SPEED does not execute a *!!label!* when it encounters one. *SPEED* ignores a label except during execution of an *Olabel\$* command. (This allows you to use *!!label!* for writing comments and reminders to yourself as you compose command lines. You will find this, and the fact that *SPEED* ignores `□` and `)` outside of text strings, helpful when you begin constructing command files.)

If the *O* command is within the Command Loop, and the *!!label!* outside, *SPEED* executes the commands in the Loop until it reads the *O* command, and then jumps out to the label. It is illegal to branch into a Command Loop. If the *!!label!* is within the Command Loop, and the *O* command outside, *SPEED* displays the message

Error: Illegal command

Conditional Iteration

You now have the tools for putting together a command line that *SPEED* executes automatically a given number of times, but only if certain conditions hold.

Suppose you want (for whatever reason) to insert a series of 15 decrementing numbers. You accomplish this by

- Setting a variable, *V0*, to 15
- Setting a label
- Typing in a Backslash command
- Using a Decrement command on the *same* variable, *V0*, as an argument to the Conditional Execution
- Typing in a space-insertion command (for aesthetics)
- Placing an *Olabel\$* command *within* the Conditional Execution

If you select *dec* as your label and Variable 0 as your variable, the command line looks like

```
!15VS0$!dec!V0\V0D0''GI□$Odec$'T$$
```

Here's the sequence of events: *SPEED* sets *V0* to 15. It ignores the label, since it hasn't yet encountered an *O* command. It inserts the current value of *V0* (15). Then it decrements *V0* by one to 14. But *V0* is also the numerical argument to the following Conditional Execution. The condition is *G*, so *SPEED* checks to see whether *V0* is greater than 0. *V0* is 14, so the condition is *true* and *SPEED* begins execution of the command string in the Conditional Execution. The command line consists of an insertion and *Odec\$*, so *SPEED* inserts a space and hunts for the label *!dec!* It finds the label in the preceding command line, and begins execution from that point. So the next command to execute is the Backslash. This time, however, *V0* is 14, because of the earlier *VD* command, and this time around, *SPEED* inserts that new value. It again decrements *V0*, this time to 13. But since 13 is still greater than zero, *SPEED* again inserts a space and returns once more to the label.

This continues until *SPEED* inserts a 1 with the Backslash command, and decrements *V0* once more. This time, the decrement sets *V0* to zero. When *SPEED* checks *V0* against the *G* condition, the condition at last is *false*, and *SPEED* skips past the insertion and *O* command to the apostrophe and executes the *T* command. The result is

```
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1*
```

As you might suspect from the examples, simple as they are, you can automate many tedious editing tasks with Command Loops, Conditional Executions, and Over to label commands.

Exercises

Exercise 5-1.

Open a convenient file. Use a Command Loop to number the lines on its first page automatically. Do not type in any numbers.

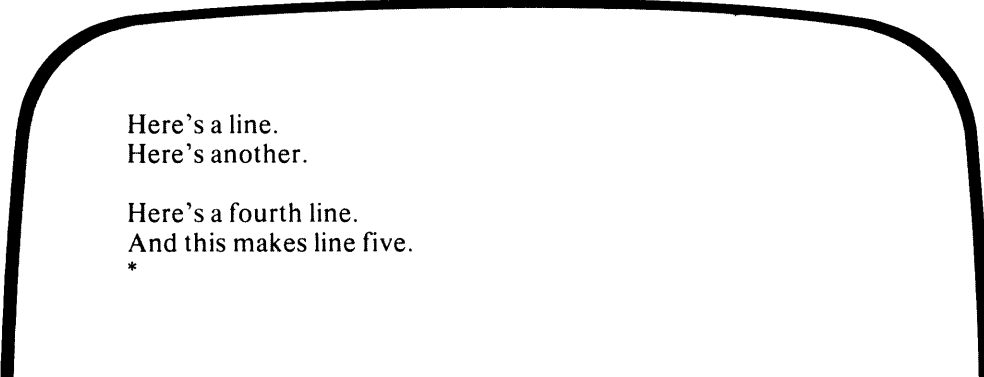
Exercise 5-2.

Using the same file, get an estimate of the number of words in the file by asking *SPEED* to count spaces and NEW LINES. Get a display each time it finds one, and get a count.

Exercise 5-3.

Using a label and a Conditional Execution, have *SPEED* insert an *incrementing* series of numbers.

Answer 5-1.

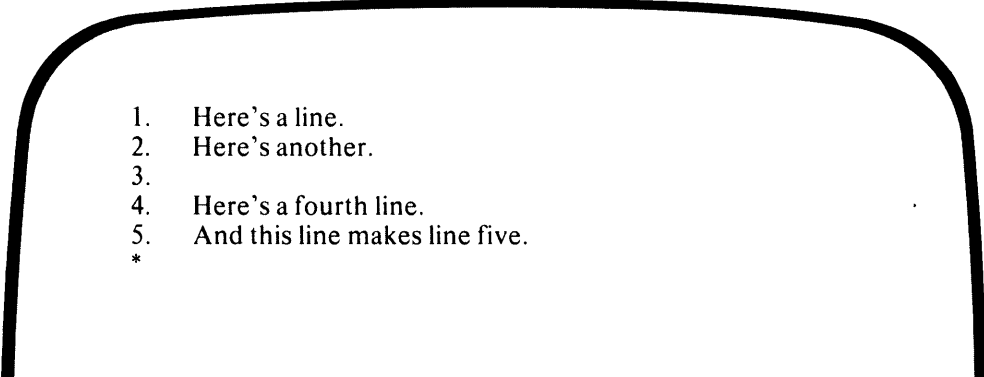


Here's a line.
Here's another.

Here's a fourth line.
And this makes line five.
*

SD-02391

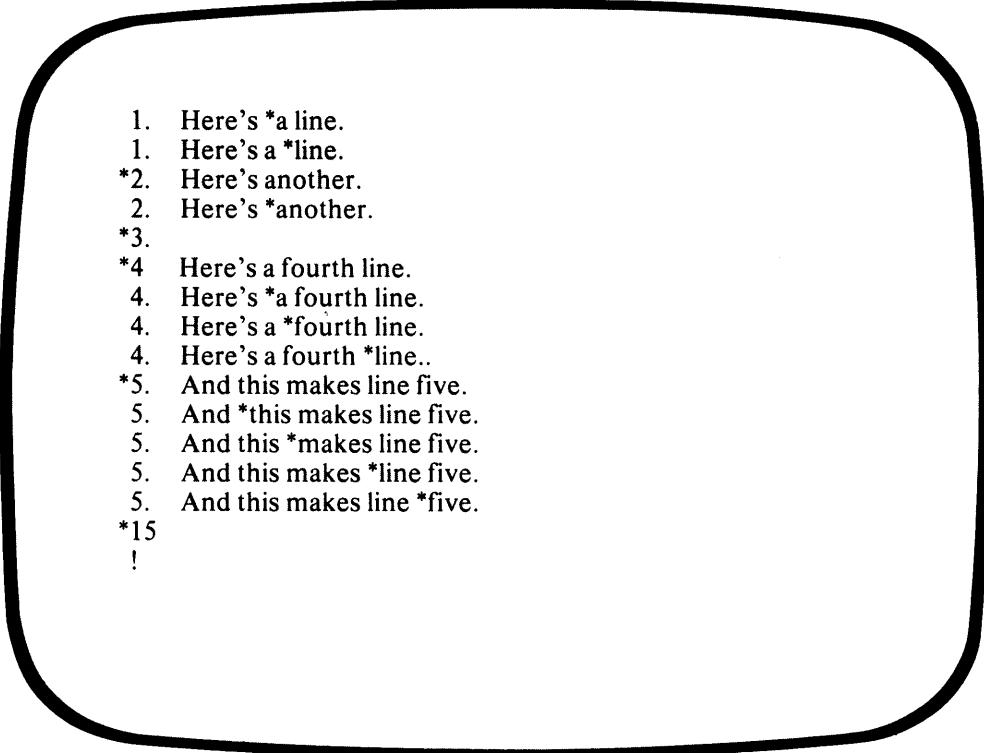
!VN<VL\|.—\$1L>\$\$

- 
1. Here's a line.
 2. Here's another.
 - 3.
 4. Here's a fourth line.
 5. And this line makes line five.
- *

SD-02392

Answer 5-2.

```
!<S|\□)  
] \$;TVIO>VO= $$
```



1. Here's *a line.
1. Here's a *line.
*2. Here's another.
2. Here's *another.
*3.
*4 Here's a fourth line.
4. Here's *a fourth line.
4. Here's a *fourth line.
4. Here's a fourth *line..
*5. And this makes line five.
5. And *this makes line five.
5. And this *makes line five.
5. And this makes *line five.
5. And this makes line *five.
*15
!

SD-02393

Answer 5-3.

```
!15VS0$!incl16-V0\VD0''NI□$Oinc$'T$$  
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15*
```

End of Chapter

Chapter 6

Advanced Editing with SPEED

In this chapter, you acquire the tools for constructing, checking, storing, and using SPEED macros.

Saving a Command Line

The `_x` command, where *x* is not the current buffer, lets you salvage a long command after an error. If, in the current buffer, you have just issued a command line that produced an unexpected result, or that you realize you will want to use again, you need only issue a `_x$$` command immediately, where *x* is a free buffer. You may then switch to Buffer *x* to edit the command line, if it needs correction, or issue the command line again with a `]Bx` command. The command line must be at least 10 characters long, or there must be no previous unsaved command line 10 or more characters long.

Suppose that you have just issued the command

```
!JVN<VL\l.→$1L>$$
```

for numbering the lines of a page, and realize that you will need it again when you have finished editing another page. Before issuing any other command, pick an appropriate buffer, say Buffer N, and tell SPEED

```
!_N$$
```

SPEED then stores your command line in Buffer N for later use.

Tracing a Command Line

You may trace the execution of a command line with the Trace Mode Toggle. The `?` command turns Trace Mode on if it was off, and off if it was on. When Trace Mode is on, SPEED displays the characters in the command line as it executes them. SPEED echoes each character including NEW LINE, form feed, space, arguments to the command, and the first letter of the command. For brevity, the rest of the characters in a command string, such as a long insertion, do not echo.

When the Toggle is on, you can trace an error in the current command line by noting where execution stopped.

Interacting with SPEED

You can provide for interactions between SPEED and its users with the `BG` Buffer Get command if you place it in a command string. For example, suppose you are in Buffer 0, and Buffer A contains

```
@ T$Choose □a □digit: □ □$1:BGNS  
!Your □number □is: □ □$I↑BN$$
```

(The commercial at (`@`) modifier of the `T` command allows you to specify the string that follows for display; the first character following the `T` command functions as a delimiter of the string, and must recur at the end of the string. The colon (`:`) modifier of the `BG` command suppresses the question mark prompt (`?`) of the `BG` command in favor of the string you display. The numerical argument to the `BG` command specifies that it will take one character as input.)

If you execute Buffer A, SPEED displays

```
Choose a digit:
```

When you type in 9, for instance, SPEED puts that character in Buffer N and then makes the insertion

```
Your number is: 9
```

SPEED inserts whichever digit you select.

Writing SPEED Commands

If you plan fairly elaborate command lines, you may wish to compose and study them, typing in a few elements at a time to see how they work, before executing them. Several reminders from earlier sections will perhaps make your task easier.

If you wish to *insert*, rather than *execute*, a delimiter, you must set a different, temporary delimiter, so as to include the ESC delimiter in the text you are typing. You do this with search and insertion commands by prefixing them with the commercial at (@) modifier. The first character you type after the command name serves as your temporary delimiter for the duration of the command. If, for example, you wish to create *as text* (perhaps in a command file), rather than execute, the command

Ccat\$dog\$\$

you must issue an I command with this command as the text string. To avoid truncating the insertion with the first occurrence of the ESC delimiter, you must issue the I command with a temporary one.

@I%Ccat\$dog\$%\$\$

SPEED then inserts the C command as text.

If you wish to *insert*, rather than *execute*, certain control keys, you must take certain precautions.

- To insert **↑B** as part of a command line you are composing, double the CTRL-B, typing **↑B↑B**
- To insert **↑F** as part of a command line you are composing, double the CTRL-F, typing **↑F↑F**
- To undertake a search for template characters, rather than the strings they are templates for, prefix them with a CTRL-W: **↑W↑N** in a search string tries to match **↑N**, rather than anything that is not the next character.
- To enter **↑T**, type CTRL-P, CTRL-T.

Thus, if you want to write a command for storage that tells SPEED to insert *abc*, the contents of Buffer 9, and *xyz*, you must type in the I command as a text string to an I command name as

Iabc↑B↑B9xyz\$\$

If you omit one of the CTRL-Bs, and Buffer 9 currently contains *000*, SPEED will insert

Iabc000xyz

instead of your intended

Iabc ↑ B9xyz

Creating and Using SPEED Macros

It is possible to reuse SPEED command lines, both in the current editing session and in subsequent ones.

Executing a Buffer

Once you have created and stored a SPEED command line in a noncurrent buffer, regardless of whether you saved the line with an **_x** command, or wrote the command line in that buffer and then switched buffers, you can reissue the command line. Simply issue **↑Bx\$\$** as a command.

For instance, if you have stored in Buffer 5 the command line

```
5 < 3 < 1000 → $ > VL \ I \
$ >
```

and, in the current buffer (with the CP at line 231), issue the command

↑!B5\$\$

SPEED will insert

```
000 000 000 232
000 000 000 233
000 000 000 234
000 000 000 235
000 000 000 236
```

For another example, if Buffer T contains the command line

↑<S↑\?.?!↑\\$.VIO>\$\$

Then your command line (assuming this chapter is in the current buffer)

↑OVS0\$J↑BT\$V0=\$\$

will get from SPEED in response

42

which approximates the number of sentences in the current buffer. (There is no reason why you should not put the rest of the command line in the buffer; go ahead and do so.)

Executing Files

You recall that, when you exit from SPEED, you lose all buffer contents that you have not written to output files. If, therefore, you wish to save a command line that you have stored in, say, Buffer T, you can switch to Buffer T, create an output file, and write the buffer contents (the SPEED macro) to the file.

```
!BST$BFWcount.spd$:P$BFC$$
```

(Use the .SPD extension to remind yourself that this is a SPEED command file.) Later in this session, or in a subsequent session, you can issue the commands in the file simply by invoking the file as a command.

```
!|Fcount.spd$$
```

The result, for this buffer in this session, is

55

(We've added a few sentences since we executed the buffer.)

When placing commands in a file, you may want to write yourself a note about what the commands do, or precautions to take with them. Since SPEED disregards characters between pairs of exclamation points, you might have inserted in the file COUNT.SPD, before you closed it:

```
!This macro counts sentences!
```

If you use notes, you can remind yourself of the file's contents when you display it.

Using the /I= Switch

You may execute a file of SPEED commands by entering SPEED with the /I= (Invocation) switch. You set the switch with the filename of the file containing the commands, and follow that with the filename of the file you want SPEED to carry out the commands on. If the file for this chapter is CHAPTER.6, then from the CLI you can issue the command

```
X SPEED /I=count.spd chapter.6
```

SPEED in this case takes no commands from the terminal, and you get the CLI prompt back as soon as SPEED has carried out the commands in COUNT.SPD on CHAPTER.6.

If you plan to use a command file with the /I= switch, include in the file an updating FU or FB command; otherwise SPEED will not update the text file with the editing commands in the command file.

An effective SPEED macro file

- has an extension, perhaps .SPD, which you give to the names of all and only SPEED command files
- contains !notes! to remind yourself of the functioning of the commands
- is formatted with spaces and NEW LINES for easy reading (when you need to remind yourself of its contents)
- contains appropriate closing commands for updating the text file

Exercises

Exercise 6-1.

In Buffer N, construct a SPEED command line that will automatically number the pages of a file.

Exercise 6-2.

Check out the commands in Buffer N on a convenient file. Edit them if necessary. When they seem right, create a command file and place them in it.

Exercise 6-3.

Bring a paged text file into SPEED. Execute your command file on it to see how it works.

Exit from SPEED. Enter SPEED with the I switch and execute your command file on a text file.

Buffer N should contain something like

```
!OVS0$<7<→$>IPage□$VIO\|)
$R;>FU$$
```

Answer 6-2.

```
!FWpage.spd$PWFC$$
```

```
!FOanyfile$$
!|Fpage.spd$$
H$$
```

```
)
) X SPEED / I = page.spd anotherfile)
```

Envoi

If you have come this far, congratulations! You have mastered the elements of SPEED. Please consult Chapter 7, "A Dictionary of SPEED," when you need further details about the use of the SPEED Text Editor.

End of Chapter

Chapter 7

A SPEED Dictionary

This dictionary contains detailed information about each SPEED command, console control key, search template control key, switch, and symbol. The entries cross-reference each other so that you can efficiently look up distinctions of function and use.

How to Use this Dictionary

If you are an experienced SPEED user and have little need of the preceding tutorial chapters, you may use this chapter as a self-contained SPEED reference manual. Change bars indicate changes in functionality from the last revision of the manual, including enhancements. Where there are multiple examples, more complex cases follow simpler ones.

If you are a beginning SPEED user, you may wish to consult individual entries as you encounter specific commands and other SPEED apparatus in the preceding chapters. Use the dictionary to confirm what you have learned. Do not try to master every detail of an entry at once. If, after careful reading, an entry still seems obscure to you, study the examples carefully and try them out step by step at the terminal. Read the entries for any related commands in order to grasp the differences between them.

Entering SPEED

You may enter SPEED in different ways for various editing tasks. You may enter with or without a filename argument, and you may use either of the two SPEED switches.

Entering without a Filename

If you enter without a filename argument,

```
) XEQ SPEED!
```

SPEED displays its revision number and gives you a prompt:

```
SPEED REV mm.nn  
!
```

SPEED is then ready to take input from the terminal. You must open or create any files you wish to work with.

Entering with a Filename

If you enter SPEED with a filename,

```
) XEQ SPEED first.1!
```

SPEED does one of two things, depending on whether the file already exists or not. If there is no such file, SPEED displays its revision number as before and asks you

```
Create new file?
```

If you type in *y*), SPEED creates an output file with that name and gives you a prompt. If you do not want a new file (because, for example, you made a typo when you tried to type in the name of an existing file), type in any other character. SPEED will not create a file but will give you a prompt.

If the file does exist, SPEED opens your file FIRST.1 for input, creates a file for output with the same name and an extension, FIRST.1.TM, turns global Update Mode on, reads in the first page, and gives you a prompt. If your file contains lowercase letters, SPEED tells you

```
** Lower case input encountered **
```

before it gives you a prompt.

SPEED strips nulls on input.

Permanence

If Permanence is On for the filename you enter, SPEED does not open the file for updating. Instead, you receive the message

```
Error: Attempt to edit a permanent file
```

and the command line, if any, aborts. If you wish to edit the file's contents, you must either change its PERMANENCE to OFF or open it in some other way (perhaps with an FR command) after you enter SPEED.

Switches

You may enhance your use of SPEED with one of two switches. If you enter SPEED with the /D (Display) switch,

```
) XEQ SPEED/D [first.l] )
```

SPEED gives you an automatic 20-line display of the current buffer contents surrounding the position of the Character Pointer after executing each command line and before restoring your prompt. This saves you the effort of constantly issuing T commands to check the contents of the buffer. For details, consult the entry for the /D switch.

If you have a file of SPEED commands that you wish to carry out on another file, you may do so by using the /I= (Invocation) switch.

```
) XEQ SPEED/I=command.file text.file)
```

This switch causes SPEED to take its commands from `command.file` rather than from the keyboard. You receive no SPEED prompt, but SPEED tells you if it encounters lowercase letters. If you wish SPEED to update `text.file` with the commands, `command.file` must contain appropriate FU or FB file-closing commands. For details, consult the entry for the /I= switch.

Exiting from SPEED

To make an orderly exit from SPEED, copy the contents of all buffers that you wish to save to appropriate output files, close all files, and issue the Exit command.

The Exit Command

The H command permits you to exit from a SPEED editing session. It closes any files that you have left open, but it does not update files or create backup files.

The H command does not save the contents of buffers. If your current buffer is not empty or you have open files, SPEED asks you

Confirm?

which requires a y) reply for execution. SPEED does *not* query you about active but noncurrent buffers, and you lose their contents permanently. Consult the H command.

A Note about Line Printer Listings

If you edit files which *contain SPEED commands*, and exit from SPEED with the intention of getting a line printer listing of the file, you should be aware that delimiters and control key characters will not show in the line printer listing. SPEED echoes these symbols at

the terminal and copies them to output files, but they do not print.

If you wish to have a listing of filename that shows CTRL characters, you might consider

- creating filename.CC (for “CTRL character”)
- copying filename to filename.CC
- replacing all ESCs (\$) in filename.CC with dollar signs (\$)
- replacing all other CTRL characters with a caret (^) and the uppercase character corresponding to the CTRL character (VC+64, for all VC less than 32)
- replacing resultant ^I, ^J, ^M with their corresponding CTRL characters (for a readable format)

The edited filename.CC will be a readable (but not executable) version of filename.

Organization of the Entries

Each entry begins with its name or characteristic shape (in color) and the type of entry (command, console control, switch, symbol, or template). A boldface sentence or phrase briefly and roughly sketches the entry.

The description of each command follows this outline:

Format

The format specifies the obligatory and optional parts of the command line, with options in italics. We list alternate formats showing choices of numerical arguments on separate lines. We show the positioning of search strings and text strings, and necessary delimiters. Where arguments must be of a specific sort, we spell them out in lowercase letters.

Function

The function describes in more detail what the command does. Where the purpose of the command is not apparent, we sketch some possible uses. In a few instances, we refer you to other commands that interact importantly with the command we are describing.

Numerical Arguments

Numerical arguments precede some but not all commands. We describe the use of possible positive, negative, and paired numerical arguments. We single out special cases (such as zero), and characterize the default value of the numerical argument when you do not enter an argument explicitly.

Symbolic Modifiers

Symbolic modifiers allow you to select an alternate delimiter, use an alternate radix, or condition the execution of the next command on the success of the preceding one. They also have specialized uses which we describe in the individual entries.

Characteristics

Characteristics of the command not easily covered in other sections occupy this section. We specify which forms of the command require a delimiter, what effect the omission of string arguments has, how some file opening and closing commands operate, and how to implement certain powerful devices such as Command Loops and Conditional Executions.

Precautions and Error Messages

Precautions and error messages attempt to warn you about characteristic difficulties you may encounter with the command, especially if you are a novice user of SPEED.

Related Commands

Other commands may be more appropriate to your needs than the command under description. We distinguish them briefly in this section.

Examples

Examples follow all else. In some cases, we first give a simple example for the neophyte, and then a more elaborate one for users with sophisticated needs.

Where appropriate, somewhat abbreviated entries describe the noncommand characters (console control keys, switches, symbols, expansion and template control keys), but every entry specifies format and function, and gives at least one example.

The Structure of SPEED Commands

In SPEED commands, numerical arguments and symbolic modifiers precede the command name; character strings and delimiters follow the command name.

Numerical Arguments

A SPEED command may begin with a numerical argument, which tells SPEED such things as how far to move the Character Pointer, how much text to delete or copy, or how many times to carry out the following command. See Table 7-1.

Under some entries, we give extended examples of the sorts of numerical arguments that can lend power and efficiency to your use of SPEED. As numerical arguments you may use not only simple numbers, but also more complex numerical expressions, values of variables and pseudo-variables, and functions you can define on them.

Take, for example, the command line

```
!Z/2MVN<VIOI>$$
```

and suppose that you are at the beginning of the buffer, that the buffer contains 26 lines, and that the current value of Variable 0 is 64. The effect of this command line is to insert the alphabet in the middle of the buffer. SPEED represents the number of characters in the buffer by Z. Dividing Z by 2 specifies the middle of the buffer, so using Z/2 as an argument to the M command places the CP at the midpoint of the buffer, where the following insertion takes place. The insertion is in a Command Loop, whose numerical argument is the value of VN, the number of lines in the buffer. Since the value of VN is 26, SPEED executes the Command Loop 26 times. The content of the Loop is the insertion of the ASCII character corresponding to an ASCII value that is in turn equivalent to the number stored in Variable 0 plus one. Since you stored 64 in Variable 0, SPEED increments Variable 0 to 65. Since A corresponds to ASCII decimal 65, SPEED inserts A the first time around. The next time around, SPEED inserts the character corresponding to the new increment, 66, which is B, and so on for 24 more times to Z.

The example exhibits overkill, since we have used simple commands in it. You will appreciate the full force of complex numerical arguments when you issue Command Loops and Conditional Executions.

Numerical arguments may either precede or follow symbolic modifiers, but must precede the command name.

Table 7-1. Numerical Arguments to Commands

By	We denote	Which refer to
n	positive integers and 0	positions and operations to the right or below the CP
-n	negative integers	positions and operations to the left or above the CP
m,n	pairs of positive integers, where m is not greater than n	a string in the buffer from the $m+1^{st}$ to the n^{th} character inclusive

Symbolic Modifiers

The symbolic modifiers (@ : &) precede the command name, if the command takes any. The ampersand modifier (&), which specifies the alternate radix, may precede both a numerical argument and the command name.

Inclusion of a search command in an iteration automatically simulates the : colon modifier.

Command Name

Every SPEED command must contain a command name. SPEED command names are short and have several forms:

- from one to four letters of the alphabet
- single nonalphabetic symbols for commands (= ; ? \) and arithmetic operators (* - + /)
- double nonalphabetic symbols for Boolean operators (^* ^- ^+ ^/)
- pairs of symbols containing other command lines:

<command-line>	Command Loop
“ $\left. \begin{matrix} G \\ L \\ E \\ N \end{matrix} \right\}$ command-line”	Conditional Execution
!!label!command-line\$Olabel\$ or Olabel\$command-line!!label!	Over to Label

Search Strings and Text Strings

Search strings and text strings, if any, follow the command name. SPEED uses the search strings you type into the command line after a C, N, Q, or S command name in attempts to match them against the text you have placed in the buffer.

SPEED uses text strings you type into the command line after Csearch□string\$ or an I, or @T command name for insertion into a buffer or for display.

Delimiters

The standard SPEED delimiter, < ESC >, echoes on the screen as a dollar sign (\$). The SPEED command line terminator, CTRL-D, which echoes on the screen as a double dollar sign (\$\$), also acts as a delimiter.

Delimiters must follow all search strings and text strings in command lines. A delimiter should follow the B? command when you do not wish to query a single buffer.

A delimiter should intervene between a Vx or Wx command and the next command name if that command name takes a numerical argument and you do not wish it to do so.

NEW LINE does *not* serve as a delimiter in SPEED. Outside of search strings and text strings, however, you may place it in command lines with no effect. Consequently, you may insert it to improve the readability of your command lines.

Command Line Terminator

The SPEED command line terminator, CTRL-D, echoes on your screen as a double dollar sign (\$\$). When you have typed in a command line and are satisfied with it, you must issue a CTRL-D for SPEED to execute the command line. NEW LINE does *not* serve as a SPEED command line terminator.

This completes the sketch of the sequence of elements in a SPEED command line.

Convention on Capitals

In the examples in this manual, we cite SPEED command names in capital letters so that you can see them easily. In fact, SPEED accepts alphabetic command names in either lowercase or uppercase. Use whichever is convenient.

A Note on Command Precedence

Please recall that if you have opened files both locally and globally in a single buffer, SPEED executes its commands *for the local file first*. Only after you close the local file does SPEED execute commands for the global file.

Functional Analysis of SPEED Commands

If you need only a general reminder of what a SPEED command does, you may find it useful to look it up in Table 7-2, “Functional Analysis of SPEED Commands,” rather than under its entry in the main body of the dictionary. Appendix C contains a convenient SPEED code graph.

Entry Sequence in this Dictionary

SPEED commands with alphabetic names precede all other entries in the dictionary. The /D and /I= switches follow the D and I commands.

Commands and modifiers with nonalphabetic names follow immediately, in the order of the ASCII digital value of the first punctuation mark.

The dictionary concludes with an alphabetic listing of control keys.

A*Command*

Append text from an input file to the buffer.

Format

A

Function

This command finds the next window or page of text in the input file and puts a copy of it at the end of the buffer, following all other buffer contents. Use the A command to add material to the buffer without affecting the current contents of the buffer.

Numerical Arguments

None

Symbolic Modifier

This command takes the colon modifier, which inhibits error messages. It returns a +1 if the command succeeds or a 0 if the command fails, so that you may use the :A form of this command as a numeric argument to the next command; see the colon symbol in this dictionary and the second example below.

Inclusion in an iteration automatically simulates the : colon modifier.

Characteristics

If the remainder of the input file contains no form feeds and Window Mode is set at 0, the A command appends the rest of the input to the end of the buffer. It also appends the rest of the file if Window Mode is set to *n* and fewer than *n* lines remain in the input file. If you are in page mode (WM=0), and the file contains form feeds, it appends to the next form feed (the ↑L never shows).

If you have both local and global input files open for the current buffer, this command appends a page from the *local* file rather than from the *global* file.

The A command appends to the *end* of the buffer rather than at the position of the CP. This command does not affect the position of the CP. (If the CP is at the end of the current buffer when you issue this command, it will immediately precede the appended page or window.)

Precautions and Error Messages

If SPEED displays the error message

Error: No more characters in input file

it merely means that you have already brought the last page or window of the input file into the buffer.

Related Commands

The A command does not copy previous buffer contents to an output file or clear the buffer. To clear the buffer of current contents while getting new input, see the Y command. To copy the previous buffer contents to an output file, and get new input, see the R command. To copy the buffer *without* getting new input, see the P and PW commands. To copy the buffer and the rest of the input file to an output file, see the E command.

Examples

```
!FC$FRdistrib$A$$
```

You close global files without clearing the buffer, open the file DISTRIB and append the first page or window of it to the buffer. After performing some editing tasks, you find yourself in the text at

**Consider:*

```
!:A*9M$$
```

*Consider:**

You issue the command to append a page, using the colon modifier. If the command succeeds, the CP will move +1 times 9 = 9 places to the right. If it fails, the CP will move 0 times 9 = 0 places to the right. It succeeds, and you have repositioned your CP where you wanted it in case there was another page to append.

BC*Command*

Buffer Copy: copy from the current buffer to another buffer.

Formats

BCbuffer-name
nBCbuffer-name
-nBCbuffer-name
m,nBCbuffer-name

Function

This command copies some or all of the contents of the current buffer from that buffer to a second buffer, **buffer-name**. The only legal buffer names are the digits 0 through 9 and single letters of the alphabet.

The BC command clears the destination buffer of previous text. It does not delete the copied text from the current buffer. Use this command to copy text from the current buffer to other buffers, where, for instance, you can perform further editing on the text and insert it in various open local output files.

Numerical Arguments

This command takes positive, negative, and paired numerical arguments. It copies text to Buffer *x* as follows:

nBCx, except for 0BCx
text from the current CP position up to the *n*th NEW LINE. That is, it copies the next *n* lines of text, but the first line copied will only contain text from the CP forward

1BCx
text from the CP to the end of the line

0BCx
text from the beginning of the current line up to the current CP position

-nBCx
text from the beginning of the *n*th line preceding the current line through *n* NEW LINES up to the position of the CP; that is, text from the previous *n* lines and, if the CP is not at the beginning of the current line, text on the current line up to the position of the CP

-1BCx
text from the previous line and the current line up to the CP

m,nBCx
text from the character *after* the *m*th character up to and *including* the *n*th character. Simple numbers refer to positions in the current buffer. You may also use numerical expressions containing arithmetic operators, and the values of variables and pseudo-variables such as V0, ., and VN. For example:

17,41BCx copies from the 18th character in the current buffer through the 41st.

17,20+V0BCx (V0 positive) copies from the 18th character in the buffer through the character whose position from the beginning of the buffer is 20 plus the current value of Variable 0.

using your current CP position, 0,.BCx copies from the beginning of the buffer up to the position of the CP, and .,ZBCx copies from the position of the CP to the end of the buffer.

#BCx
abbreviates 0,ZBCx, which copies the entire current buffer to buffer *x*.

The default value
for this command (issuing the command without a numerical argument as BCx) is also 0,Z, which represents the entire buffer.

BC (continued)

Symbolic Modifiers

None

Characteristics

The BC command does not require a delimiter.

This command does not affect the position of the CP in the current buffer. The CP of the destination buffer will precede the copied text.

Precautions and Error Messages

The value of *m* in *m,nBCx* commands must not be greater than the value of *n*. If it is, SPEED will send you the message

Error: First argument greater than second argument

Recalculate and reissue the command, if you still wish to execute it.

If you inadvertently try to copy to the current buffer, SPEED will send you the message

Error: Attempt to delete current buffer

Related Commands

To copy text to other buffers and delete it from the current buffer, see the BT command.

If the destination buffer already contains text, BC deletes it. If you wish to save the contents of that buffer, you may wish to copy your new text to an inactive buffer. To find out which buffers are active, issue the B? command.

If you want to copy text from another buffer *into* the current buffer and you don't want to delete the contents of the current buffer, see the CTRL-B template in this dictionary. Do *not* attempt to add text to the current buffer by switching to a new buffer containing text, issuing a BC command for the original buffer, and switching back. You will destroy all material in the current buffer except the copied text.

Example

```
!.-10,ZBC1$3BC2$2L$40M$0BC3$$
```

You copy to Buffer 1 text from the 10th character preceding the CP to the end of the buffer. Next you copy the current line to the right of the CP, and the next two lines, to Buffer 2. Then you go down two lines, move 40 characters to the right, and copy them to Buffer 3.

BFB*Command*

Buffer File Backup: create a local backup file and close local files.

Format

BFB

Function

This command copies the current buffer and remainder of the local input file, if any, to the local output file, retains a copy of the input file, clears the current buffer, and closes both files. The effect of this command is to update your old local file with the changes you made. It retains a copy of the original input file as a backup file if Update Mode is on. Using this command, you

- place the edited contents of the input file in the new output file
- close both local files
- if Update Mode is on, rename the original input file as a backup file, `filename.BU`
- if Update Mode is on, rename the output file with the original name, `filename`, of the input file

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

The **BFB** command works in several steps. When you open a file with the **BFO** command, you automatically turn Update Mode on. **SPEED** then creates a temporary local output file with the same name as your input file and a `.TM` extension. Later, your **BFB** command

- copies the current buffer and the remainder of the local input file, `filename`, into `filename.TM`
- clears the current buffer

- closes both `filename` and `filename.TM`
- renames `filename` to `filename.BU` and
- renames `filename.TM` to `filename`

Update Mode then turns off, and you may issue new local file opening commands.

Precautions and Error Messages

You can execute the **BFB** command successfully only if there is an output file available. The command will close a file opened with the **BFO** (Buffer File Open) command, the **BFW** (Buffer File Write) command, or the **BFNW** (Buffer File New Write) command. It does *not* close a file opened with the **BFR** (Buffer File Read) command unless there is a corresponding output file, for instance, one opened with the **BFW** command.

If you issue the **BFB** command in a buffer that has a global output file open, but no local output file open, **SPEED** will display the message

Error: No open file

and will abort the rest of the command line. Reissue the next command, if you still wish to execute it.

If you try to issue the **BFB** command when you have no open files, you will receive the message

Error: No open file

and the rest of the command line will not execute. Reissue the next command, if you still wish to execute it.

Related Commands

If you wish to reject the editing you have done and retain the old file without changes, see the **BFC** command. If you do not wish to keep the old version of the file, see the **BFU** command.

BFB (continued)

Example

!F?\$B?\$\$

Global:

Input File - None

Output File - None

Local:

Input File - :UDD:LUCI:SCHEd

Output File - :UDD:LUCI:SCHEd.TM

Update Mode On

= >Buffer 0 - 666

!BFB\$BFOillo\$\$

User Luci checks the status of her files and buffers. She has a locally open file. Only the current buffer is active. She wants to save the local input file as a backup. She closes the local files with the **BFB** command, renaming them appropriately. Next, she opens the file **ILLO** locally for updating. When she again issues the **F?B?\$\$** command line, she sees

Global:

Input File - None

Output File - None

Local:

Input File - :UDD:LUCI:ILLO

Output File - :UDD:LUCI:ILLO.TM

Update Mode On

= >Buffer 0 - 498

BFC

Command

Buffer File Close: close local input and output files.

Format

BFC

Function

Use this command to close local input and output files. Since this command does not clear the buffer or copy it to the output file, use it only when you have in the buffer and in the output file exactly what you want or wish to keep in the input file without making editing changes.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

The **BFC** command only closes the current local files. It does *not* copy from the buffer to the output file, clear the buffer, rename input or output files, or create a backup file

Precautions and Error Messages

This command closes any local file, no matter how you opened it. If you have turned Update Mode on by opening a file with a **BFO** command, you have separate files **filename** and **filename.TM** when you exit from **SPEED**. The **.TM** file contains the results of your editing if you copied them from the buffer. (See the various output commands **E**, **P**, **PW**, **R** for transferring material from the buffer to an output file.)

The **BFC** command has no effect on global files.

You will not receive a message from **SPEED** if you try to execute **BFC** with no file open.

Related Commands

If you wish to copy and clear the buffer, see the **BFU** (Buffer File Update) command. If you wish to create a backup file, see the **BFB** (Buffer File Backup) command. If you wish to close current files and open new ones, see the **BFNR** (Buffer File New Read) and **BFNW** (Buffer File New Write) commands.

BFC (continued)

Example

```
!BFRprime$A$#T$$  
!BFC$$
```

You open the local input file PRIME, append a page or window to the buffer, display it, and discover that PRIME is not the file you want. You close the file, and are now ready to open another local input file.

BFNR

Command

Buffer File New Read: open a new local file for input.

Format

BFNR [*filename*]

This command closes the old local input file, if any, and opens a new local input file, *filename*, if you specify one. The file must already exist. Use this command when you wish to open, edit, and close a series of input files.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

This command requires a delimiter.

If you omit *filename* and Update Mode is off, the command merely closes the current local input file, if any.

The BFNR command does *not* copy from the input file to the buffer (see the input commands A, R, and Y), copy the buffer to an output file, or clear the buffer

Precautions and Error Messages

If the BFNR command does not open a new input file for you, you may have turned Update Mode on by issuing a BFO command. You will receive the message

Error: Update mode on

and the rest of the command line will abort. Reissue the next command, if you still wish to execute it.

This command has no effect on local files opened with a BFW or BFNW command.

The BFNR command has no effect on files opened globally.

The BFU and BFB commands will not close files opened with the BFNR command unless you have opened a local output file. SPEED will send you the message

Error: No open file

and the rest of the command line will abort. Reissue the next command, if you still wish to execute it.

BFNR (continued)

Example

```
!BFRyr81$$  
.      (editing commands)  
.      (editing commands)  
!BFNRyr82$$  
.      (editing commands)  
.      (editing commands)  
!BFNRyr83$$  
.      (editing commands)  
.      (editing commands)  
!BFNR$BFWyr81_83$$
```

You open, use, and close a series of local input files. You close your last file by issuing the command without a filename argument, and open a local output file.

BFNW

Command

Buffer File New Write: create a new local file for output.

Format

BFNW [*new.filename*]

Function

This command closes the old local output file, if any, and opens *new.filename*, which you create with this command. The filename you select must not already exist. Use the **BFNW** command when you wish to create, open, edit, and close a series of output files.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

This command requires an ESC delimiter.

If you omit *new.filename* and Update Mode is off, this command merely closes the current local output file, if any.

The **BFNW** command does *not* copy from the buffer to the output file, and does not clear the buffer. (See the output commands **E**, **P**, **PW**, and **R**.)

Precautions and Error Messages

If you have turned Update Mode on by issuing a **BFO** command, you will receive the message

Error: Update mode on

and the rest of the command line will abort. Reissue the next command, if you still wish to execute it.

This command has no effect on local files opened with a **BFR** or **BFNR** command.

The **BFNW** command has no effect on files opened globally.

BFNW (continued)

Example

```
!BFWexample.A$$  
.      (editing commands)  
!  
!BFNWexample.B$$  
.      (editing commands)  
!  
!BFNWexample.C$$  
.      (editing commands)  
!  
!BFNW$BFRbiblio$$
```

You create, open, use, and close a series of local output files. You close your last file by issuing the command without a filename argument. After you finish with that editing task, you open a new local input file, BIBLIO, to begin another.

BFO

Command

Buffer File Open: open local file for update.

Format

BFOfilename\$

Function

Use this command when you wish to make changes in an already existing file, filename. The BFO command

- opens filename for local input
- turns Update Mode on for filename
- creates a new local file, filename.TM, for output
- yanks the first page or window of filename into the buffer

You are now ready to carry out your editing tasks on filename. When you are through, you use either the BFB (Buffer File Backup) command or the BFU (Buffer File Update) command to complete the editing cycle.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

This command requires an ESC delimiter.

The BFO command turns Update Mode on. When Update Mode is on, the file is opened *exclusively*. No one, including yourself, may have additional access to the file.

Precautions and Error Messages

If you attempt to open a file with the BFO command, and the file has PERMANENCE ON, SPEED will display the message

Error: Attempt to edit a permanent file

and abort the command line.

BFO (continued)

If you attempt to open two files for updating in the same buffer with an FO and a BFO command, you lose text from one of the two files. Avoid using BFO and FO commands in the same buffer. If you want to open a second file for updating without closing the first, switch to an available buffer with the BS command and open the file locally in that buffer with a BFO command.

When Update Mode is on for a file, no one can have additional access to the file, including yourself if you try, for example, an X TYPE filename or X QPRINT filename command.

When you open a file with the BFO command, you preserve the record type (data sensitive, dynamic, fixed length, variable) of the input file and of the associated output file, either filename.BU or filename.TM (see the BFB and BFU commands). Opening for update also preserves file type, record length if fixed, element size, and UDA (User Data Area).

You cannot issue BFNR or BFNW commands while Update Mode is on, whether or not you issue them with filename arguments. You must first close the file opened with the BFO command. If you do not, you will receive the message

Error: Update mode on

If you try to open a second local file in the current buffer when Update Mode is on, you will receive the message

Error: File already open

whether you attempt the opening with BFO, BFR, or BFW.

SPEED strips nulls on input.

Related Commands

If you don't know whether you want to open a file locally or globally, consult the section "Using Local Files" in Chapter 3.

Instead of updating a current file, you may want to create new files by rewriting or combining old ones. If so, see the BFR (Buffer File Read) and BFNR (Buffer File New Read) commands for opening input files, and the BFW (Buffer File Write) and BFNW (Buffer File New Write) commands for creating output files.

Use one of the three local closing commands when you finish editing the file you opened with the BFO command. If you are happy with the changes you made, see the BFU (Buffer File Update) command. If in addition you want to keep the previous contents of the file, see the BFB (Buffer File Backup) command. If you decide to reject your changes and keep the original file, see the BFC (Buffer File Close) command.

Example

```
!BFOshort$BS9$FOlong$F?$$
```

Global:

```
Input File - :UDD:PAT:LONG
Output File - :UDD:PAT:LONG.TM
Update Mode On
```

Local:

```
Input File - :UDD:PAT:SHORT
Output File - :UDD:PAT:SHORT.TM
Update Mode On
```

User Pat opens exclusively the local file SHORT, switches to Buffer 9, opens exclusively the global file LONG, and verifies what she has done by issuing the F? command to get a display.

BFR*Command***Buffer File Read: open local file for input.**

Format

BFRfilename\$

Function

This command opens a file for local input, where filename names the file you wish to open. The file must exist. Use this command to open a file especially when you plan to add to another file or create a new output file from its contents, rather than update the input file itself. Since Update Mode will be off, you will have to create a file for output, if you want one, with the BFW (Buffer File Write) or BFNW (Buffer File New Write) command.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

This command does *not* copy from the input file to the buffer. See the input commands A, R, and Y.

This command does *not* copy from the buffer to the output file, and it does not clear the buffer.

Precautions and Error Messages

The BFU and BFB commands will not close a file opened with the BFR command unless you have opened a local output file. SPEED displays the message

Error: No open file

and aborts the rest of the command line. Reissue the next command, if you still wish to execute it.

Related Commands

To open a series of files locally for input, see the BFNR (Buffer File New Read) command. To open a file locally for updating, see the BFO (Buffer File Open) command.

Example

```
!BFRsupply$A$#T$BFWorder$$
```

You open the file SUPPLY, append a page or window to the buffer, display it, and create an output file ORDER whose contents you will edit from SUPPLY.

BFU*Command***Buffer File Update: update and close local files.**

Format

BFU

Function

This command updates your local input file and retains the original filename. It renames a new local file for output with the same name as the old local input file. The new local output file contains the material of the input file as you have edited it during the current session.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

The **BFU** command carries out the updating in several steps. When you open a file with the **BFO** (Buffer File Open) command, you automatically turn Update Mode on. **SPEED** then creates a temporary local output file, **filename.TM**, with the same name as your input file and a **.TM** extension. When you subsequently issue the **BFU** command, it

- copies the current buffer and the remainder of the local input file, **filename**, into **filename.TM**
- clears the current buffer
- closes both **filename** and **filename.TM**
- deletes **filename**
- renames **filename.TM** to **filename**
- turns Update Mode off

You may issue new local file opening commands, if you wish.

Precautions and Error Messages

The **BFU** command applies only to open files with Update Mode on. If global Update Mode is on, but local Update Mode is off, the **BFU** command applies to global files.

You can execute the **BFU** command successfully only if

- local Update Mode is on, or
- a local output file is open, or
- global Update Mode is on, or
- a global output file is open

The **BFU** updates and closes files opened with the **BFO** (Buffer File Open) command, the **BFW** (Buffer File Write) command, and the **BFNW** (Buffer File New Write) command, or with the corresponding global commands (**FO**, **FW**, **FNW**). The **BFU** command does *not* close a file opened with the **BFR** or **BFNR** commands or their global counterparts (**FR**, **FNR**) unless you have opened a corresponding output file, for instance, one opened with the **BFW** command.

If you have no open local or global file, or if you shift from the buffer in which you opened a global file for updating to a buffer which has no local output file open, and issue a **BFU** command in that buffer, **SPEED** displays the message

Error: No open file

and aborts the rest of the command line. Reissue the next command, if you still wish to execute it.

Related Commands

If you wish to reject the editing you have done and retain the local input file without change, see the **BFC** (Buffer File Close) command. If you wish to keep both the old and new versions of the file, see the **BFB** (Buffer File Update) command.

BFU (continued)

Example

```
!F?$B?$$
```

Global:

Input File - None

Output File - None

Local:

Input File - :UDD:CHRIS:DESIGN

Output File - :UDD:CHRIS:DESIGN.TM

Update Mode On

```
= >Buffer 0 - 4702
```

```
!BFUH$$
```

User Chris checks the status of her files and buffers. She has a locally open file. Only the current buffer is active. She updates her file, automatically clearing the buffer and closing both files, and exits from SPEED.

BFW

Command

Buffer File Write: create a local file for output.

Format

```
BFWnew.filename$
```

Function

This command creates and opens a local output file, where `new.filename` names the file you want to open and create. The filename you select must not already exist. Use this command to create a new output file in which, for example, you might store the edited contents of old files.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

This command requires an ESC delimiter.

Precautions and Error Messages

This file does *not* copy the buffer to the output file, and it does not clear the buffer. See the output commands **E**, **P**, **R**, and **Y**.

Update Mode must be off to issue this command. The **BFW** command does not turn Update Mode on. If you wish to have an input file, you must issue a **BFR** or **BFNR** command.

You must close the current local output file before you can execute another **BFW** command.

Related Commands

To update an existing file, rather than create a new output file, see the **BFO** command. To create and open local output files in series, see the **BFNW** command.

BFW (continued)

Example

```
!BFRsurvey$A$$  
.  
    (editing commands)  
.  
!BFWreport$EFC$$
```

You open the input file SURVEY locally, append a page or window to the buffer, edit in the buffer, create and open the file REPORT, copy the buffer to it, and close both files.

BG

Command

Buffer Get: get a line and copy it to a buffer.

Formats

BGbuffer-name
nBGbuffer-name

Function

Read a line of characters that the user types in at the terminal and store it in the specified buffer, buffer-name. The only legal buffer names are the digits 0 through 9 and single letters of the alphabet.

Issue the BG command and a CTRL-D. SPEED will display a question mark and space as a prompt. Enter the characters from the keyboard. You regain the standard ! prompt after SPEED executes this command.

Use this command to store in another buffer a line you plan to use during the current editing session, either as a recurring line of text or as a command line you will need later.

Numerical Arguments

The BG command accepts single numerical arguments

nBGx
stores in buffer *x* the *n* characters you type.

OBGx
accepts up to 136 characters, the default record length. This command accepts as a character a data-sensitive delimiter such as ESC.

SPEED also uses the default record length, 136, if *n* is negative or if you give the command no numerical argument.

Symbolic Modifier

This command accepts a colon modifier before the numerical argument. The :nBGx or :BGx form of the command inhibits SPEED from displaying the ? prompt. This allows you to define your own prompt. See the @T form of the T command.

BG (continued)

Characteristics

This command does not affect the CP in the current buffer. The CP of the destination buffer precedes the stored characters.

When you issue a positive, fixed-length `nBGx` command, you terminate input if you issue a `CTRL-D`. Data-sensitive `OBGx` and `BGx` commands require a terminating `CTRL-D`.

This command accepts as a character a data-sensitive delimiter such as `ESC` in the string you type in, allowing you to store a command line of more than one command.

Precautions and Error Messages

This command destroys the previous contents of the destination buffer.

The `BG` command does *not* enter the typed line in the current buffer.

Do *not* issue a `CTRL-D` at the end of the input string in a positive, fixed-length `nBGx` command.

If you issue nonzero `nBGx` (fixed length input) commands within a Command Loop, and type text in excess of the length, the excess text passes to the next execution of the `BG` command in the loop. Outside of a Command Loop, if you exceed the number `n` (or the default value) in the line you type, `SPEED` will try to interpret the next symbols entered as commands. If you do not keep track of your text, you may issue inadvertent `D`, `K`, or `Y` commands. You will receive error messages appropriate to inadvertent commands that abort.

Related Commands

To copy text from the current buffer to another buffer, see the `BT` and `BC` commands. For other strategies for storing and reusing lines, see the `CTRL-B` and `CTRL-F` templates in this dictionary.

Examples

```
BG1$$
```

```
? _
```

```
Wait [ ] for [ ] the [ ] signal [ ] to [ ] turn [ ] the [ ] page. $$
```

```
.  
.  
.  
.  
.  
(editing commands)
```

```
I|B1$$
```

```
.  
.  
.  
.  
.  
(editing commands)
```

```
I|B1$$
```

While you are constructing a timed test at the terminal, you realize that you are going to use a line repeatedly. You store it in a buffer with the `BG` command and insert it in your text when you need it with an `I` command using a `CTRL-B` template.

```
! :BG2$$
```

```
_ (system response)
```

```
!!Lines [ ] = [ ] $VN \ | [ ] and [ ] characters [ ] = [ ] $Z \ $$
```

You wish to keep a record in your text of the text's length as you go along. You issue the `BG` command above and from time to time insert the information with a `CTRL-B`. `SPEED` inserts

```
Lines = 103 and characters = 2882
```

if that is your place in the text.

You need to type lines of text that are mostly repetitive but contain a variation at one point. You type in answers to some problems this way

```
!3<|The [ ] answer [ ] is [ ] $BGQ|BQ$> $$
```

BG (continued)

Each time you get the question prompt, you type in the next answer and a CTRL-D:

```
?_
?_ .1047<CTRL-D> ?_
?.1047? 31<CTRL-D> ?_
?.1047?31? 6.8584<CTRL-D> !_
?.1047?31?6.8584! $$
```

You thereby create the text

```
The answer is .1047
The answer is 31
The answer is 6.8584
```

You must convert a list of words from British to American spellings. You must change *flavour* to *flavor* without changing *four* to *for*. You want SPEED to ask you whether to make the change or not each time. You specify

```
!<Sou$;T@T$Delete?$
:1BGDBSDVCVS0$BS0
V0-89'E-1D'>$$
```

Your Command Loop tells SPEED: find successive instances of *ou*. Display each one and the question *Delete?* Read my one-character reply into Buffer D. Switch to Buffer D and store the ASCII decimal value of the character in Variable 0. Switch back to Buffer 0. If V0=89 -- that is, if the character was Y (for *yes*) -- delete the preceding character *u*. If V0 is anything else, continue with the search.

Given a list containing *hour*, *behavioural*, and *valour*, in that order, if you make the right choices, SPEED ultimately displays

```
hou*r
Delete?Nbehavioural
Delete?Yvalou*r
Delete?Y
!_
```

Remember in this fixed-length case *not* to enter a CTRL-D after your Y or N response.

BK

Command

Buffer Kill: kill another buffer.

Format

BKbuffer-name

Function

This command deletes a buffer, *buffer-name*. The only legal buffer names are the digits 0 through 9 and single letters of the alphabet. The buffer must not be the current buffer.

Use this command when you no longer need the contents of a buffer other than the one you are in.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

You may have files open locally in the buffer you kill. This command does not copy from the noncurrent buffer to an output file, nor does it close files.

Precautions and Error Messages

If you issue a BK command with an illegal buffer name, SPEED displays the message

Error: Illegal buffer name

If you try to kill an inactive buffer, SPEED displays the message

Error: Buffer is inactive

Related Commands

You do not need to kill buffers before copying to them. See the BC and BT commands.

You do not need to kill buffers before exiting from an editing session; SPEED does that automatically. See the H command.

To kill any or all lines in the current buffer, see the K command.

BK (continued)

Example

```
!B?$$
```

```
= >Buffer 0 - 1532  
    Buffer 1 - 0  
    Buffer 4 - 767  
    Buffer 5 - 14  
    Buffer C - 5
```

```
!BK1BK5BKC$$
```

You inspect the status of your buffers and recall that only Buffer 0 and Buffer 4 contain material you need to retain. You kill the other buffers.

BS

Command

Buffer Set: switch to another buffer.

Format

BSbuffer-name

Function

This command makes a new buffer, **buffer-name**, the current buffer. The only legal buffer names are the digits 0 through 9 and single letters of the alphabet.

Use an additional buffer, switching to and from it, when, for example, you wish to

- examine and edit text that you place there with a BC, BG, BT or `_buffer-name` command
- examine and edit a file opened locally in that buffer

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

You do not modify the contents of a buffer by switching to or from it.

You may have up to 36 buffers active in an editing session.

When you set the current buffer to *n*, SPEED activates automatically the buffer that you switch to.

This command does not affect the positioning of CPs. SPEED saves the CP position in each buffer and restores it when you switch into the buffer.

This command does not affect the status of files. The local files open in the current buffer will still be open when you switch back to that buffer.

If you issue no BS commands, you remain in the default buffer, Buffer 0.

BS (continued)

Precautions and Error Messages

If you issue the command with an illegal buffer name, SPEED displays the message

Error: Illegal buffer name

Related Commands

To find out the status of your buffers, issue a B? command.

Example

```
!FObiblio$BS1$BFOarticles$$
```

While in your current buffer you open the file BIBLIO for updating. You switch to Buffer 1 and open the file ARTICLES locally for updating.

BT	<i>Command</i>
Buffer Take: take text to another buffer.	

Formats

BTbuffer-name
nBTbuffer-name
-nBTbuffer-name
m,nBTbuffer-name

Function

This command transfers some or all of the contents of the current buffer to a second buffer, *buffer-name*. The only legal buffer names are the digits 0 through 9 and single letters of the alphabet.

This command deletes from the current buffer the text it takes to the destination buffer.

Numerical Arguments

This command takes positive, negative and paired numerical arguments. It takes text out of the current buffer and puts it into Buffer *x* as follows

nBT_x, except for OBT_x
from the current CP position up to the *n*th NEW LINE. That is, it takes the next *n* lines of text, but the first line copied will contain text only from the CP forward.

1BT_x
from the CP through the next NEW LINE.

OBT_x
from the beginning of the current line up to the current CP position.

-nBT_x
from the beginning of the *n*th line preceding the current line through *n* NEW LINES up to the position of the CP. That is, it takes text to buffer *x* from the previous *n* lines and, if the CP is not at the beginning of the current line, text on the current line up to the position of the CP.

-1BT_x
from the previous line and the current line up to the CP.

BT (continued)

m,nBTx

from the character *after* the m^{th} character up to and *including* the n^{th} character. Numbers in this command refer to positions in the current buffer. You may use numerical expressions containing arithmetic operators, and the values of variables and pseudo-variables such as V0, ., and VN. For example

19,43BTx takes from the 20th character in the current buffer through the 43rd.

23,20+V0BTx takes from the 24th character in the buffer through the character whose position from the beginning of the buffer is 20 plus the current value of Variable 0.

0,.BTx takes from the beginning of the buffer up to the position of the CP, and .,ZBTx takes from the position of the CP to the end of the buffer.

The command #BTx

abbreviates 0,ZBTx, which takes the entire current buffer to buffer x .

The default value

for this command (issuing the command without a numerical argument as BTx) is 0,Z, which refers to the entire buffer.

Symbolic Modifiers

None

Characteristics

This command does not require a delimiter.

This command does not affect the position of the CP in the current buffer relative to remaining text, although the value of the CP position (denoted by .) may change. The CP of the destination buffer precedes the text.

Precautions and Error Messages

If you inadvertently try to take text to the current buffer, SPEED displays the message

Error: Attempt to delete current buffer

In the m,nBTx variant of this command, the value of m must not be greater than the value of n . If it is, SPEED displays the message

Error: First argument greater than second argument

Recalculate and reissue the command, if you still wish to execute it.

Do *not* attempt to add text to the current buffer by switching buffers, issuing a BT command and switching back. You will destroy all material in the current buffer except the transferred text.

Related Commands

To copy text to other buffers but retain it in the current buffer, see the BC command.

To insert text *into* the current buffer *without deleting its present contents*, use an I|Bx command. See the |Bbuffer-name expansion in this dictionary.

Example

```
!. + 12,ZBT1$-4BT2$0BT3$$
```

You take to Buffer 1 text from the 12th character following the CP to the end of the buffer. Next you take the four lines preceding the current line, and the current line up to the CP, to Buffer 2. Then you take the new last line of the current buffer to Buffer 3.

B? *Command*
Buffers? give the status of active buffers.

Formats

B?\$
B?buffer-name

Function

The B? (Buffers?) command tells you which of the 36 buffers are active, how many characters each contains, and which buffer is current. The only legal buffer names are the digits 0 through 9 and single letters of the alphabet.

Use this command to keep track of your buffer use, especially during complex editing tasks which require the manipulation of several buffers.

Numerical Arguments

None

Symbolic Modifiers

This command takes the ampersand (&) modifier. When you issue a &B? command, SPEED displays the character count in the alternate radix. (The alternate radix is octal unless you reset it with a WR command.)

Characteristics

If you have not switched buffers, you are in the default buffer, Buffer 0.

The B? form of this command lists all active buffers. It requires a delimiter, since it tries to interpret any other character as the name of a buffer. An arrow points at the current buffer. You may use this form of the command in any buffer to determine the status of all buffers.

This command does not tell you whether a buffer has files open locally.

Precautions and Error Messages

Remember that the question mark *precedes* the buffer name.

SPEED displays the message

Error: Illegal buffer name

if you follow B? with any character other than a letter, a digit, ESC, or CTRL-D.

Related Commands

To keep track of which files you have open in a buffer, see the F? (Files?) command.

Examples

You decide to check your buffers. You enter

!B?\$\$

```
= >   Buffer 0 - 1728
      Buffer 2 - 144
      Buffer P - 0
      Buffer D - 12
```

Your current buffer, Buffer 2, contains 144 characters. Buffer 0 contains 1728 and Buffer D contains 12. Buffer P contains no characters but has been active at some time in the editing session, or it would not be listed.

!B?1\$\$

Buffer 1 - 3

Buffer 1 contains 3 characters.

!B?%\$\$

Error: Illegal buffer name

You forgot to release the shift key to check on Buffer 5.

!B?D\$\$

Buffer D - Inactive

You have not placed any text in Buffer D, or used Buffer D in any way (for example, a BSD command) during the editing session.

C*Command*

Change one string of text into another.

Formats

C [string1] \$ [string2] \$
nC [string1] \$ [string2] \$
-nC [string1] \$ [string2] \$
m,nC [string1] \$ [string2] \$

Function

This command conducts a search in the text for the first argument *string1*. If the command finds the string, it replaces that string with the second argument, *string2*. The range in which the C command conducts the search depends upon the numerical arguments you furnish to the command.

Numerical Arguments

This command takes positive, negative, and paired numerical arguments.

- OCstring1\$string2\$
searches from the beginning of the current line up to the CP position.
- 1Cstring1\$string2\$
limits the search from the CP position through the end of the current line (NEW LINE).
- nCstring1\$string2\$, except for OCstring1\$string2\$,
limits the search for *string1* from the current CP position up to the *n*th NEW LINE forward. That is, it searches through the next *n* lines forward, but searches the first line only from the CP position forward.
- 1Cstring1\$string2\$
limits the search to the preceding line and the current line up to the CP.
- nCstring1\$string2\$
limits the search to the *n* lines preceding the current line and, if the CP is not at the beginning of the current line, the current line up to the position of the CP.
- m,nCstring1\$string2\$
limits the search from the character *after* the *m*th character up to and *including* the *n*th character.

Numbers in this command refer to positions in the current buffer. You may also use numerical expressions containing arithmetic operators, and the values of variables and pseudovariables such as V0, ., and VP. For example:

60+V0,.+40Cstring1\$string2\$\$ limits the range of the search from the character after the character whose position is 60 plus the current value of Variable 0 to the 40th character following the current position of the CP.

VP,.Cstring1\$string2\$\$ limits the range of the search from the previous position of the CP to the current position of the CP.

The default value

for the C command, when you issue it without a numerical modifier, is .,Z, the range from the current position of the CP to the end of the buffer.

Symbolic Modifiers

This command takes both the colon (:) and commercial at (@) modifiers. You may use both in a single command.

@C%string1%string2%\$
allows you to define as a temporary delimiter the character immediately following the command name, in the example the percent sign (%). This allows you to include the standard delimiter, ESC (which echoes on the screen as \$), as a *part of* either of the string arguments. Choose as a temporary delimiter any convenient character which will not itself appear in the string arguments.

:Cstring1\$string2\$
inhibits an error message, returning a +1 if the search succeeds and a 0 if the search fails. You may use this form of the command as a numeric argument for the next command; see the colon symbol and the example below.

Inclusion of a search command in an iteration automatically simulates the : colon modifier.

You may define a temporary delimiter and inhibit error messages by combining the two modifiers in either order.

Characteristics

This command requires two delimiters, one following the first string argument, and one following the second. The standard delimiters are ESC, which echoes as \$, for the first argument, and either ESC or CTRL-D, which echoes as \$\$, for the second. You may define a different temporary delimiter as outlined in the "Symbolic Modifiers" section.

When the command has a negative numerical modifier, and there is more than one instance of *string1* in the range of the search, the command operates on the *earliest* instance of *string1* rather than on the instance closest to the CP.

To find the current position of the CP in terms of characters from the beginning of the buffer, issue a `. =` command. To find the current position of the CP in terms of characters to the end of the buffer, issue a `Z. =` command.

Both string arguments are optional. If you omit the first string argument but include the first delimiter, issuing the command in the form `C$string2$`, the command searches for an instance of the string that you searched for with the last previous search command (C, S, N, or Q). If it finds the string, it replaces it with your current *string2*.

If you omit the second string argument but include both delimiters, issuing the command in the form `Cstring1$$`, SPEED searches for the string and, if it finds the string, deletes it.

If you omit both string arguments but include both delimiters, issuing the command in the form `C$$`, SPEED will search for an instance of the *previous* string, as outlined above, and, if it finds the string, delete it.

In all cases, if the search for the first string argument is successful, SPEED replaces the string with the second string argument, and the CP follows the last replacing character.

The position of the CP following an unsuccessful search depends on the value assigned to position mode. (To find out your position mode, issue a `WP =` command; see the WP command for details.) If your position mode is set at 0, the default setting, the new position of the CP after an unsuccessful search will be

- *n* lines after its position before the search for `nC` commands and unchanged for `OC` commands
- at the position before the search for `-nC` commands
- after the *n*th character for `m,nC` commands
- at the beginning of the buffer for default `C` commands

If you set the position mode to a nonzero value, the new position of the CP after an unsuccessful search will be at the position where the search actually starts, that is,

- at the position before the search for `nC` commands
- *n* lines before the previous position for `-nC` commands and `OC` commands
- after the *m*th character for `m,nC` commands
- at the position before the search for default `C` commands

The value of search case match mode determines whether or not the `C` commands will match alphabetic characters regardless of case; see the `WS` command. In the default value of case match mode, 0, the command matches independently of case (*a* matches *A* in the first string argument), but insertions are case dependent. If you set the value at anything other than 0, the match in the first string argument is case independent (*a* will match *a* but not *A*).

C (continued)

Precautions and Error Messages

In the `m,nCstring1$string2$` variant of the command, the value of *m* must not be greater than the value of *n*. If it is, SPEED will display the message

Error: First argument greater than second argument

Recalculate and reissue the command, if you still wish to execute it.

If you inadvertently omit the third and final occurrence of the temporary delimiter, SPEED sends you the message, after you type CTRL-D,

Error: Unterminated string

SPEED carries out the search. If it is successful, SPEED replaces the first string argument with everything between the second instance of the temporary delimiter and the CTRL-D, *including any material you intended as a command line*, and ends with the CTRL-D echo, \$\$.

If you type the first argument and inadvertently type CTRL-D instead of ESC, SPEED will search for the string and, if it finds the string, delete it.

If you have executed no previous search, or the previous string exceeded 31 characters, SPEED will send you the message

Error: Incomplete string in search buffer

and continue the search, using the incomplete string (the first 31 characters of the previous string).

This command accepts templates and expansions, like the other search commands S, N, and Q. The C command takes templates in the usual fashion of a search command in the first string argument, the *search* component of the command. If you type in any other template in the second string argument, the command will treat it literally and insert the template character instead of that part of the string you are trying to match with it.

In search strings, SPEED flags any control character other than ↑B (expansion to a buffer), ↑F (expansion to a file), TAB (↑I), NEW LINE (↑J), vertical tab (↑K), form feed (↑L), CR (↑M), or ESC as an error unless you precede it with the control characters ↑G or ↑W (on models other than 6052, 6053, D100, or D200, ↑| or ↑_).

Remember to treat form feed, NEW LINE, carriage return, tab and space as single characters when using the C command.

Related Commands

To apply a single C command throughout a section of text, use it in conjunction with a Command Loop; see the `<command-line>` command.

To perform searches without changing text, see the S, N, and Q commands. To alter text at the position of the CP, see the D, K, I and \ commands.

Examples

*Pr*essing htis key has no affect, for esample,*

```
!Cht$th$C□a$□e$C□es$ex$$
```

*Pressing this key has no effect, for ex*ample,*

You spot several errors in the current line to the right of the CP. Your first command corrects a typo to *this*, your second command picks out the proper instance of *a* to change to *e*, and your third command closes up a space and spells *example* correctly.

**wiht Dick, who tried to to kiss Jane to.*

```
!Cht$th$C□to$$3MC$□too$$
```

with Dick, who tried to kiss Jane too.*

You make three changes in this line: you exchange the misplaced letters in *with*. The second C command finds the first instance of *to*. Since you intentionally omitted the second argument, it deletes the redundant word. You delete the extra space and move over the next instance of *to*. The third command uses the first argument of the previous C command to find *to* again, since you omitted its first argument, and replaces the string with the correct word.

**had less cavities. They ate less sweets and had regular*

```
!Cless$fewer$C$almost□no$$
```

had fewer cavities. They ate almost no sweets and had regular*

You correct *less* to *fewer* with the first command. You omit the first string argument in the second command to find the second instance of *less* and revise it to *almost* □ *no*.

all agents. Send 3 copies to all agents, and remind all agents that*

!.-25,.Call□agents\$each□agent\$\$

all agents. Send 3 copies to each agent, and remind all agents that*

You single out the one instance of *all agents* that you want to change by restricting the range of the search to the 25 characters before the CP with the double numerical modifier.

You spot an error in one of Walter's command lines, which reads

!End of File.FU\$

Walter intended to insert the phrase *End of File* and then update the file. The command line as it stands inserts the phrase and the letters *FU* since Walter omitted a necessary delimiter. When you try to edit Walter's line for him by entering

!C.\$\$\$\$

You merely succeed in deleting the period. Then you recall that you must set a temporary delimiter in order to treat the standard ESC delimiter as text. You issue the command with the commercial at modifier and select the percent sign as the temporary delimiter

!@C%.%.%T\$\$

*End of File.\$*FU\$*

This time you manage to help Walter out.

D

Command

Delete a number of characters.

Formats

nD
-nD

Function

This command deletes a specified number of characters from the position of the CP.

Numerical Arguments

This command takes positive and negative numerical arguments. It does not take double numerical arguments.

nD
deletes the *n* characters following the CP from the current buffer.

-nD
deletes the *n* characters preceding the CP from the current buffer.

The default value of the D command depends on the setting of the default argument mode. (To find out the setting, issue a WA= command; for details, see the WA command.) If the default argument value is set at 0, the D command without a numerical modifier has no effect. If you set the default argument value at +1, the D command deletes one character after the CP from the current buffer.

Symbolic Modifiers

None

Characteristics

If you issue an nD or -nD command when *n* exceeds the number of characters in the buffer in that direction, the command will delete all characters in that direction.

Precautions and Error Messages

Remember to count a form feed (␣), NEW LINE, carriage return, tab, or space as a single character.

You may be able to save some material that you inadvertently deleted. See the `_buffer-name` command (Save a Line).

D (continued)

Related Commands

To delete entire lines, see the K command. To delete material from the current buffer but store it in another buffer, see the BT command. To delete a noncurrent buffer, see the BK command.

To delete characters as you are typing them in, use the DEL or RUBOUT key.

To delete unexecuted commands on a single line, use CTRL-U, and to delete unexecuted commands on several contiguous lines, use the CTRL-C, CTRL-A sequence.

Example

There is no reasons to write up a longish description*

```
!-1D$long$3D$$
```

There is no reason to write up a long description*

You delete the character immediately preceding the CP, search forward for *long* and delete the suffix following it.

/D

Switch

Display text automatically

Formats

```
) X□SPEED/D□filename)
) X□SPEED/D)
```

Function

When you enter SPEED with the /D switch, SPEED tries to give you an automatic display of 10 lines on either side of the CP. SPEED executes your command line, gives you a display, and restores your prompt. Use this switch when you need a constant display and wish to avoid issuing T commands continuously.

Characteristics

When the CP is closer than 10 lines to the beginning or end of the buffer, SPEED displays the first or last 20 lines in the buffer.

Even if you enter SPEED with the /D switch, SPEED will not display text if the previous command line generates output from the terminal or if the previous command is an X command.

If you turn the Trace Toggle (?) on, SPEED inhibits the automatic display of text. When you turn the toggle off, SPEED restores the automatic display. See the ? command.

Precautions and Error Messages

The /D switch has no effect when you use it with the /I= (Invoke) switch. To get displays when using the /I= switch, place appropriate T, @T\$string\$ and ? commands in the command file.

Related Commands

If you wish to alter the display characteristics during the current editing session, see the WD command.

Example

```
) X□SPEED/D□lux)
```

You open SPEED with a filename and the /D switch. SPEED opens file LUX and displays the first 20 lines of that file.

E*Command***End: copy buffer and input file to output file.**

Format

E

Function

The **E** (End) command ends the editing of the current file by copying the contents of the buffer and the rest of the input file into the output file.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

This command clears the buffer. It does not close input or output files.

If you have both local and global files open in the current buffer, the command operates on the local files first.

Precautions and Error Messages

If you happen to issue the command a second time on the same input file, **SPEED** does not reduplicate the

input file in the output file. If you issue the command without an open input file, **SPEED** displays the message

Error: No open file

Related Commands

To bring material from the input file into the buffer, rather than send it to the output file, see the **A** command. To copy a page into the output file and bring another page into the buffer, see the **R** command. To discard the buffer and obtain more text from the input file, see the **Y** command. To copy the buffer to the output file without also copying the rest of the input file, see the **P** and **PW** commands.

Example

```
!FRdraft$FWrevise$A$$
```

```
.
```

```
    (editing commands)
```

```
.
```

```
!EFC$$
```

You open global input and output files, append a page or window to the buffer and carry out your editing tasks. The buffer contains the editing you have done, and you wish to leave the rest of the input file as it is. You issue the **E** command, which copies the buffer and the rest of the file **DRAFT** to the file **REVISE**. Then you close both files.

FB*Command*

File Backup: create a global backup file and close files.

Format

FB

Function

The FB (File Backup) command copies the current buffer and remainder of the input file, if any, to the output file. It retains a copy of the input file, clears the current buffer, and closes both files.

The effect of this command is to update your old global file with the changes you wish to make, retaining a copy of the original file as a backup if Update Mode is on. Using this command, you

- place the edited contents of the input file in the new output file
- close both files
- if Update Mode is on, rename the original input file as a backup file, `filename.BU`
- if Update Mode is on, rename the output file with the original name, `filename`, of the input file

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

The FB command works in several steps. When you open a file with the FO command, you automatically turn Update Mode on. SPEED then creates a temporary global output file with the same name as your input file and a .TM extension. Later, your FB command

- deletes the current file `filename.BU`, if there is one

- copies the current buffer and the remainder of the global input file, `filename`, into `filename.TM`
- clears the current buffer
- closes both `filename` and `filename.TM`
- renames `filename` to `filename.BU`
- renames `filename.TM` to `filename`

Update Mode then turns off, and you may issue new global file opening commands.

Precautions and Error Messages

You can execute the FB command successfully only if there is an output file available. The command closes a file opened with the FO (File Open) command, the FW (File Write) command, and the FNW (File New Write) command. It does *not* close a file opened with the FR (File Read) command, unless there is a corresponding output file, for instance, one opened with the FW command.

If you issue this command in a current buffer which has both local and global output files open, the command closes the *local file first* if it can, in spite of the fact that it is not the BFB command. The command also operates on local files if you have no open global files.

If you try to issue the FB command when you have no open files, SPEED displays the message

Error: No open file

and aborts the rest of the command line. Reissue the next command, if you still wish to execute it.

Related Commands

If you wish to reject the editing you have done and retain the old file without change, see the FC (File Close) command. If you do not wish to keep the old version of the file, see the FU (File Update) command.

Example

!F?\$B?\$\$

Global:

*Input File - :UDD:MIKE:FORMULA
Output File
:UDD:MIKE:FORMULA.TM
Update Mode On*

Local:

*Input File - None
Output File - None
Buffer 0 - 3405*

= >

!FB\$FOmatrix\$\$

User Mike checks the status of his files and buffers. He has a globally open file, and only the current buffer is active. He has finished his editing, but is uncertain of the results. He issues the **FB** command, closing and retaining both versions of the file. He turns to another editing task, opening the file **MATRIX** for updating.

FC

Command

File Close: close input and output files.

Format

FC

Function

Use this command to close input and output files.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

This command only closes files. It does not

- copy from the buffer to the output file
- clear the buffer
- rename input or output files
- create a backup file

Precautions and Error Messages

This command closes any file, no matter how you opened it. If you have turned Update Mode on by opening a file with an **FO** command, you have separate files **filename** and **filename.TM** when you exit from **SPEED**. The **.TM** file contains the results of your editing if you copy them from the buffer. (See the various output commands **E**, **P**, **PW**, **R** for transferring material from the buffer to an output file.)

The **FC** command has no effect if you have no files open.

The **FC** command has no effect on files opened locally.

FC (continued)

Related Commands

If you wish to copy and clear the buffer, see the FU (File Update) command. If you wish to create a backup file, see the FB (File Backup) command. If you wish to close current files and open new ones, see the FNR (File New Read) and FNW (File New Write) commands.

Example

```
!FRalpha$$#T$$  
!FC$$
```

You open the global input file ALPHA, read in a page or window to the buffer, display it, and discover that ALPHA is not the file you want. You close the file, and are now ready to open another global input file.

FNR	<i>Command</i>
File New Read: open a new file for input.	

Format

FNR [*filename*]

Function

This command closes the old global input file, if any, and opens a new global input file, *filename*, if you specify one. The file must already exist. Use this command when you wish to open, edit, and close a series of input files.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

This command requires a delimiter.

If you omit *filename* and Update Mode is off, the command merely closes the global input file, if there is one.

This command does *not* copy from the input file to the buffer (see the input commands A, R, and Y), copy the buffer to an output file, or clear the buffer.

Precautions and Error Messages

If the FNR command does not open a new file for you, you may have turned Update Mode on by issuing an FO command. You will receive the message

Error: Update mode on

and the rest of the command line will abort. Reissue the next command, if you still wish to execute it.

This command has no effect on files opened with an FW or FNW command.

The FNR command has no effect on files opened locally.

FNW *Command*
File New Write: create a new file for output.

The FU and FB commands will not close files opened with the FNR command unless you have opened a global output file. SPEED sends you the message

Error: No open file

and the rest of the command line aborts. Reissue the next command, if you still wish to execute it.

Example

```
!FRcase1$$  
.      (editing commands)  
!  
!FNRcase2$$  
.      (editing commands)  
!  
!FNRcase3$$  
.      (editing commands)  
!  
!FNR$$$
```

You open, use, and close a series of global input files. You close your last file by issuing the command without a filename argument, and exit from SPEED.

Format

FNW [*new.filename*]

Function

This command closes the old global output file, if any, and opens *new.filename*, if you specify it. You create the file with this command. The filename must not already exist. Use the FNW command when you wish to create, open, edit, and close a series of output files.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

This command requires a delimiter.

If you omit *new.filename* and Update Mode is off, this command merely closes the output file, if any.

This command does *not* copy the buffer to the output file, and does not clear the buffer. (See the output commands E, P, PW and R.)

Precautions and Error Messages

If you have turned Update Mode on by issuing an FO command, SPEED sends you the message

Error: Update mode on

and the rest of the command line aborts. Reissue the next command, if you still wish to execute it.

This command has no effect on global files opened with an FR or FNR command.

The FNW command has no effect on files opened locally.

FNW (continued)

Example

```
!FWproof1$$  
.  
    (editing commands)  
.  
!FNWproof2$$  
.  
    (editing commands)  
.  
!FNWproof3$$  
.  
    (editing commands)  
.  
!FNW$$H$$
```

You create, open, use, and close a series of global output files. You close your last file by issuing the command without a filename argument, and exit from SPEED.

FO

Command

File Open: open a file for updating.

Format

FOfilename\$

Function

Use this command when you wish to make changes in an already existing file, *filename*. The FO command

- opens *filename* for global input
- turns Update Mode on for *filename*
- deletes the current file *filename*.TM, if there is one
- creates a new global file, *filename*.TM, for output
- yanks the first page or window of *filename* into the buffer

You are now ready to carry out your editing tasks on *filename*. When you are through, you use either the FB (File Backup) command or FU (File Update) command to complete the editing cycle.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

This command requires a delimiter.

The FO command turns Update Mode on. When Update Mode is on, the file is opened *exclusively*. No one, *including yourself*, may have additional access to the file.

Precautions and Error Messages

If you attempt to open a file with the FO command, and the file has PERMANENCE ON, SPEED will display the message

Error: Attempt to edit a permanent file

and abort the command line.

If you attempt to open two files for updating in the same buffer with a **BFO** and an **FO** command, you lose text from one of the two files. Avoid using **FO** and **BFO** commands in the same buffer. If you want to open a second file for updating without closing the first, switch to an available buffer with the **BS** command and open the file locally in that buffer with a **BFO** command.

When Update Mode is on for a file, no one can have additional access to the file, including yourself if you try, for example, an **X TYPE filename** or **X QPRINT filename** command. You receive the message at the terminal (or from the line printer)

*WARNING: FILE IS EXCLUSIVELY OPENED,
CAN'T OPEN, FILE filename*

When you open a file with the **FO** command, you preserve the record type (data sensitive, dynamic, fixed-length, variable) of the input file and of the associated output file **filename.BU** or **filename.TM** (see the **FB** and **FU** commands). Opening for update also preserves file type, record length (if fixed), element size, and **UDA** (User Data Area).

You cannot issue other file-opening commands such as **FNW** or **FNR** when Update Mode is on. If you try to do so, **SPEED** displays the message

Error: Update mode on

If you try to open a second global file in the current buffer when Update Mode is on, you will receive the message

Error: File already open

whether you attempt the opening with **FO**, **FR**, or **FW**.

SPEED strips nulls on input. This means that you may create a file (containing nulls) and file it, but you cannot modify it thereafter (without losing the nulls).

Related Commands

If, at the close of the editing session, you are happy with the result, see the **FU** (File Update) command. If you like the result but would like also to retain the previous contents of the file, see the **FB** (File Backup) command. If you wish to discard the result of the editing session and retain the original file, see the **FC** (File Close) command.

Instead of updating a current file, you may want to create new files by rewriting or combining old ones. If so, see the **FR** (File Read) and **FNR** (File New Read) commands for opening input files, and the **FW** (File Write) and **FNW** (File New Write) commands for creating and opening output files.

Example

```
!FOmain$BSABFOvariant$F?$$$
```

Global:

```
Input File - :UDD:BOB:MAIN  
Output File - :UDD:BOB:MAIN.TM  
Update Mode On
```

Local:

```
Input File - :UDD:BOB:VARIANT  
Output File - :UDD:BOB:VARIANT.TM  
Update Mode On
```

User Bob opens exclusively the global file **MAIN**, switches to Buffer A, opens exclusively the local file **VARIANT**, and verifies what he has done by issuing the **F?** command to get the display that follows.

FR *Command*
File Read: open an existing file for input.

Format

FRfilename\$

Function

This command opens filename for global input. The file must already exist. Use this command to open a file especially when you plan to add to another file or create a new output file from its contents, rather than update the input file itself. Since Update Mode will be off, you will have to create a file for output, if you want one, with the FW or FNW command.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

This command requires a delimiter.

The FR (File Read) command only references the file to be read. It does not copy from the buffer to the output file, clear the buffer, or copy from the input file to the buffer. (See the input commands A, R, and Y.)

Precautions and Error Messages

The FU and FB commands do not close a file opened with the FR command unless you have opened a global output file. SPEED displays the message

Error: No open file

and aborts the rest of the command line. Reissue the next command, if you still wish to execute it.

Related Commands

To create a series of files for input, see the FNR (File New Read) command. To open a file for updating, see the FO (File Open) command.

Example

```
!FRmenu$A#TFWshoplist$$
```

You open the file MENU, append a page or window to the buffer, display it, and create an output file SHOPLIST whose contents will consist of MENU's edited pages. You do not alter MENU.

FU

Command

File Update: update and close files.

Format

FU

Function

This command in effect updates your input file with the changes you wish to make, and retains the original filename. This command renames a new global file for output with the same name as the old input file. It contains the material of the input file as you have edited it during the current sessions.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

The FU command carries out the updating in several steps. When you open a file with the FO (File Open) command, you automatically turn Update Mode on. SPEED then creates a temporary output file, filename.TM, with the same name as your input file and a .TM extension. When you subsequently issue the FU command, it

- copies the current buffer and the remainder of the input file, filename, into filename.TM
- clears the current buffer
- closes both filename and filename.TM
- deletes filename
- renames filename.TM to filename

Update Mode then turns off. You may issue new global file opening commands, if you wish.

Precautions and Error Messages

You can execute the FU command successfully only if there is an output file available. The command closes a file opened with the FO (File Open) command, the FW (File Write) command, or the FNW (File New Write) command.

The FU command closes an input file opened with the FR (File Read) command if there is a corresponding output file, opened with the FW (File Write) command.

If you issue this command in a current buffer that has both local and global output files open, the command closes the *local file first* if it can, in spite of the fact that it is not the BFU command. The command operates on local files if you have no global files open.

If you try to issue the FU command when you have no open output files, SPEED displays the message

Error: No open file

and aborts the rest of the command line. Reissue the next command, if you still wish to execute it.

Related Commands

If you wish to reject the editing you have done and retain the old file without change, see the FC (File Close) command. If you wish to keep both the old and new versions of the file, see the FB (File Backup) command.

Example

!F?\$B?\$

Global:

*Input File - :UDD:CHRIS:LAYOUT
Output File
:UDD:CHRIS:LAYOUT.TM
Update Mode On*

Local:

*Input File - None
Output File - None
Buffer 0 - 3601*

= >

!FUH\$

User Chris checks the status of her files and buffers. She has a globally open file. Only the current buffer is active. She updates her file, automatically clearing the buffer and closing both files, and exits from SPEED.

FW *Command*
File Write: create a file for global output.

Format

FWnew.filename\$

Function

This command creates and opens new.filename for global output. The file must not already exist.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

This command requires a delimiter.

Precautions and Error Messages

This file does *not* copy the buffer to the output file, and it does not clear the buffer. See the output commands E, P, PW, and R.

Update Mode for this command is off. If you wish to have a global input file, you must open one with an FR or FNR command.

You must close the current file before you can execute another FW command.

Related Commands

To update an existing file, rather than create a new output file, see the FO (File Open) command. To create and open global files in series, see the FNW (File New Write) command.

Example

```
!FRinquiries$A$$  
.  
    (editing commands)  
.  
!FWreply.form$EFC$$
```

You open the input file INQUIRIES, append a page or window to the buffer, edit in the buffer, create and open the file REPLY.FORM, copy the buffer to it, and close both files.

F? *Command*
Files? list files open in the current buffer.

Format

F?

Function

The F? (Files?) command tells you which files you have open in the current buffer and whether Update Mode is on.

This command lets you check on the status of your files, in case you need to do so, before you issue any commands opening or closing files, or exit from SPEED.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

If you have opened any files locally, you may wish to check each active buffer in turn, since F? gives the status of the *current buffer only*.

Precautions and Error Messages

If you issue a B? (Buffers?) and F? (Files?) command in the same command line, remember to follow the B? command with a delimiter, or to give the commands in the order F?B? SPEED takes B?F? as an inquiry about Buffer F followed by the ? Trace Toggle.

Related Commands

To find out which of your buffers are active, see the B? (Buffers?) command.

Examples

```
!F?$$  
  
Global:  
    Input File - :UDD:LIL:CHECK  
    Output File - :UDD:LIL:CHECK.TM  
    Update Mode On  
  
Local:  
    Input File - None  
    Output file - None
```

User Lil determines that she opened the file CHECK with Update Mode on; SPEED therefore created the temporary file with the extension .TM.

!BS1F?\$\$

Global:

*Input File - :UDD:LIL:CHECK
Output File - :UDD:LIL:CHECK.TM
Update Mode On*

Local:

*Input File - :UDD:LIL:BALANCE
Output File - SCALES*

User Lil switches to Buffer 1 and reissues the F? command. She sees that she had locally opened BALANCE for input and created SCALES for output.

H

Command

Halt: exit from SPEED.

Format

H

Function

The H (Halt) command ends the editing session and returns you to your parent process, usually the CLI. Use this command to make an orderly exit from SPEED.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

If the current buffer is not empty or you have files open in the current buffer, SPEED asks you

Confirm?

If you type y) for yes, the H command closes all files and terminates the session. If you type any other character, SPEED ignores the command.

If, when you are already in SPEED, you successfully issue an `XX[SPEED` or `XX[SPEED[filename$` command, you create a SPEED son process. An H command in the second, offspring SPEED process will return you to the parent SPEED process rather than to the CLI.

Similarly, if you enter SPEED with multiple filenames, using a CLI command like `X[SPEED[(file.1[file.2)],` an H command will open files successively for updating rather than return you to the parent process.

Precautions and Error Messages

This command does not send the query for *noncurrent* buffers that are not empty. *It does not save any buffers for subsequent editing sessions*, so make certain that you have copied all buffer contents that you wish to save to the appropriate output files.

H (continued)

This command does not send the query for local files open in noncurrent buffers. It will close any such files, but will not copy the relevant buffer to an output file. If, for example, you opened a local file in a noncurrent buffer for updating with the BFO command, and issue the H command in a different buffer, SPEED closes both the input file filename and the output file filename.TM. The input file will contain none of your editing from this session, and the output file will be empty.

Related Commands

You may wish to check on the status of your buffers and files before issuing this command; see the F? and B? commands.

To abort a SPEED session, use the CTRL-C, CTRL-B sequence. See that console control sequence in this dictionary.

Example

```
!FCH$$
```

Confirm?

```
y!
```

```
) _ (system response)
```

You close files without clearing the buffer and issue the exit command. SPEED queries you but you are willing to discard the buffer contents. You reply yes to SPEED's query and return to the parent CLI, as the new prompt shows.

I

Command

Insert text at the current CP position.

Formats

lstring\$
nl

Function

This command inserts into the current buffer the text string that you type at the keyboard. It also allows you to type in a character using its ASCII decimal equivalent.

Numerical Arguments

None

Symbolic Modifiers

This command accepts the commercial at modifier (@). This allows you to define as a temporary delimiter the character immediately following the command name. This permits you to insert as part of the text the standard delimiter ESC.

Choose for the temporary delimiter any convenient character that will not appear as part of the text you want to insert. For example, you might issue the command @!%C#\$No.\$% with % as the temporary delimiter. (The I command inserts a C command in your current buffer.) The temporary delimiter is in effect only for the current command.

The nl version of this command accepts the ampersand modifier (&) preceding the numeral. If the alternate radix is set at the default value (octal), and you issue a &102!\$\$ command, the command will type in B rather than f, since 102_8 is 66_{10} and the ASCII decimal equivalent of B is 66. See the WR command and the ampersand symbol for specification and use of an alternate radix.

Characteristics

The lstring\$ form of this command requires a delimiter. The standard delimiters are ESC, which echoes as \$, and CTRL-D, which echoes as \$\$\$. To define a temporary delimiter, see the preceding section. The nl form of this command does not require a delimiter.

The string in the lstring\$ form of the command may be longer than a line. That is, it may contain NEW LINES and carriage returns, since these do not act as delimiters in SPEED.

The command repositions the CP at the end of the inserted text.

The nl form of the command allows you to insert ASCII characters that are difficult to insert from the keyboard. For example, if you wish to type on a single line a command line involving form feeds, line feeds, and tabs, you can type

```
!|)  
→Page□2|  
|  
$$
```

on a single line with the command string

```
!10I$9I$IPage□2$10I$10I$$
```

Normally, however, you save keystrokes (and looking up ASCII values) by typing in the character directly. Use the nl command when it is not possible to key the character; for instance, issue 4I to type in ↑D.

Precautions and Error Messages

If you inadvertently omit the second instance of the temporary delimiter before you type in a CTRL-D, SPEED displays the message

Error: Unterminated string

SPEED inserts your text, *including any commands you intended to issue after the insertion*, and echoes CTRL-D as \$\$.

For the ASCII decimal equivalents of characters, see Appendix A. Although the nl command allows you to type control keys into your current buffer (and output file) and echo them on the keyboard, the line printer will not print them out. If you wish listings showing control characters, follow the procedure outlined in "A Note about Line Printer Listings" in the introduction to this chapter.

If you are inserting both ASCII decimal equivalents and other text, you must reissue the lstring\$ command after the nl command; if you fail to do so, and type 10IDear Sir: instead SPEED will try to interpret everything after the l as a command line. Issue instead the command line 10IIDear Sir:.

It is easy, especially for an experienced typist, to forget the l command name and simply start typing in text. When you do this, SPEED tries to interpret your keyboard input as a command line. This can be harmful to the contents of your current buffer, especially if you inadvertently issue a Y or H command because your text contains either letter. Check your command line to make sure you began it with the l command name before keying in CTRL-D.

To abort a single-line insertion before typing in a CTRL-D, use CTRL-U or hold down DEL and REPT at the same time. To abort a multiple-line insertion, use the CTRL-C, CTRL-A sequence.

Related Commands

You may use this command with the templates ↑Bbuffer-name and ↑Ffilename\$ to insert into the current buffer material from other active buffers or files to which you have access. See the entries for those templates in this dictionary.

To insert the values of numerical expressions, pseudo-variables, and variables, use the Backslash; see the n\ command.

The l command inserts text into the current buffer as you type it from the keyboard, and has no direct effect on the contents of your open files. To copy the buffer, including your insertions, to an output file, see E, P, and PW output commands, and the BFU, FU, BFB, and FB file-closing commands.

I (continued)

Examples

You are writing a letter and are checking for typos.

you may available at you desk.*

```
!lr$OSmay$I□have$ZJI□□In|
conclusion,□$$
```

*you may have available at your desk. In conclusion, **

You insert r, correcting you to its possessive form. You spot an omission earlier in the line, restrict your search for the place to insert to the current line, and insert a space and the missing verb. You then jump to the end of the buffer and continue composing.

You want to check the numbers in your text. You type

```
!BSN$@I%S1234567890$%$BS0
```

SPEED switches you to Buffer N. By modifying the I command with the commercial at (@), you were able to insert

```
S1234567890$
```

including the delimiter (\$) in that buffer. Now, whenever you want to search for the next number in Buffer 0, you will merely issue the command]BN instead of the long string above. The delimiter ensures that you will not include extraneous material in your search.

/I=

Switch

Invoke a commandfile.

Format

) X□SPEED /I=command.file□text.file|

Function

When you enter SPEED with the /I= switch, SPEED takes its commands from the file you mention in the /I switch, command.file, and executes them on the file argument of the entry command, text.file. This switch allows you to execute entire SPEED editing programs from the parent process.

Give your SPEED command files names with a unique extension, perhaps .SCF (for SPEED Command File), so that you can tell quickly which files you may invoke with the /I= switch.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

When you use the /I switch, SPEED terminates itself when it reaches the end of file, and restores the prompt of the parent process.

SPEED does not give you a display when you enter it with the /I switch, unless your command file contains display commands.

The CLI permits you to enter a series of command files if you wish

```
) X SPEED /I=(file.1□file.2□file.3).scf□text.file|
```

Precautions and Error Messages

When you enter SPEED with this switch and a filename argument, Upade Mode is on. For the editing in the command file to take effect on the text file, the command file must contain an updating command, such as FU. Otherwise, SPEED will close the text file without making editing changes.

Related Commands

You may also execute a command file from within SPEED. See the CTRL-F template in this dictionary.

J **Jump the CP to the beginning of a specified line.**

Example

Your file PROOF contains

*If x wants a motorcycle, the market will fall.
Bob wants a motorcycle.
Therefore, the market will fall.*

) X□SPEED /I=line.scf□proof)

1. If x wants a motorcycle, the market will fall.
2. Bob wants a motorcycle.
3. Therefore, the market will fall.

You previously created a file of SPEED commands, LINE.SCF, which numbers the lines of text in other files. Now you execute it on the file PROOF from your parent process.

Formats

J
nJ

Move the CP from its current position to the beginning of a line specified by its line number.

Numerical Modifier

This command takes only nonnegative numerical modifiers.

J, 0J and 1J

are equivalent and move the CP to the beginning of the buffer, regardless of the default argument value; see WA.

nJ

moves the CP to the beginning of the n th line from the beginning of the buffer.

ZJ

moves the CP to the very end of the buffer, past the last character of the last line.

The default value

for the J command is 0, which represents the beginning of the buffer.

Symbolic Modifiers

None

Characteristics

This command repositions the CP relative to the beginning of the buffer, *not* relative to the current position of the CP. To use the command effectively, you must know the number of the line you want the CP on. To find out which line the CP is currently on, issue a VL= command.

This command counts as a line a string terminated by a NEW LINE character. It does *not* count as two lines a string containing a carriage return (CR) character.

If you issue the command with n greater than the number of lines you actually have on the buffer, the CP will reposition at the very end of the buffer.

J (continued)

You may use numerical expressions other than simple numbers with the J command. For example, VN/2J will place the CP at the beginning of the line nearest the middle of the buffer.

The J command takes arguments in the range 0 through +65535₁₀.

Precautions and Error Messages

If you inadvertently issue the command with a negative number, the CP will reposition at the end of the buffer. If you inadvertently issue the command with a double numerical argument, the CP will not move and SPEED will send you an error message

Error: Illegal number of arguments to command

Related Commands

To move the CP backward or forward relative to its current position, see the L (Line) command. To reposition the CP from one character to another, see the M (Move) command. To place the CP after a particular string, see the S (Search) command.

Example

```
.
.           (text)
.
with the utmost care and precision.
.
.           (more text)
.
Yours sincerely,*
```

```
!10J$Cht$th$ZJ$$
```

You spot a typo on the tenth line in the buffer, jump the CP to the beginning of that line, make the correction, and jump back to the end of the buffer so that you can continue composing your letter.

K

Command

Kill lines of text in the buffer.

Formats

K
nK
-nK
m,nK
#K
ZK
-ZK

The K (Kill) command deletes, or *kills*, lines and parts of lines in the buffer relative to the position of the CP.

Numerical Modifiers

This command takes positive, 0, negative, and paired numerical arguments.

OK

deletes text from the beginning of the current line up to the current CP position.

1K

deletes text from the CP to the end of the line, including the NEW LINE character.

nK

deletes text from the current CP position up to the n^{th} NEW LINE. That is, it deletes the next n lines of text, but will delete the first line only from the CP forward.

ZK

deletes text from the current CP position to the end of the buffer.

-1K

deletes text from the previous line and the current line up to the CP.

-nK

deletes text from the beginning of the n^{th} line preceding the current line through n NEW LINES up to the position of the CP. That is, it deletes text from the previous n lines and, if the CP is not at the beginning of the current line, text on the current line up to the position of the CP.

m,nK

deletes text from the character *after* the m^{th} character up to and *including* the n^{th} character. The value of m must be less than the value of n . Numbers in this command refer to positions in the current buffer. You may also use numerical expressions containing arithmetic operators, and the values of variables and pseudo-variables such as V0, ., and VN:

For example, 67,100K deletes from the 68th character in the current buffer to the 100th.

For another example, 67,80+VOK deletes from the 68th character in the buffer to the character whose position from the beginning of the buffer is 80 plus the current value of Variable 0.

0,.K or -ZK

deletes from the beginning of the buffer up to the position of the CP.

The command #K

abbreviates 0,ZK, which deletes the entire current buffer.

The default value

for this command (issuing the command without a numerical argument as K) depends upon the default argument value; see the WA command.

When the default argument mode is set at 0, the K command acts like the OK command, deleting all characters from the beginning of the line up to the CP.

If you set the default argument mode at +1, the K command acts like the 1K command, deleting all characters from the CP to the end of the line (including the NEW LINE character).

Symbolic Modifiers

None

Characteristics

If you specify a numerical argument which exceeds the limits of the buffer, SPEED deletes only the material between the CP and the limit of the buffer. It does *not* reposition the CP to delete the full amount of text specified. For example, if you issue a -10K command when the CP is on line 7, SPEED kills only the preceding seven lines.

The default value for this command depends upon the value of the default argument mode. See the WA command and the "Numerical Arguments" section of this command.

Precautions and Error Messages

In the m,nK form of the command, m must not be greater than n . If it is, SPEED displays the message

Error: First argument greater than second argument

Recalculate and reissue the command, if you still wish to execute it.

Related Commands

To delete a specific number of characters at the position of the CP, see the D command. To delete a specific string, see the Cstring1\$\$ form in the "Characteristics" section of the C command entry. To delete noncurrent buffers, see the BK command. To delete lines from the current buffer but store them in another buffer, see the BT command.

Example

!-2K\$J\$0,10K\$\$

You delete the preceding two lines, jump to the beginning of the buffer and delete the first 10 characters.

L

Move the CP from Line to line.

Formats

L
nL
-nL

The L (Line) command repositions the CP relative to the current position of the CP, placing it at the beginning of an earlier or later line.

Numerical Arguments

This command takes positive and negative numerical arguments, but not paired arguments.

- OL
moves the CP to the beginning of the current line.
- 1L
moves the CP across one NEW LINE, that is, to the beginning of the next line.
- nL
moves the CP forward across n NEW LINE characters, that is, to the beginning of the n^{th} line forward from the CP.
- ZL
moves the CP to the very end of the buffer, to the *end* of the last line.
- 1L
moves the CP to the beginning of the previous line.
- nL
moves the CP backward to the beginning of the n^{th} line preceding the current line.

The default value of the L command depends upon the setting of the default argument value; see the WA command.

If the value is set at 0, the L command acts like the OL command, moving the CP to the beginning of the current line.

If you set the value to +1, the L command acts like the 1L command, moving the CP to the beginning of the next line. To find your default argument value, issue a WA= command.

Symbolic Modifiers

None

Characteristics

If the numerical argument exceeds the limits of the buffer, the CP repositions at the limit. For example, if the CP is on line 15 and you inadvertently specify a -100L command, the CP repositions at the beginning of the buffer.

Precautions and Error Messages

Instead of issuing sequences of commands such as -nL\$C or -nL\$\$, simplify your command lines by issuing -nC and -nS commands.

Related Commands

To move to a particular line in the text regardless of the current position of the CP, see the J command. To move the CP within the current line or from character to character, see the M command.

Example

```
1980  
1981  
1983  
1984*
```

```
!-1L$11982$ZJ$$
```

You spot an omitted line, move to the beginning of the line it should come before, insert the line (and NEW LINE) and jump back to the end of the buffer to continue entering text.

M*Command*

Move the character pointer across characters.

Formats

M
nM
-nM

This command repositions the CP backward and to the left, or forward and to the right, relative to its current position.

Numerical Arguments

This command takes positive and negative numerical arguments, but not paired arguments.

nM
moves the CP from its current location *n* characters to the right, or forward.

ZM
moves the CP to the end of the current buffer.

-nM
moves the CP from its current location *n* characters to the left, or backward.

-ZM and -.M
move the CP to the beginning of the current buffer.

The default value of **M** depends on the setting of the default argument value; see the **WA** command. To find your default argument value, issue a **WA=** command.

If the default argument value is set at 0, the **M** command has no effect.

If you set the default argument value at +1, the **M** command moves the CP one character forward.

Symbolic Modifiers

None

Characteristics

If the numerical argument exceeds the limits of the buffer, the CP repositions at that limit of the buffer. For example, if there are 70 characters in the buffer, and you inadvertently issue a **500M** command, the CP repositions at the end of the buffer.

To use this command effectively, you may need to know your exact position in the buffer. To find out how far (how many characters) the CP is from the beginning of the current buffer, issue a **.=** command. To find out how far the CP is from the end of the current buffer, issue a **Z.=** command. To find out how far the CP is from the beginning of the current line, issue a **VM=** command.

Related Commands

To move from line to line, rather than from character to character, see the **J** and **L** commands. To search for a particular string of text and reposition the CP relative to it, see the **S** command and the related **N** and **Q** commands.

Example

You are proofreading some instructions. You read

If you try issue the command*

!-5M\$!to□\$\$

*If you try to *issue the command*

You have omitted a word to the left of the CP. You move 5 characters back and insert the missing material.

N

Command

Conduct a Nonstop search.

Format

N [*string*] \$

Function

This command searches from the current position of the CP to the end of the buffer for *string*. If SPEED does not find the string in the current buffer, it executes an R command, copying the contents of the buffer to the output file, if you have one open. It then clears the buffer, reads the next page or window of the input file into the buffer, and continues the search. SPEED works through the input file in this manner until it locates the string or reaches the end of the file (at which point the buffer will be empty).

Numerical Arguments

None

Symbolic Modifiers

This command takes both the colon (:), and commercial at (@) modifiers. You may use both in a single command.

@N%string%\$

allows you to define as a temporary delimiter the character immediately following the command name, in this case the percent sign (%). This permits you to include the standard delimiter, ESC (which echoes on the screen as \$), as a *part of* the string argument, rather than as a delimiter of it. Choose as a temporary delimiter any convenient character which does not itself appear in the string argument.

:Nstring\$

inhibits an error message, returning a +1 if the search succeeds and a 0 if the search fails. You may use this form of the command as a numerical argument for the next command; see the colon symbol.

You may define a temporary delimiter and inhibit error messages by combining the two modifiers in either order.

Characteristics

This command requires a delimiter. The standard delimiters are ESC, which echoes as \$, and CTRL-D, which echoes as \$\$\$. For a different, temporary delimiter, see the commercial at modifier (@) in the section above.

If you omit the string argument, the N\$ command searches for the same string that the immediately preceding search command (C, N, Q, or S) searched for.

If the nonstop search succeeds, the CP follows the string it finds.

This command accepts control key templates in the search string. For their behavior, consult the individual entry for each template in this dictionary.

The value of the search case match mode determines whether or not this command matches characters independently of case; see the WS command. If the mode is set at its default value, 0, the N command is case independent (*a* matches *A*). If you set the value at anything other than 0, the match is case dependent (*a* matches only *a*, and not *A*). To find the value of the mode, issue a WS= command.

Precautions and Error Messages

If you inadvertently omit the second occurrence of the temporary delimiter, SPEED displays the message, after you type in CTRL-D,

Error: Unterminated string

SPEED carries out the search, and if it is successful, replaces the string argument with everything between the temporary delimiter and the CTRL-D, *including any material you intended as a command line*, and ends with the CTRL-D echo, \$\$.

If you issue an N\$ command, without a search string, and there was no previous search, or the previous string exceeds 31 characters, SPEED sends you the message

Error: Incomplete string in search buffer

and continues the search using the incomplete search string (the first 31 characters of the previous string.)

If the search fails, SPEED reads the last page or window of the input file into the output file, if there is one, and sends you the message

Error: Unsuccessful search

At that point, your current buffer will be empty. Your output file will still be open.

If you have no output file open, SPEED detects that at the first R execution, displays the message

Error: No open file

and retains the current page or window in the buffer.

In search strings, SPEED flags any control character other than **↑B** (expansion to a buffer), **↑F** (expansion to a file), **TAB (↑I)**, **NEW LINE (↑J)**, **vertical tab (↑K)**, **form feed (↑L)**, **CR (↑M)**, or **ESC** as an error unless you precede it with the control characters **↑G** or **↑W** (on models other than 6052, 6053, D100, or D200, **↑↑** or **↑_**).

Related Commands

If you wish to discard the searched pages or windows instead of copying them to an output file, see the **Q** (Quick) command. If you wish to confine your search to the page or window currently in the buffer, see the **S** command.

Example

Proust has moved, and you wish to correct his address in your paged file ADDR.

```
) X SPEED addr|
INProust, □M.$L5T$$
```

*Proust, M.
99 Memory Lane
Souvenir City*

You open your ADDR file, which you have organized in pages, skim past the preceding pages directly to his name, and get a display of the entry. You are ready to make the correction. The pages you have flipped past are in your output file.

O

Command

Pass command control Over to a label.

Formats

```
Olabel$...!label!  
!label!...Olabel$
```

Function

You may select for a label any string you wish, but it must not contain an exclamation point or a delimiter. The string within the exclamation points and the string following the **O** command name must be identical for the command to function properly.

SPEED ignores a **!label!** until, during command execution, it encounters an **Olabel\$** command. SPEED then halts command execution and searches in the command line for a preceding or following occurrence of the same label between exclamation points. If it finds one, it resumes command execution with the text immediately following the first occurrence of the label.

You may combine this command in a command line with a Conditional Execution or a Command Loop in order to reissue automatically a sequence of commands an appropriate number of times under appropriate conditions, as you will see in the "Characteristics" section of this entry. If you have not already done so, please consult the **n'Xcommand-string'** command (Conditional Execution) and the **<command-string>** command (Command Loop) for further details of the recursive application of SPEED commands.

If you wish to write comments into a command line, you may do so by setting them off with pairs of exclamation points, since SPEED ignores a **!label!** whose contents do not occur in an **O** command.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

The **O** command requires a standard delimiter. You cannot define a temporary delimiter for this command.

Failure to find a label does not affect the position of the CP.

O (continued)

When the label *precedes* the O command, SPEED executes the commands following the label until it encounters the O command with a matching label for the first time. SPEED then returns control to the first command following the label.

When the label *follows* the O command, SPEED skips over the commands between the O command and the label, and passes control to the first command following the label.

When you place more than one O command and its associated label in a command line, the particular configuration of labels and O commands determines which parts of the command line SPEED skips or repeats. Figure 7-1 shows how SPEED might insert a string with characters in the reverse order.

If you set more than one instance of the same label for a O command, control passes to the earliest instance of the label; later instances are ignored.

You may issue more than one O command using the same label. The earliest of the O commands will execute first, and subsequent O commands will return you to the label.

Precautions and Error Messages

If you enter a command line containing a label, a subsequent corresponding O command, and no Command Loop or Conditional Execution, you run the risk of creating an endless loop. If you issue this command with no provisions for terminating any loop you may create, you may have to issue a CTRL-C, CTRL-A sequence to restore your prompt, or wait for SPEED to send you the message

Error: Insufficient memory available

If you issue this command with no preceding label in the command line, SPEED displays the message

Error: Label not found

If you inadvertently issue the command without its label argument, or include the ESC delimiter in the label, SPEED displays the same message.

You must separate adjacent labels with a delimiter. If you do not, SPEED will send you the message

Error: Label not found

If the O command is within the Command Loop, and the !label! outside, SPEED executes the commands in the Loop until it reads the O command, and then jumps out to the label. It is illegal to branch into a Command Loop. If the !label! is within the Command Loop, and the O command outside, SPEED displays the message

Error: Illegal command

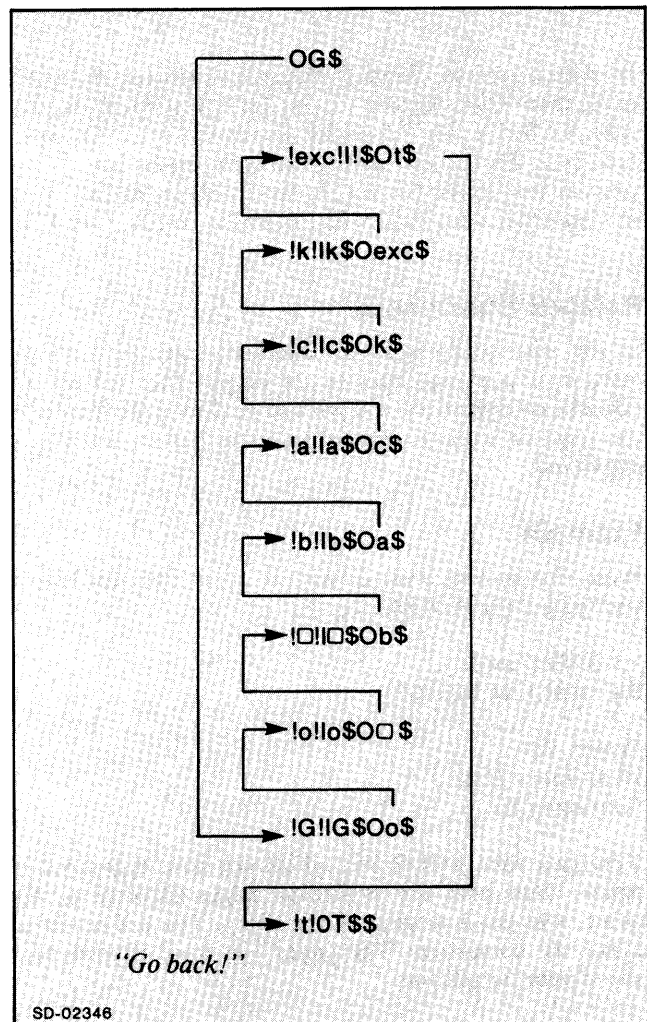


Figure 7-1. Command Control

Related Commands

Consult the entry for Conditional Execution in this dictionary for a description of its interaction with the O (Over) command.

Example

You have to set up some rating sheets for a psychological experiment. Rather than doing it laboriously, one character at a time, you type

```
!5VS1$!resp!V1\ $4 <I-$> $VD1''G$Oresp$I0$$
```

You set Variable 1 to 5, and set the label `resp`. You insert the current value of the variable, 5, insert four hyphens, and decrement Variable 1 by 1 to 4. If the variable is greater than 0 (and it is), the O command will return you to the label. The second time, the value inserted is 4. You keep returning to the label as long as the value of Variable 1 is positive. When it reaches 0 by your VD command, control passes to the command following the O command, which inserts a 0. SPEED responds, creating

```
5----4----3----2----1----0*
```

P

Command

Put buffer text into the output file and append a form feed.

Formats

P
nP
-nP
m,nP

Function

The P (Put) command puts the material you specify in an open output file. It does not close the file, and it does not kill the material in the current buffer unless you use the colon modifier (:). The P command places a form feed after the copied material in the file.

Numerical Arguments

The P command takes positive, negative, and double numerical arguments.

OP

copies text in the buffer to the output file from the beginning of the line to the CP.

1P

copies text in the buffer to the output file from the CP to the end of the line.

nP, except for OP,

copies text in the current buffer to the output file text from the current CP position up to the n^{th} NEW LINE. That is, it copies the next n lines of text, but the first line copied will contain text only from the CP forward.

-1P

copies text in the buffer to the output file from the previous line and the current line up to the CP.

-nP

copies text in the buffer to the output file from the beginning of the n^{th} line preceding the current line through n NEW LINES up to the position of the CP. That is, it copies text in the buffer to the output file from the previous n lines and, if the CP is not at the beginning of the current line, text on the current line up to the position of the CP.

P (continued)

*m,n*P

copies text in the buffer to the output file from the character *after* the *m*th character up to and *including* the *n*th character. Numbers in this command refer to positions in the current buffer. You may also use numerical expressions containing arithmetic operators, and the values of variables and pseudo-variables such as V0, ., and VN. For example,

23,53P copies from the 24th character in the current buffer to the 53rd.

31,40+VOP copies from the 32nd character in the buffer to the character whose position from the beginning of the buffer is 40 plus the current value of Variable 0.

0,P copies from the beginning of the buffer up to the position of the CP, and .,ZP copies from the position of the CP to the end of the buffer.

#P

abbreviates 0,ZP, which copies the entire current buffer to buffer *x*.

The default value

for this command (issuing the command without a numerical argument as P) is 0,Z, which represents the entire buffer.

Symbolic Modifier

This command takes the colon modifier (:). The :P command deletes from the current buffer the characters it copies to the output file.

Characteristics

The unmodified command does not change the position of the CP.

The CP repositions at the deletion site when you modify the command with the colon.

Precautions and Error Messages

The value of *m* must not be greater than the value of *n*. If it is, SPEED displays the message

Error: First argument greater than second argument

Recalculate and reissue the command, if you still wish to execute it.

If you try to issue this command with no open output file, SPEED displays the message

Error: No open file

and retains the buffer contents.

Related Commands

This command automatically appends a form feed at the end of the insertion in the output file; to append without a form feed, see the PW command. To copy the buffer, clear it, and read in another page or window, see the R command. To copy the buffer and the rest of the contents of the input file, see the E command. To discard rather than copy material in the buffer, see the Y or K command.

Example

You are transferring the contents of an open global input file to various output files.

```
!BFWseries.3$.Z:PBFCFU$$
```

You create and open a local output file, SERIES.3. You copy to it everything in the buffer from the CP forward, automatically appending a form feed, and close both local and global files. Since you issued the command with the colon modifier, the material you copied has been deleted from the buffer. In effect, you have split your original file in two.

PW*Command***Put buffer text into the output file
Without a form feed.**

Formats

PW
nPW
-nPW
m,nPW

Function

This command copies the material you specify to an open output file. It does not close the file, and it does not kill the material in the current buffer unless you use the colon modifier (:). The PW command does not place a form feed after the copied material in the file.

Numerical Arguments

The PW command takes positive, negative and paired numerical arguments.

OPW

copies text in the buffer to the output file from the beginning of the line to the CP.

1PW

copies text in the buffer to the output file from the CP to the end of the line.

nPW, except for OPW,

copies text in the current buffer to the output file text from the current CP position up to the n^{th} NEW LINE. That is, it copies the next n lines of text, but the first line copied will contain text only from the CP forward.

-1PW

copies text in the buffer to the output file from the previous line and the current line up to the CP.

-nPW

copies text in the buffer to the output file from the beginning of the n^{th} line preceding the current line through n NEW LINES up to the position of the CP. That is, it copies text in the buffer to the output file from the previous n lines and, if the CP is not at the beginning of the current line, text on the current line up to the position of the CP.

m,nPW

copies text in the buffer to the output file from the character *after* the m^{th} character up to and *including* the n^{th} character. Numbers in this command refer to positions in the current buffer. You may also use numerical expressions containing arithmetic operators, and the values of variables and pseudo-variables such as V0, ., and VN. For example,

23,53PW copies from the 24th character in the current buffer to the 53rd.

31,40+VOPW copies from the 32nd character in the buffer to the character whose position from the beginning of the buffer is 40 plus the current value of Variable 0.

0,.PW copies from the beginning of the buffer up to the position of the CP, and .,ZPW copies from the position of the CP to the end of the buffer.

#PW

abbreviates 0,ZPW, which copies the entire current buffer to buffer x .

The default value

for this command (issuing the command without a numerical argument as PW) is 0,Z, which represents the entire buffer.

Symbolic Modifier

This command takes the colon modifier (:). The :PW command deletes from the current buffer the characters it copies to the output file.

Characteristics

The unmodified command does not change the position of the CP.

The CP repositions at the deletion site when you modify the command with the colon.

PW (continued)

Precautions and Error Messages

For the `m,nPW` command, the value of `m` must not be greater than the value of `n`. If it is, SPEED displays the message

Error: First argument greater than second argument

Recalculate and reissue the command, if you still wish to execute it.

If you try to issue this command with no open output file, SPEED displays the message

Error: No open file

and retains the buffer contents.

Related Commands

This command does not append a form feed at the end of the insertion in the output file; to append a form feed, see the `P` command. To copy the buffer, clear it, and read in another page or window, see the `R` command. To copy the buffer and the rest of the contents of the input file, see the `E` command. To discard rather than copy material in the buffer, see the `Y` or `K` command.

Example

You are creating new files from a globally open input file. You do not wish to alter the input file.

```
!BFWsuppl.4$0,.PWBFCFU$$
```

You create and open a local output file, `SUPPL.4`. You copy to it everything in the buffer from the beginning of the buffer to the `CP`, and close both local and global files. Since you did not issue the command with the colon modifier, the material you copied remains in the buffer for the subsequent update command. In effect, you have replicated the first part of the original file in a new file.

Q

Command

Conduct a Quick search.

Format

`Q [string] $`

Function

The `Q` (Quick) search command searches from the current position of the `CP` to the end of the buffer for **string**. If SPEED does not find the string in the current buffer, it executes a `Y` command, clearing the buffer. It does *not* copy the searched text to an output file, but discards it instead. It then reads the next page or window of the input file into the buffer, and continues the search. SPEED will work through the input file in this manner until it locates the string or reaches the end of the file (leaving an empty buffer).

Numerical Arguments

None

Symbolic modifiers

Inclusion of a search command in an iteration automatically simulates the `:` colon modifier.

This command takes both the colon (`:`) and commercial at (`@`) modifiers. You may use both in a single command.

`@Q%string%$`

allows you to define as a temporary delimiter the character immediately following the command name, in this case the percent sign (`%`). This permits you to include the standard delimiter, `ESC` (which echoes on the screen as `$`), as a *part of* the string argument, rather than as a delimiter of it. Choose as a temporary delimiter any convenient character which does not itself appear in the string argument.

`:Qstring$`

inhibits an error message, returning a `+1` if the search succeeds and a `0` if the search fails. You may use this form of the command as a numerical argument for the next command; see the colon entry in this dictionary.

You may define a temporary delimiter and inhibit error messages by combining the two modifiers in either order.

Inclusion in an iteration automatically simulates the colon modifier.

Characteristics

This command requires a delimiter. The standard delimiters are ESC, which echoes as \$, and CTRL-D, which echoes as \$\$. For a different, temporary delimiter, see the commercial at modifier (@) in the section above.

If you issue a Q command when global or local update modes are on and the command is not in a command loop, SPEED asks you

Confirm (Q-command)?

If you intended to use the Q command, type Y\$\$ for yes. If you issued the command inadvertently, type another character. SPEED will ignore the command. (If you type another command in place of the yes response, SPEED will ignore that too; reissue the command if you still wish to execute it.)

If you omit the string argument, the Q\$ command will search for the string which the immediately preceding search command (C, N, Q, or S) searched for.

This command accepts control key templates in the search string. For the behavior of each template, see their individual entries in this dictionary.

The value of the search case match mode determines whether or not this command matches characters independently of case; see the WS command. If the mode is set at its default value, 0, the Q command is case independent (*a* matches *A*). If you set the value at anything other than 0, the match is case dependent (*a* matches only *a*, and not *A*). To find the value of the mode, issue a WS= command.

Precautions and Error Messages

If Update Mode is off, that is, you opened files with local or global FR and FW commands, or if the Q command is in a Command Loop, the command will clear the buffer *without the query*.

If there has been no previous search when you issue a Q\$ command without a string argument, or the previous string exceeded 31 characters, SPEED sends you the message

Error: Incomplete string in search buffer

and continues the search using the incomplete search string (the first 31 characters of the previous string).

If the Quick search fails, SPEED displays the message

Error: Unsuccessful search

At that point, your current buffer will be empty; your output file will be open but will contain none of the searched text.

In search strings, SPEED flags any control character other than ↑B (expansion to a buffer), ↑F (expansion to a file), TAB (↑I), NEW LINE (↑J), vertical tab (↑K), form feed (↑L), CR (↑M), or ESC as an error unless you precede it with the control characters ↑G or ↑W (on models other than 6052, 6053, D100, or D200, ↑↑ or ↑...)

Related Commands

To retain the searched material in the output file instead of deleting it from the buffer, see the N command. To confine your search to the window currently in the buffer, see the S command.

Example

```
) X SPEED notes!  
!Qdecisions$$  
Confirm (Q-command?) y!  
!
```

You open for updating the file NOTES. You intend to discard from the file all pages preceding the first occurrence of **decisions**. Since Update Mode is on, SPEED gets a confirmation from you before yanking the pages.

R*Command*

Roll: copy the buffer out and read the next page or window in.

Formats

R
nR

This command copies the contents of the current buffer to the output file, clears the buffer, and reads a page or window from the input file into the buffer. The command allows you to change buffer contents quickly and easily.

Numerical Argument

This command accepts only positive integers. The nR command performs the command *n* times.

Symbolic Modifier

This command accepts the colon modifier (:). The :R command inhibits an error message, returning a +1 if it succeeds and a 0 if it fails. You can use these values as numerical arguments to the next command. See the colon symbol in this dictionary.

Inclusion in an iteration automatically simulates the : colon modifier.

Characteristics

Unless you entered SPEED with the /D switch, this command will not automatically display the page or window it reads in. For display, see the T command and the entry for the /D switch in this dictionary.

Precautions and Error Messages

SPEED treats a OR command like an R command and attempts to read in the next page.

If there is no open input file, SPEED displays the message

Error: No open file

If you try the command when you have already read in the last window or page, SPEED displays the message

Error: No more characters in input file

If you try to read in a second window or page with no open output file SPEED will again send you the message

Error: No open file

Related Commands

To retain the contents of the current buffer but add to them from the input file, see the A command. To read the buffer and the rest of the input file into the output file, see the E command. To read into the output file from the buffer, with or without clearing the buffer, and with or without appended form feeds, see the P and PW commands. To discard the contents of the buffer and read in from the input file, see the Y command.

Example

```
!15WM$FOintro$A#T$$  
!3:R*ZJ-20T$$
```

You set window mode at 15 lines, open the file INTRO for updating, read the first window into the buffer and type it out. You see that this is the file you want. You wish to add to the contents of the fourth window, so you issue the command again with the proper numerical modifier. You use the colon modifier so that if the R command succeeds, the CP will move to the end of the buffer. You use the following T command to check your position.

S*Command*

Search for a string of text in the current buffer.

Formats

S [*string*]
nS [*string*]
-nS [*string*]
m,nS [*string*]

Within the range of text you specify, this command matches **string**, the string of text in the command, or the contents of the previous *search string*, with the first instance of that string in the current buffer.

Unless the search string contains a CTRL-G template, as described in the “Characteristics” section below, the command repositions the CP at the end of the string of text it matches.

Numerical Arguments

This command takes positive, negative, and paired numerical arguments.

0Sstring\$

searches from the beginning of the current line up to the position of the CP.

1Sstring\$

searches from the position of the CP to the next NEW LINE character; that is, to the end of the current line.

nSstring\$, except 0Sstring\$,

searches from the current CP position forward to the *n*th NEW LINE character; that is, through the next *n* lines, including the current line.

-1Sstring\$

searches the preceding line and the current line up to the position of the CP.

-nSstring\$

searches from the *n*th NEW LINE preceding the CP up to the CP; that is, it searches the preceding *n* lines and the current line up to the CP.

m,nSstring\$

searches from the character following the *m*th character up to and including the *n*th character. If you specify simple numbers, SPEED counts *m* and *n* from the beginning of the buffer. You may use more complicated numerical expressions and specify *m* and *n* using the values of variables and pseudo-variables. For example,

Z/2, Z-V9Sclaim\$ conducts a search for *claim* from the character whose position from the beginning of the buffer is nearest the middle of the buffer to the character whose position from the end of the buffer is the current value of Variable 9, whatever that is, and regardless of the current position of the CP.

.-100,.+100S%\$ restricts the range for a search for instances of the percent sign (%) from the 99th character preceding the current CP position to the 100th character following the current CP position.

#Sstring\$

abbreviates 0,ZSstring\$ and specifies the entire current buffer as the range of the search.

The default value

for Sstring\$ specifies the range of the search from the current CP position to the end of the buffer.

Symbolic Modifiers

This command takes both the commercial at modifier (@) and the colon modifier (:).

@S%string%

allows you to set as a temporary delimiter the first character following the command name, in this case the percent sign. This allows you to use the standard delimiter, ESC, which echoes as \$, as part of your search string.

:Sstring\$

inhibits error messages, returning the value +1 if the search succeeds, and the value 0 if the search fails. You may use this value as a numerical argument to the next command; see the colon symbol.

S (continued)

Inclusion of a search command in an iteration automatically simulates the : colon modifier.

You may combine the two symbolic modifiers in either order for both effects.

Characteristics

The search string in this command must end in a delimiter, ordinarily ESC (which echoes as \$) within a command line, or CTRL-D (which echoes as \$\$) at the end of a command line. To specify a temporary delimiter in order to include the standard delimiter ESC in your search string, see the "Symbolic Modifiers" section in this entry.

If you omit the search string argument, issuing the S\$ forms of the command, the command will search for the string that the previous C, N, Q, or S search command tried to match.

If the command conducts a successful search, and your search string contains no CTRL-G template, SPEED will position the CP at the end of the matched string. The position of the CP following an unsuccessful search depends on the value assigned to position mode. (To find out your position mode, issue a WP= command; see the WP command for details.) If your position mode is set at 0, the default setting, the new position of the CP will be

- n lines after the position before the search for nS commands, and at the CP position for OS commands
- at the position before the search for -nS commands
- after the n^{th} character for m,nS commands
- at the beginning of the buffer for default S commands

If you set the position mode to $+n$ or $-n$, the new position of the CP after an unsuccessful search will be at the position where the search actually starts, that is,

- at the position before the search for nS commands and at the beginning of the line for OS commands
- n lines before the previous position for -nS commands
- after the m^{th} character for m,nS commands
- at the position before the search for default S commands

This command accepts templates and expansions to files and buffers, like the other search commands C, N, and Q. See the individual entries in this dictionary for the behavior of each template.

If you issue an S command with a CTRL-G template in the search string, and the search is successful, the Character Pointer will reposition at the point of the CTRL-G insertion rather than at the end of the matched string. That is, if you successfully search for pass]Gword, the result will be not *password** but *pass*word*.

The value of search case match mode determines whether or not the S commands will match characters regardless of case; see the WS command. In the default value of case match mode, 0, the command matches independently of case (*a* matches *A* in the search string argument). If you set the value at anything other than 0, the match in the search string argument will be case independent (*a* matches *a* but not *A*).

Precautions and Error Messages

If m is greater than n , SPEED displays the message

Error: First argument greater than second argument

Recalculate and reissue the command, if you still wish to execute it.

If you issue an S\$ command (without a search string) when you have conducted no previous C, N, Q, or S search, SPEED displays the message

Error: Incomplete string in search buffer

and does not conduct a search.

If you issue an S\$ command (without a search string) and the previous search string was longer than 31 characters, SPEED displays the same message and continues the search, using the incomplete string (the first 31 characters of the previous string).

If you issue a S]Gstring\$ command (with CTRL-G preceding the first search string character), and the search succeeds, SPEED continues to match *that instance* of the string on subsequent S\$ commands. To search for a further instance, you must move the CP forward before you issue the S\$ command.

In search strings, SPEED flags any control character other than ↑B (expansion to a buffer), ↑F (expansion to a file), TAB (↑I), NEW LINE (↑J), vertical tab (↑K), form feed (↑L), CR (↑M), or ESC as an error unless you precede it with the control characters ↑G or ↑W (on models other than 6052, 6053, D100, or D200, ↑↑ or ↑_.)

Related Commands

To search for and change text with one command, see the C command. To make the same search a number of times, use a Command Loop; see the <x> command. To search not only through the buffer but through the rest of the pages or windows of your input file, see the Q command if you are willing to discard the searched material or the N command if you wish to retain the searched material in the file.

Examples

You are editing some instructions your supervisor wishes to distribute.

the writer correct all mistakes.*

```
!0Ster$I□will$$
```

the writer will correct all mistakes.*

You spot an omission in your current line. Your search command takes you to the beginning of the line and moves forward to the position for the insertion command.

```
!:Seach$*1T$$
```

and every programmer must present

You suspect yourself of some flabby prose. You issue a search command with the colon modifier, so that if it succeeds, you will get a display of the rest of the line, and if it fails, you will not. The search command succeeds, and the display takes place, since $1 * 1 = 1$. You can now delete the offending phrase with a 10D command. After doing so, you wonder if there is not a further case of the same problem. You issue the command

```
!:S$*1T$$
```

without a search string, so that it will use the previous search string. SPEED reveals

and every accountant should report

and you may delete the phrase again.

T

Command

Type out or display text, showing the position of the CP.

Formats

```
T
nT
-nT
m,nT
#T
@T%string%
```

Function

This command displays text on a video display terminal or types it out at a hard-copy terminal. At the terminal it shows the Character Pointer position with a flashing asterisk on models 6052, 6053, D100, and D200. On other models, it shows the CP position with a caret (^) in parentheses.

Numerical Arguments

This command accepts positive, 0, negative, and paired numerical arguments.

0T

displays the current line from the beginning of the line up to the position of the CP. The ZT command displays text from the current CP position to the end of the buffer. If that text exceeds the display characteristics of the screen, the text will roll to the bottom of the buffer.

1T

displays the current line from the position of the CP to the first NEW LINE character.

nT, except for 0T,

displays the contents of the current buffer from the CP location up to the n^{th} NEW LINE following the CP; that is, it shows n lines, counting the line in which the CP resides.

-1T

displays the immediately preceding line and the current line up to the position of the CP.

-nT

displays the contents of the n preceding lines and the current line up to the position of the CP.

T (continued)

m,nT

displays the contents of the current buffer from the character following the m^{th} character up to and including the n^{th} character. Simple numbers specify positions from the beginning of the buffer. You may specify these arguments with more complex numerical expressions, and with the values of variables and pseudo-variables. For example, Z/2,.T specifies a display from the middle of the buffer to the current CP location if the CP is in the second half of the buffer.

#T

abbreviates the O,ZT command, which specifies a display of the contents of the entire buffer. If the length of the contents exceeds the display characteristics of the screen, the text will roll to the bottom of the buffer.

The default value (T)

specifies a display of the entire current line, regardless of the position of the CP on it. The WA command does not affect the default value for this command.

Symbolic Modifier

This command accepts the commercial at modifier (@). You specify the first character following the command name as a temporary delimiter (denoted by % in the format above). This form of the command allows you to type out or display a string containing the standard delimiter ESC. This command does *not* insert the string in the text or in any buffer; see instead the BG command.

Characteristics

This command does not affect the position of the Character Pointer. To move the CP, see the C, J, L, M, N, Q, and S commands.

This command does not alter text. To change, delete, or insert text, see the C, D, I, and K commands.

The WD command, which sets the display mode, does not affect the T command. Whichever value of that mode you select, a #T command will still display the entire buffer.

Precautions and Error Messages

To halt a rolling screen for inspection of text after issuing a #T command, freeze the terminal with a CTRL-S. To resume the roll and regain control of the terminal, issue a CTRL-Q.

If you issue a T command with no result, you may have inadvertently frozen the terminal. Issue a CTRL-Q to release it. If the terminal is not frozen and your text seems to have disappeared, check the contents of the buffers with the B? command. You may have inadvertently issued a K or Y command and lost the contents of the buffer. (To retain your input files in such a case, issue a CTRL-C,CTRL-B interrupt.)

If you get an unexpected display, such as misplaced text, you may have mistaken the position of your Character Pointer. Issue VC, VL, VM, VP, Z=, or .= commands to locate the CP in the text; see the individual entries for those commands in this dictionary.

Related Commands

To display command execution, use the Trace Mode Toggle; see the ? command.

To display momentarily a closed file, issue an X□TYPE□filename\$\$ command.

To avoid issuing T commands repeatedly while carrying out close editing tasks, enter SPEED with the display switch by issuing an X□SPEED/D□filename\$\$ or X□SPEED/D\$\$ command. SPEED will automatically display the 20 lines surrounding the CP before returning the prompt to you. (It will not do so, however, if your previous command was an X command or generated output; see the /D Switch in this dictionary.)

Example

```
!FRmedical$A$20T$$
```

You open the file MEDICAL, append the first page or window to the buffer, and display the first 20 lines to make sure you opened the correct file.

V*Command*

Return the Value of a variable.

Format

Vn

Function

This Variable Value command returns the value of one of the 10 SPEED variables, which you may use as numerical arguments to other commands, and to carry out calculations. The only legal variable names are the ten digits 0 through 9.

Numerical Arguments

None, but see the various Vx commands.

Symbolic Modifiers

None, but see the various Vx commands, the ampersand modifier (&), and the WR command.

Characteristics

A variable may take any integral value up to 65535. Variables functioning as arguments to SPEED commands are restricted to the range -32768 through +32767 for commands that take both positive and negative numerical arguments, and to the range 0 through +65535 for commands which take double or positive numerical arguments.

To find the value of a variable, issue a Vn= command.

Precautions and Error Messages

If you inadvertently omit a proper variable name, SPEED displays the message

Error: Illegal variable name

Related Commands

To insert the value of a variable in text, issue a Vn\ command; see the Backslash n\ command.

To insert a character for which the value of a variable is the ASCII decimal equivalent, issue a Vn! command; see the ! command.

To manipulate the values of variables, see the VD, VI, and VS commands. To use the values of variables as conditions on the execution of commands, consult the information under Conditional Execution and Command Loop; see the n"Xcommand-line' and <command-line> commands.

Example

**How far will the CP move this time?*

!VOM\$\$

*How far will th*e CP move this time?*

Since the current value of V0 is 15, the CP moves 15 characters to the right.

VC

Command

Get the Value of a Character.

Format

VC

Function

This Character Value pseudo-variable gives the numeric equivalent of the ASCII character following the CP.

Numerical Arguments

None

Symbolic Modifiers

None, but see the remarks on the ampersand modifier (&) in the "Characteristics" section in this entry.

Characteristics

If the CP is at the end of the buffer, with no characters following it, the value of VC is 0, the default value.

This command stores the decimal (base 10) equivalent of the character. It does not take the ampersand modifier (&).

Precautions and Error Messages

Do not confuse this command, which gives the *value* of the character to the *right* of the CP, with the `.` command, which gives the *number* of characters on the *left* between the beginning of the buffer and the CP.

Related Commands

You may use the value of VC as an argument to the next command. For example, if the CP is to the left of the character T, the command VC displays 84, the ASCII Decimal equivalent of T. If you issue a VCI command, SPEED inserts a T in the text, and if you issue a VC\ command, it inserts 84 in the text. See the `=`, `nl`, and `n\` commands.

Use this command to find the value of a character quickly rather than having to resort to a character set table (such as Appendix A of this manual). Use this command in conjunction with the `n\` and `nl` command to convert character text to numerical equivalent text and back to character text.

Example

The text below is the total contents of the current buffer.

**THIS IS A SECRET.*

You wish to convert each of the characters of the sentence following the CP to their ASCII decimal equivalents, perhaps as an elementary code. You type, remembering to insert spaces to keep the numerals apart

```
!Z<I□$VC\1D>$$
```

and SPEED replaces your text with

```
84 72 73 83 32 73 83 32 65 32 83 69 67 82 69 84 46*
```

Your VC command within the brackets of the Command Loop serves as a numerical argument to the `\` command, and consequently it inserts the value of the next character in the text. The next command inserts a space, and the last command in the Loop deletes the character whose value you have just inserted.

VD*Command*

Value Decrement: decrement the value of a variable by 1.

Format

VDn

Function

This Variable Decrement command subtracts 1 from the value of the n^{th} variable and resets the variable to the decremented value.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

The only legal variable names are the digits 0 through 9.

Precautions and Error Messages

If you attempt to iterate this command with a string of Ds, or attach a numerical modifier to the command, you will get the error message

Error: Illegal variable name

and processing ceases.

If you try a multiple decrement with VnD, of course, SPEED attempts Vn deletions.

Related Commands

You may use this command as a numerical argument to the next command. For example, if V5 = 666, the command VD5= displays the new value 665.

To decrement in larger amounts, apply a VS command to the variable (Vn-xVSn).

Use this command as a numerical argument for the control of Command Loops and Conditional Executions; see the O, <command-line>, and n'Xcommand-line' commands.

Example

You wish to key in a line of decreasing numbers.

```
!15VS5$!line!V5\VD5''GI□$Oline$'T$$  
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1*
```

You set Variable 5 to 15 initially. Each time the Conditional Execution applies, the VD command reduces Variable 5 by 1. When the result of the reduction is no longer greater than 0, the Conditional Execution halts and control passes to the next command, T. The effect of the commands is to type into the buffer a sequence of numbers automatically.

VI*Command*

Value Increment: increment the value of a variable by 1.

Format

VIn

Function

This Variable Increment command adds 1 to the value of the n^{th} variable and resets the variable to the incremented value.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

The only legal variable names are the ten digits 0 through 9.

This command does not *require* a delimiter, but it attempts to act as a numerical argument to the next command if you do not use one. Type in a delimiter when the next command takes a numerical argument and you do not wish this command to serve as one.

Precautions and Error Messages

If you attempt to iterate this command with a string of Is, SPEED displays

Error: Illegal variable name

If you attempt a numerical argument before the V or after the variable name, SPEED tells you

Error: Syntax error

and processing ceases. If you specify a number between the V and the I, of course, SPEED will attempt to insert the character whose ASCII value you inadvertently specified, and process the next symbol as a command.

Related Commands

You may use this command as a numerical argument to the next command. For example, if V5 = 666, the command VI5= displays the new value 667.

To decrement in larger amounts, apply a VS command to the variable. For example, V4+10VS4 resets Variable 4 at 10 more than its current value.

Use this command as a numerical argument for the control of Command Loops and Conditional Executions; see the O, <command-line>, and n'Xcommand-line' commands.

Example

You want to type out the alphabet.

```
!0VS026<V0+65IV10>T$$
```

```
ABCDEFGHIJKLMNPOQRSTUVWXYZ*
```

You initially set Variable 0 to 0. Each time the Command Loop applies, it inserts the letter whose ASCII value is 65 + the value of Variable 0. (Uppercase letters begin at ASCII 65.) It then increments the value of Variable 0 by 1, so that the next time the Loop applies, it inserts the character with the next higher value. Since you specified 26 executions, character insertion stops at Z. The effect of the commands is to list the alphabet automatically.

VL

Command

Get the Value of the current Line.

Format

VL

Function

This Line Number pseudo-variable stores the number of the line on which you have placed the CP in terms of lines from the beginning of the buffer.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

This command does not *require* a delimiter, but it attempts to act as a numerical argument to the next command if you do not use one. Type a delimiter when the next command takes a numerical argument and you do not wish this command to serve as one.

This command counts the beginning of the buffer as the first line, and all subsequent NEW LINES as beginnings of lines. It does *not* count a carriage return (CR) as the start of a line.

The VL command counts lines from the beginning of the window or page *in the buffer* and not from the beginning of your input file unless these two coincide.

To find out the number of your current line, issue a VL= command.

Precautions and Error Messages

Do not confuse this command with the VN command, which stores the *total* number of lines in the buffer. (Since the VN command does not count uncompleted lines, the value of VL in the last line will be one greater than the value of VN, if you don't end the last line with a NEW LINE.)

Related Commands

You may use this command as a numerical argument to the next command. For example, you can number the lines in the buffer by issuing a VL\ command after each NEW LINE you type.

Use this command to find the number of the current line in order to jump back to it later using an nJ command. To find the distance from the beginning of the buffer in terms of characters rather than lines, issue a .= command. To find the distance of the CP from the beginning of the line, see the VM command. To find the total number of lines in the buffer, see the VN command.

Example

```
!VL\I.—$$
```

```
55.      *
```

At the beginning of the 55th line, you use the VL command as an argument to the n\ command in order to insert the current line number in the text.

VM*Command*

Value Moved: get the number of characters between the beginning of the line and the CP.

Format

VM

Function

This Move Number pseudo-variable gives the number of characters between the beginning of the line and the Character Pointer.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

This command does not *require* a delimiter, but it attempts to act as a numerical argument to the next command if you do not use one. Type in a delimiter when the next command takes a numerical argument and you do not wish this command to serve as one.

This command counts from the beginning of the buffer or the last NEW LINE character, whichever is closer. It does *not* count a carriage return (CR) as the beginning of a line.

Precautions and Error Messages

Do not confuse this command with the VC command, which returns the ASCII decimal equivalent of the next character. Also do not confuse it with the VP command,

Precautions and Error Messages

Do not confuse this command with the VL command, which gives the number of the *current* line.

which returns the position of the CP before the last search, or the . = command, which returns the position of the CP in characters from the beginning of the buffer.

Related Commands

You may use this command as a numerical argument to the next command. See in particular the nl, =, and n\ commands. This command does not take the ampersand modifier (&) to specify an alternate radix, but those commands do; see the ampersand symbol and the WR command.

Example

After entering a heading, you decide to center it, recalling that there are 80 spaces to a line.

```
INTRODUCTION*
```

```
!80-VM/2VS0$LV0<I□$>$$
```

```
*INTRODUCTION
```

You subtract the number of characters on the line that you have already typed from 80 to learn how many spaces remain. You divide the result by two, to get the number of spaces you want on the left. You store that number in Variable 0 because you are going to move the CP from the current position. You go to the beginning of the line and issue a Command Loop for inserting the number of spaces equal to the value in Variable 0, which is half of the length left on the line and the desired result.

VN*Command***Value Number: get the total number of lines in the current buffer.**

Format

VN

Function

The Number of Lines pseudo-variable stores the total number of lines in the current buffer, regardless of the current position of the CP.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

This command does not *require* a delimiter, but it attempts to act as a numerical argument to the next command if you do not use one. Type in a delimiter when the next command takes a numerical argument and you do not wish this command to serve as one.

This command counts NEW LINES. It does *not* count as a separate line anything ending with a carriage return (CR).

Since this command does not count incomplete lines, the value of VN will be one less than the value of VL at the end of the buffer (if the last line does not end in a NEW LINE).

This command gives the total number of lines for the current page or window *in the buffer* and not the total number of lines in your current input file unless those two happen to coincide.

Related Commands

You may use this command as a numerical argument to the next command. For example, VN/2J will take you to the beginning of the line nearest the middle of the buffer. To find the number of lines in the buffer, issue a VN= command.

Use this command to determine the length of the current buffer in terms of the number of lines. To determine the buffer's length in numbers of characters, issue a Z= command. To determine the length of the current line from the beginning of the line to the CP, see the VM command. To determine the distance from the beginning of the buffer to the CP in number of characters, issue a .= command.

Example

!VN=\$\$

66

!ZJ|L\$\$

You use the VN command as an argument to the = command to find out that the buffer contains 66 lines, the number of lines on a standard (8 1/2 by 11-inch) line printer and typewriter page. You jump to the bottom of the buffer and insert a form feed to stay within the 66-line format.

VP

Command

Value Position: get the position of the Character Pointer before the last search.

Format

VP

Function

This Previous Position pseudo-variable gives the position of the CP before the last search in terms of the number of characters between the CP and the beginning of the buffer. Use this command to trace, and perhaps return to, your previous CP position without resorting to close inspection of your text.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

This command does not *require* a delimiter, but it attempts to act as a numerical argument to the next command if you do not use one. Type a delimiter when the next command takes a numerical argument and you do not wish this command to serve as one.

This command specifies the CP after the last *unsuccessful* search when the position mode is +1 (or nondefault); that is

- at the position before the search for default C and S commands
- at the position before the search for nC and nS commands

- *n* lines before the previous position for -nC and -nS commands
- after the *m*th character for m,nC and m,nS commands

For further details, see the WP command.

Precautions and Error Messages

Do not confuse this command with the WP command, which *sets* the position mode. To find your previous CP position, issue a VP= command. To find your *current* position, issue instead a .= command.

Related Commands

You may use this command as a numerical argument to the next command. See in particular the nl, =, and n\ commands. Although this command does not take the ampersand modifier (&) for a temporary alternate radix, those commands do; see the ampersand symbol and the WR command.

Example

```
!J$VPM$$
```

You realize that you have further editing to do at the position of the last search. You jump to the beginning of the buffer and issue a command to move forward the number of characters equal to the *current* value of VP, which is the position of the CP *previous* to the last search.

VS

Command

Value Set: set a new value for a variable.

Format

iVS*n*

Function

This Variable Set command sets the *n*th variable equal to the numerical expression *i*.

Numerical Arguments

This command takes positive and negative numerical arguments.

Symbolic Modifier

This command takes the ampersand modifier (&). The &iVS*n* command stores the number *i* to the base of the alternate radix rather than as a decimal (base 10). If your alternate radix is binary (base 2), and you issue a &11VS8 command, you get the response 3 when you issue a V8= command. See the ampersand symbol and WR command for details.

Characteristics

This command does not *require* a delimiter, but it attempts to act as a numerical argument to the next command if you do not use one. Type in a delimiter if the next command takes a numerical argument and you do not wish this command to serve as one.

The only legal variable names are the ten digits 0 through 9. The numerical expression is the *preceding* argument and the name of the variable is the *following* argument.

SPEED variables are restricted to the range -32768 through +32767 if they apply as positive or negative numerical arguments, and to the range 0 through +65535 if they apply as paired or positive numerical arguments.

You may set a variable with the value of another variable or a pseudo-variable. For example, Z/VN+V6VS9 sets Variable 9 equal to the sum of Variable 6 (whatever that is) and the result of dividing the current number of characters in the buffer by the current number of lines in the buffer. For likely candidates as arguments to the VS command, see all the commands starting with V and W, and the symbols Z and period (.).

All variables are set to 0 when you enter SPEED.

To find the value of a variable to see if it needs resetting, issue a V*n*= command.

Precautions and Error Messages

If you inadvertently enter a VS*n* command *within* a numerical expression, following an arithmetic operator, you set the value of the variable to that portion of the expression preceding the variable; that is, if you type in 3+5+VS0-2=, you set Variable 0 to 8, lose your previous value, and get the response 6.

You may set a variable to numerical expressions more complex than simple numerals, but remember that SPEED truncates the answer. 7/2VS1, for example, sets Variable 1 to 3.

Related Commands

Set variables to initial values when you wish to conduct a count of certain characters or strings, when you wish to perform a Command Loop a determinate number of times, and when you wish to control a Conditional Execution; see the O, <command-line> and n''Xcommand-line' commands for details.

Examples

```
!JOVS2$<S%$;VI2>V2=$$
```

55

You jump to the beginning of the buffer and set Variable 2 to 0. Your subsequent commands search for instances of the percent sign (%), add each instance to the value in Variable 2, and finally display that value, which in this instance is 55.

```
!15VS3$!!line!16-V3\VD3''GI□$Oline$'T$$
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15*
```

You set Variable 3 to 15. Your subsequent commands use the decreasing value of that variable to insert a string of numerals from 1 to 15 automatically.

WA

Command

Window Argument: set a new value for the default argument of deletion and movement commands.

Formats

WA
nWA

Function

This Default Argument Mode command sets a new value for the default argument of the D, J, K, L, and M commands. The new value is valid for the remainder of the editing session, or until you choose to reset the value. If you exit from and re-enter SPEED, the value reverts to the default value, 0.

Numerical Arguments

Although you may issue the command with any numerical expression you wish, the command takes only two values, 0 and +1. If you issue the command with a nonzero argument, you will set the default argument value at +1.

If you issue the deletion and movement commands listed above without numerical arguments, they behave differently depending on the default argument value you select.

If you set the value at 0 by issuing a OWA command,

- the D command has no effect
- the J command moves the CP to the beginning of the buffer
- the K command kills characters from the beginning of the current line up to the CP
- the L command moves the CP to the beginning of the current line
- the M command leaves the CP in its current position

If you set the value at +1 by issuing an nWA command (where *n* is not 0),

- the D command deletes one character to the right of the CP
- the J command moves the CP to the beginning of the buffer

- the K command kills characters from the the CP through the end of the line (including the NEW LINE character)
- the L command moves the CP to the right of the next NEW LINE, that is, one line forward
- the M command moves the CP one character to the right

The default value of the WA command itself is 0 when you enter SPEED, and is the previous value if you have already issued nWA commands.

Symbolic Modifiers

None

Characteristics

You may use WA itself as a numerical argument to other commands. For example, if WA is set at 0, and you issue a WAK command, you will kill characters from the beginning of the line up to the CP, as if you had issued an OK command. A 1WAL command will both move you forward one line and reset WA at 1.

To find your current default value, issue a WA= command.

Precautions and Error Messages

If you are wary of inadvertent movements of the CP and deletions, leave the setting at 0. To save keystrokes when you need to edit a line quickly and closely, set the value at +1. When your editing style settles in, pick the mode that suits you and don't vary it except when pushed to it. This discourages costly errors.

Related Commands

None

Example

**iss one of t*he*

```
!1WA$MDSf$DMD$$
```

*is one of t*he*

By resetting the default value before working on the line above, you save several typing strokes.

WC*Command***Window Case: create and edit lowercase and uppercase files from an uppercase terminal.**

Formats

WC
OWC
nWCcharacter1\$
nWCcharacter1character2
-nWCcharacter1\$
-nWCcharacter1character2

Function

This Case Control Mode command enables you to create and edit files containing both uppercase and lowercase alphabetic characters from a terminal that has only uppercase characters. To do this, you select a character, *character1*, to serve as a temporary shift, and you have the option of selecting a second character, *character2*, to serve as a temporary shift lock.

Use this command when you are at a terminal that handles only uppercase text, and you wish to prepare text containing both uppercase and lowercase material. Use **1WC** if most of your copy is lowercase; use **-1WC** if most of your copy is uppercase.

Numerical Arguments

This command takes positive and negative numerical modifiers. It treats all nonzero numerical expressions as +1.

OWC

turns case control off. The terminal reads characters exactly as you type them in, with no translation from uppercase to lowercase.

1WCcharacter1\$ and 1WCcharacter1character2

designate *character1* as the shift-up character. SPEED then treats any alphabetic character that follows *character1* as uppercase. It treats all other characters as lowercase. If there is a *character2*, it treats all characters between its first occurrence and its second occurrence (or an ESC) as uppercase

-1WCcharacter1\$ and -1WCcharacter1character2

designate *character1* as the shift-down character. SPEED then treats any alphabetic character that follows *character1* as lowercase. It treats all other characters as uppercase. If there is a *character2*, it treats all characters between its first occurrence and its second occurrence (or an ESC) as lowercase

The default value

for this command is 0 when you enter SPEED

Symbolic Modifier

This command takes the colon modifier (:). The unmodified command affects only those characters that you type from the keyboard. The **:WC** form of the command extends case control to the entire command line (that is, to the next CTRL-D), including characters that you insert in the buffer by using the control key templates **]Bbuffer-name** and **]Ffilename**. SPEED first converts character sequences brought in by these templates to uppercase, and then interprets the letters as uppercase or lowercase in the same way as the unmodified commands.

The **:WC** commands return a value of +1 if you have extended case control with the colon modifier; otherwise they return 0. You may use this form of the command as a numerical argument to the next command. See the colon symbol in this dictionary.

Characteristics

If you do not wish to define a shift lock in addition to the shift, you must type in a delimiter after you have selected your shift key. Conversely, if you wish to define a shift lock, you must insert it directly after the shift key with no intervening delimiter.

The shift-lock key shifts the case of the entire string you type in following it until you depress the lock key again or terminate the command with a CTRL-D.

You may set and use a case control mode value in the same command line. The case changes take effect for the command line once you execute the command.

The command ignores a shift character in the scope of a shift-lock character.

The **WC** commands, like the ALPHA LOCK key, affect only the display of the alphabetic keys. Use numeric and other symbol keys as you normally would, whatever value of case control mode you select.

In order to insert the character denoted by your shift key, type it in twice: if you make = your shift key, type in == to insert = in text. To insert the character denoted by your shift-lock key, precede it with the shift key.

WC (continued)

To simulate an uppercase-only terminal at a terminal that has both uppercase and lowercase characters, switch ALPHA LOCK on and proceed as outlined above.

To find the value of your case control mode, issue a WC= command.

Precautions and Error Messages

Do not type a preceding shift-character when typing a shift lock character unless you wish to echo the shift-lock character.

Remember to issue an l command as needed after this command. It is easy, especially for experienced typists, to omit it inadvertently after setting the case characteristics.

Related Commands

You may use the value (0, +1, or 65535 = -1) of this command as an argument to the next command. For example, if your case control value is +1, then WCM moves the CP one character to the right.

Example

At an uppercase-only terminal, you set case mode at +1 and type in:

```
!1WC=+$I=PUT ONLY =WORD 6 IN
+CAPITALS,+ PLEASE.$$
```

When you reset to 0, your terminal displays

```
=PUT ONLY =WORD 6 IN
=C=A=P=I=T=A=L=S=, PLEASE.
```

If you are at a terminal that has both cases, with case control mode set to 1, you get the same display. If you reset case control mode to 0, and ask for a display, SPEED shows you

Put only Word 6 in CAPITALS, please.

WD

Command

Window Display: set the automatic display mode value.

Formats

WD
OWD
nWD

Function

This Display Mode command sets the value of the display mode, which determines what SPEED attempts to display on the screen automatically before restoring your prompt.

Use this command when you know how much text you wish displayed before each prompt. Set a generous number if you need display feedback. Set the display at 0 for a faster return of prompt.

Numerical Arguments

This command accepts positive numerical modifiers from 1 to 10. If you exceed 10, SPEED sets the display value at 10. When you set the display mode at *n*, SPEED tries to display the *n* lines preceding and the *n* lines following the CP, showing the CP as a blinking asterisk on terminal models 6052, 6053, D100, and D200, or as a caret within parentheses on other models.

If you are less than *n* lines from either the beginning or end of the buffer, SPEED displays additional lines in the other direction such as to total *2n*.

OWD turns the display mode off. If you wish any display, you will have to issue appropriate T commands one by one.

The default argument for this command, WD, is 0 when you enter SPEED without the D switch, and the same value as the preceding WD command if you have issued one.

Symbolic Modifiers

None

WD (continued)

Characteristics

Even if your display mode is on, SPEED will not display text if the previous command generates output or is an X command.

To find your display mode, issue a `WD=` command.

Even when your display mode is set low or to 0, you may obtain larger displays by using a T command; see that command.

Precautions and Error Messages

Do not confuse this command with the WM command, with which you decide how many lines of text to read into the buffer at one time. See that command.

Related Commands

If you know before you enter SPEED that you want the widest display, append the display switch, entering with a `X□SPEED/D` or `X□SPEED/D□filename` command; see the entry for the D switch in this dictionary.

To obtain a nonautomatic display of text in the current buffer, see the T command.

To display command lines as SPEED executes them, use the Trace Mode Toggle; see the ? command.

Example

```
) X□SPEED)
!10WD$FOlux$$
```

You enter SPEED but forget to append the D switch. You set the display to 10 lines with the WD command and open the file LUX for update. SPEED displays the first 20 lines of the first page.

WM

Command

Window Mode: choose a window of text to read into the buffer at one time.

Formats

WM
OWM
nWM

Function

This Window Mode command allows you to determine how much text at a time SPEED reads into the buffer from the current input file. You may specify a particular number of lines, in *window* mode, or you may choose to place text in the buffer from form feed to form feed, in *page* mode.

You may find it convenient to remain in page mode while composing text and decide afterward where to place page breaks (form feeds). A convenient window for display at the terminal is 20 lines, or you can simulate a standard (8 1/2 by 11-inch) typewritten page with 66 lines (at 6 lines to the inch).

Numerical Arguments

This command takes nonnegative numerical arguments.

OWM

sets SPEED to read text into the buffer in page mode, from form feed to form feed. SPEED treats `-nWN` as OWM.

nWM

sets SPEED to read text into the buffer in window mode, *n* lines at a time.

The default value for WM

is 0 when you enter SPEED, and the previous value if you have issued any WM commands.

Symbolic Modifiers

None

Characteristics

You may issue a WM command at any time, but the change does not take effect until you either close the file or issue an R command, reading more material into the buffer. If you change from window to page mode and read in new material, your page extends from the end of the last window to the next form feed.

If you read in page mode into the buffer from a file containing no form feeds, SPEED tries to read the entire input file into the buffer.

WM (continued)

If you set a window mode for more lines than you have on your pages, SPEED displays the lines and echoes the form feed with |L.

When you are in page mode, SPEED does not place the end form feed in the buffer. (The beginning form feed is the last character of the previous page.) SPEED remembers the position of the form feed and puts it back in the text when you read the page to the output file. Consequently, if you wish to search for or delete form feeds, you must switch to some arbitrary window mode. (Remember to switch back again to page mode to confirm any deletions.)

To find the value of your window mode, issue a **WM=** command.

Precautions and Error Messages

If a page is too long, SPEED displays the message

Error: Insufficient memory available

and is unable to bring the whole page into the buffer. You must abort the session with a CTRL-C, CTRL-B sequence. (Any other meaningful action will require memory.)

If a line of the file exceeds 136 characters, SPEED displays the message

Error: LINE TOO LONG

but brings the page into the buffer. In this case you may want to adjust files and lines accordingly.

Related Commands

To insert a form feed, issue a |2I\$ or |L\$ command; see the CTRL-L template in this dictionary.

Example

```
!20WM$FOlisting$$  
!R$$  
!R$$  
!OWM$RZJ$$
```

You set the value of window mode at 20 and open the file LISTING, which contains no form feeds. You find what you were looking for in the third window, reset the value to page mode, read in the rest of the file, and jump to the bottom of the buffer to continue composing.

WP Command

Window Position: set the positioning of the CP after unsuccessful searches.

Formats

WP
OWP
nWP

Function

This Position Mode command sets the way in which SPEED repositions the Character Pointer after an unsuccessful C or S command.

Numerical Arguments

This command takes nonnegative numerical arguments. It treats all nonzero arguments as 1.

If you issue a OWP command, the CP positions

- at the beginning of the buffer for default (numerically unmodified) C and S commands
- *n* lines beyond the previous position for nC and nS commands
- at the position before the search for -nC and -nS commands
- after the *n*th character for m,nC and m,nS commands

If you issue an nWP command where *n* is not zero, the CP will reposition

- at the position before the search for default C and S commands
- at the position before the search for nC and nS commands
- *n* lines before the previous position for -nC and -nS commands
- after the *m*th character for m,nC and m,nS commands

The default value for this command is 0 when you enter SPEED and is the value of the previous WP command if you have issued one.

Symbolic Modifiers

None

WP (continued)

Characteristics

When the value of this command is not 0, the CP always repositions before the character where the unsuccessful search begins.

To find the current value of the position mode, issue a **WP=** command.

Except for the default commands, the 0 value repositions as far forward as the range permits, and the 1 value repositions as far backward as the range permits.

Precautions and Error Messages

Do not confuse the **WP** command with the **VP** command, which places the CP at its previous position after a search.

Related Commands

See the **WA** command for resetting default values.

Example

```
!1WP$$#$$
```

You set position mode to 1 since you want the CP to remain in position if, as you hope, there are no further instances of # in the buffer.

WR

Command

Window Radix: set a new alternate radix.

Formats

WR
nWR

Function

This command allows you to select a new alternate radix. The standard radix is always 10 (decimal). You may choose an alternate radix from 2 (binary) to 36.

Numerical Arguments

This command takes positive numbers from 2 to 36 and numerical expressions which evaluate within that range. An nWR command sets the alternate radix at *n*.

The default value for this command is 8 (octal) when you enter **SPEED**, and the value of the previous **WR** command if you have issued one.

Symbolic Modifiers

None, but see the ampersand modifier (&) for its interaction with this command.

Characteristics

Although the **WR** command sets the alternate radix until you reset it or exit from **SPEED**, the ampersand modifier (&) switches from decimal to the alternate radix only for a single **SPEED** argument or command. For the details of its behavior, see the ampersand symbol.

To find the value of your alternate radix, issue a **WR=** command.

Related Commands

None

Example

```
!2WR$&11111111=$$
```

```
255
```

You switch the radix to 2 to find out that 11111111₂ is 255₁₀.

WS*Command*

Window Shifts: set case dependency for matching alphabetic characters in search commands.

Formats

WS
OWS
nWS

Function

The WS (Window Shifts) command allows the search commands C, N, Q, and S either to ignore or be sensitive to the case of alphabetic characters when carrying out a match.

This command allows you to conduct efficient searches through your text. If, for example, you wish to search for a word regardless of whether it begins a sentence or not, set the value of the search case match mode at 0. If you wish to change Bill, but not bill, to William, set the value at 1.

Numerical Arguments

This command takes single numerical modifiers. It treats all nonzero numerical expressions as 1.

OWS

permits case-independent searches: u in the command matches *U* and *u* in the text, and L in the command matches both *L* and *l* in the text.

nWS, n not equal to 0,

forces case-dependent searches: u in the command matches *u* but not *U* in the text, and L in the command matches *L* but not *l* in the text.

The default setting, WS,
is 0 when you enter SPEED.

Symbolic Modifiers

None

Characteristics

This command does not affect searches for non-alphabetic characters such as numbers, punctuation marks, or other symbols.

This command affects only the first text argument of the C command. You must specify case explicitly in the second argument.

To find your case mode, issue a WS= command.

Precautions and Error Messages

Do not confuse this command with the WC command, which enables you to edit lowercase files from an uppercase terminal.

Related Commands

None

Example

You have misspelled Kurt's name throughout a file.

```
!1WS$<CCurt$Kurt$;>$$
```

You set the value of the search case mode to case-dependency and issue a command to change Curt to Kurt throughout the text. By specifying a nonzero value, you avoid inadvertently changing curtsy and curtain to Kurtsy and Kurtain.

X *Command*

Execute Command Line Interpreter commands from SPEED.

Formats

Xstring\$
:X\$
:Xprogram.PR\$

Function

The Xstring\$ command executes *string* as a command of your parent process, which normally is the Command Line Interpreter (CLI). After the CLI executes the command, SPEED returns its prompt and waits for your next SPEED command.

Use the various forms of the X command when you wish to execute CLI commands or create subordinate processes without exiting from your current SPEED editing session. For example, you may find it convenient to compare some other file with the file you currently have open by issuing an XTYPE filename\$\$ command, rather than closing the current file, opening the other file for inspection, closing it, and reopening the previous file.

Numerical Arguments

None

Symbolic Modifier

This command accepts the colon modifier (:) if you omit the string argument. The :X\$ form of the command executes the CLI. You may issue any number of CLI commands or macros before you return to SPEED. You must terminate the CLI with a BYE command in order to return to SPEED.

The :Xprogram.PR\$ command is equivalent to XX program\$\$ and is a quick way to execute a program from SPEED when the program does not require switches or arguments from the command line. Control returns to SPEED once the command or program has completed execution.

Characteristics

This command requires a delimiter.

You may use any command name abbreviation in SPEED for CLI commands that you can use in the CLI.

You may use any punctuation in SPEED for CLI commands that you can use in the CLI.

If you have the privilege of creating more than two sons, you may create a subordinate SPEED process by issuing an XX SPEED\$\$ command. (You may attach the usual switches and filename arguments to this command.) You may open new files globally and locally in each subordinate SPEED process you create. If you do not know how many sons you have the privilege of creating, ask your system manager, or experiment with creating them until you get the message

ERROR: TOO MANY SUBORDINATE PROCESSES

To see what process you have, issue an XTREE\$\$ command.

Precautions and Error Messages

You may omit a space between the X command name and the CLI command or macro, but you must not omit any spaces required in the CLI command. For example, if you want a list of all your files whose names start with *a*, you must issue the command as XFILESTATUS a+\$\$ rather than as XFILESTATUSa+. If you omit the spacing, you receive the message

ERROR: NOT A COMMAND OR MACRO, filestatusa+

and the SPEED prompt.

If you use the CLI more frequently than you use SPEED, you may try to issue an Xstring) or Xstring\$) command. Remember to type a CTRL-D to execute the CLI command from SPEED.

The use of the X command does *not* return you to the *parent* CLI process of which SPEED is a *son*. Instead it executes a new CLI process as a son of your SPEED process. You must therefore have the privilege of creating two son processes in order to issue X commands successfully, since the CLI process you attempt to execute with an X command would be a grandson of the CLI process that you entered SPEED with.

Z *Symbol*
Last Character: end of the current buffer.

Format

Z

Function

The SPEED symbol Z represents the length, in characters, of the current buffer. Use the symbol in commands as a numerical argument representing the end of the buffer.

Examples

- !ZI\$\$ Insert the ASCII character whose decimal equivalent is Z, truncated to the lowest eight bits.
- !ZJ\$\$ Jump to the end of the buffer.
- !ZJL\$\$ Jump to the beginning of the last line (= VNJ).
- !ZK\$\$ Kill everything from the CP to the end of the buffer.
- !JZ/2M\$\$ Move to the middle of the buffer.
- !ZT\$\$ Display everything from the CP to the end of the buffer.
- !ZVSO\$\$ Store the number of characters in the buffer in Variable 0.
- !Z\\$\$ Insert the current value of Z in the text.
- !Z=\$\$ Display the current number of characters in the buffer.
- !Z<x>\$\$ Do the Command Loop x z times.

\$ (ESC) *Symbol*
Standard Delimiter

Format

\$

which echoes the ESC key

Function

SPEED echoes with the dollar sign symbol (\$) when you type the delimiter ESC or BREAK ESC (for escape). Use the delimiter as the last character of a text insertion string or search string. Use it also to prevent numerical values from acting as numerical arguments to the next command.

In this manual, \$ is entered in formats which require a delimiter. CTRL-D acts as a final delimiter in addition to terminating and executing the command line, so you need not issue an ESC before a CTRL-D.

To set a temporary delimiter in order to *include* \$ (ESC) in a text insertion string or search string, see the commercial at (@) symbol and the commands C, I, N, Q, and S.

Examples

You are watching Smith, a neophyte user of SPEED, make several natural errors.

Instead of

!!The□End.\$J\$\$

he types in

!!The□End.J\$\$

The End.J

Smith forgets to delimit his text with ESC, and SPEED treats J as more text rather than as a command.

*Line Alpha**
Line Beta
Line Gamma
Line Delta

Instead of

!3VS0L\$\$

he types in

!3VS0L\$\$

Line Alpha
Line Beta
Line Gamma
**Line Delta*

By omitting the ESC before the L command, Smith ends up at the beginning of Line Delta instead of Line Alpha, where he wants to be. SPEED used the new value of Variable 0 as the numerical argument to the L command.

!

Symbol

Prompt that SPEED is ready for input from the terminal.

Format

!

Function

The exclamation point is the SPEED prompt. You can type in and display a command line only when you have a prompt. SPEED restores your prompt when it finishes executing your previous command line. Until it does so, it displays the cursor alone on a visual display terminal.

SPEED displays the prompt below any text display. Remember that SPEED performs its operations at the position of the Character Pointer, and not at the position of the prompt.

Example

!!Sincerely,\$\$

*Sincerely,**

!

SPEED places the Character Pointer at the end of the insertion and restores your prompt below.

!label!*Symbol***Skip past this material.**

Format

!label!

Function

SPEED ignores material placed within exclamation points, except when you use them to pass command execution control with an O command. See the O command entry for details.

Use a label to write descriptive or explanatory comments into a command line, to remind yourself how the command line works or what it does.

Example

You write a note to yourself about the purpose of a command line, and store it with the command line.

```
!!This command line creates a ruled page!  
!60<80<I_$>I!  
$>$$
```

After SPEED carries out the command in the current buffer, you store the command line *with the attached comment* in Buffer R. Then you switch to Buffer R to verify what you have done.

```
!_R$$  
!BSR$#T$$
```

```
!This command line creates a ruled page!  
60 < 80 < I_$ > I!  
$ >
```

Now the label reminds you of what the commands in Buffer R do. Later, you may reissue the command line with a]BR command from another buffer, or you may place the contents of Buffer R in a file. Whenever you open or type out the file, the label will remind you of its contents.

n''Xcommand-string'*Command***Conditional Execution: carry out commands if the numerical argument has a certain value.**

Formats

```
n''Gcommand-string'  
n''Lcommand-string'  
n''Ecommand-string'  
n''Ncommand-string'
```

Function

This command allows you to set conditions for the execution of a sequence of commands, **command-string**. SPEED executes the commands when the condition you set holds true. When it is false, or becomes false, control passes to the command following the apostrophe.

Numerical Arguments

This command requires a single numerical argument, positive, zero, or negative. The effect of the argument depends on the specific form of the command you use:

Read	as
n''Gx'y	If <i>n</i> is <i>greater than</i> 0, do <i>x</i> ; otherwise do <i>y</i> .
n''Lx'y	If <i>n</i> is <i>less than</i> 0, do <i>x</i> ; otherwise do <i>y</i> .
n''Ex'y	If <i>n</i> is <i>equal</i> to 0, do <i>x</i> ; otherwise, do <i>y</i> .
n''Nx'y	If <i>n</i> is <i>not equal</i> to 0, do <i>x</i> ; otherwise do <i>y</i> .

For *n* you may use integers, numerical expressions, and values of variables and pseudo-variables. To use the full power of this command, however, you may set *n* to the increment or decrement of a variable or pseudo-variable, and use it to return to a previously set label.

To take an example, suppose that you want to number only the lines on your page that are not already numbered. One way to do this uses two Conditional Executions within a Command Loop. Issue

```
!JVN<48-VC''GVL\I.—$T'  
VC-58''GVL\I.—$T'  
1L>$$
```


You jump to the beginning of the buffer and set a Command Loop to apply as many times as there are lines in the buffer. The ASCII digital values for the digits 0 through 9 are 48 through 58, respectively. Assume the text itself contains no numbers at the beginning of lines. You want to insert a line number (with accompanying period and tab) just in case the line does not already start with a digit. This means that the ASCII digital value of the first character of the line must be less than 48 (which is the character 0) or more than 58 (which is the character 9).

- The first Conditional Execution takes care of the first case. It reads: if 48 less the ASCII digital value of the next character is greater than zero (that is, the value of the next character is in the range zero through 47), then carry out the execution: insert the line number, a period, and a tab, and display the result. The first condition will be true only if the line begins with a control character or certain marks of punctuation. If the line begins with `!N`, for example, `SPEED` will insert the line number.
- The second Conditional Execution takes care of the second case. It reads: if the ASCII digital value of the next character, less 58, is greater than zero (that is, the value of the next character is 59 or more), then carry out the (same) execution: insert the line number, a period, and a tab, and display the result. The second condition will be true if the line begins with an alphabetic character or certain other marks of punctuation. If the line begins with an `A`, for example, `SPEED` will insert the line number.
- If the line begins with one of the digits, then neither condition will be true, and `SPEED` will not insert a line number.

Whichever of the three is the case, control now passes beyond the conditions to the next command, which takes the CP to the beginning of the next line, if there is one. The Loop then reapplies on that line. If your buffer contains just

```
1.      Alpha
Beta
Gamma
4.      Delta
Epsilon
```

the Command Loop and two Conditional Executions above convert it to

```
1.      Alpha
2.      Beta
3.      Gamma
4.      Delta
5.      Epsilon
```

Symbolic Modifiers

None

Characteristics

This command does not require a delimiter. Be careful, however, not to omit the delimiter for the `O` command if you issue one.

You must *mispair* punctuation marks for this command: the *quotation marks* on the left accompany the *apostrophe* on the right.

You may use `O` (Over) commands and labels to branch into or out of Conditional Executions.

Precautions and Error Messages

Remember to set up your variables in such a way that the Conditional Execution eventually terminates. If, for example, you set `V9` to 5, the command string

```
!!write!lx$VD9''GOwrite$'$'$
```

terminates execution after the fifth insertion, since `V9` is no longer greater than zero at that point. But the command string

```
!!write!lx$VI9''GOwrite$'$'$
```

attempts to make the insertion indefinitely, since each increment of `V9` leaves it still greater than zero. (Execution stops when the value of `V9` exceeds the argument range for the command; to regain the console beforehand, use a `CTRL-C`, `CTRL-A` interrupt.)

Do not try to overlap Command Loops and Conditional Executions. If you try to issue a command line such as

```
!<S□$;VD5''N-1D>1T'$'$
```

n''Xcommand-string' (continued)

SPEED displays the message

Error: < with no corresponding >

If you fail to provide a numerical argument for this command, SPEED displays the message

Error: Illegal number of arguments to command

Related Commands

At your option, you may set a label in the command line, and place an **Olabel** command within or following the command string of the Conditional Execution. The effect of this maneuver is to set conditions for returning or not returning to a label, and executing the commands that follow it. If you have not yet consulted the **Olabel** command, please do so.

To execute a command line a given number of times without attaching other conditions, use the **Command Loop**; see the **<command-string>** command.

Examples

You are writing some news copy.

```
!Z-.''EI***more***$'LT$$
```

If your CP is at the end of the buffer, subtraction of the CP position from the end of the buffer will yield 0. Since the condition is for the value to equal 0, SPEED will make the insertion, return to the beginning of the line, and display the line. If your CP is not at the end of the buffer, SPEED will merely display the line the CP is on.

You wish to insert a decremting series of numbers from 29 to 20.

```
!10VSO  
!vig!19+V0\  
VD0''GI□$Ovig'  
L$$
```

```
*29 28 27 26 25 24 23 22 21 20
```

You set a variable to 10, set a label, and insert the first value of the sum of 19 and the variable. The variable decrements, and Command Execution keeps inserting a space and returning you to the label until the variable reaches 0, at which point SPEED returns you to the beginning of the line.

You wish to convert all lowercase text in a file to uppercase, but leave it otherwise unchanged. You issue

```
!!up!  
123-VC''GVC-96''GVC-32!1DOup$''  
VC''EOend$'  
1MOup$  
!end!$$
```

The lowercase alphabet is ASCII 97₁₀ through 122₁₀. An uppercase character has the value of the corresponding lowercase less 32. The first, outer Conditional Execution is valid if the value of the next character is not greater than z. The second, inner Conditional Execution is valid if the value of the same character is not less than a. So both Conditional Executions are valid only if the next character is lowercase alphabetic. If it is, the inner Conditional Execution inserts the corresponding uppercase character, deletes the lowercase, and returns you to the **!up!** label. If either condition is false, then either the next character has a value higher than z or lower than a (in which case you want to skip over it), or there is no next character and you are at the end of the buffer. If you are at the end of the buffer, then VC = 0, and the third Conditional Execution takes you to the **!end!** label. If you are not at the end of the buffer, the third Conditional Execution is not valid. In that case, you move forward one character (which you want to skip over), and return again to the **!up!** label to begin checking the value of the next character. The commands will convert

Please insert 28 characters.

to

PLEASE INSERT 28 CHARACTERS.

changing lowercase to uppercase without disturbing numbers, uppercase, or punctuation.

Symbol
Entire contents of the buffer

Format

#command

Function

The number sign (#) in SPEED commands abbreviates the paired numerical argument (0,Z) and thus represents the entire contents, in characters, of the current buffer. Use the symbol only with commands which take paired arguments.

Examples

#K\$\$	Kill the entire buffer.
#Sstring\$\$	Search from the beginning of the buffer.
#T	Display the entire buffer (the CP does not move).

& *Symbol*
Switch to the alternate radix.

Formats

n&command-name
&ncommand-name
&n&command-name

Function

In addition to the standard decimal radix, SPEED permits you to select an alternate radix from base 2 (binary) to base 36. See the WR command. If you specify no alternate radix, your default alternate radix is base 8 (octal).

You use the ampersand to switch from decimal to the current alternate radix for the next command. An ampersand before the command name interprets the result of the command in the alternate radix. Ampersands before numerical arguments interpret those arguments in the alternate radix.

Remember that the digits following ampersands must be valid expressions in the alternate radix. For example, 8 and 9 are not valid octal digits, and SPEED sends you the error message

Error: Illegal command

if your alternate radix is octal and you insert them after the ampersand.

Example

$400 + 20 = 420$	$400 * 20 = 8000$	$400 / 20 = 20$
$\&400 + \&20 = 272$	$\&400 * \&20 = 4096$	$\&400 / \&20 = 16$
$400 + 20\& = 644$	$400 * 20\& = 17500$	$400 / 20\& = 24$
$\&400 + \&20\& = 420$	$\&400 * \&20\& = 10000$	$\&400 / \&20\& = 20$

*

Symbol

Multiplication Operator

Format

m*n

Function

SPEED uses the asterisk to represent multiplication. Do not try to follow this operator with another, for example a minus. If you do, SPEED displays the message

Error: Syntax error

Remember that SPEED is restricted to the range -32768_{10} through $+32767_{10}$ for commands that take both positive and negative numerical arguments, and to the range 0 through $+65535_{10}$ for commands that take positive or paired numerical arguments.

Examples

You do some trivial arithmetic.

```
!13*13=$$
```

```
169
```

You are setting up a table for later use.

```
!:Soutflow$*3<—Week:$>$$
```

```
Outflow      Week:Week:Week:*
```

Using the colon modifier on the search command, you tell SPEED to make an insertion three times if the search command is successful. Since the search command returns a 1 if it succeeds, you specify the insertion in terms of a multiplication on the numerical argument of the insertion. The insertion takes place because $1*3=3$. If the search had failed, SPEED would not have made the insertion since $0*3=0$.

*

Symbol

A blinking asterisk shows the position of the Character Pointer.

Format

* (blinking)

Function

SPEED represents the Character Pointer with a blinking asterisk on terminal models 6052, 6053, D100, and D200. On other models, SPEED represents the Character Pointer with a caret within parentheses (^).

The CP lies between two characters, and not on a character. The various T commands show you the location of the CP in the buffer. For other ways of ascertaining its position, see the Z and . symbols, and the commands VC, VL, VM, and VP.

You can exercise some control over the positioning characteristics of the Character Pointer. See the WP command and the CTRL-G template.

Example

```
!Sre]G-issue$1D$$
```

```
re*issue
```

You use CTRL-G to adjust the position of the CP when the search and deletion take place. Otherwise, SPEED would place the CP at the end of the word.

+

Symbol

Addition Operator.

Format

$m+n$

Function

SPEED uses the usual symbol to represent addition. Do not try to follow this operator with another one, for example a minus. If you do, SPEED displays the message

Error: Syntax error

Remember that SPEED is restricted to the range -32768_{10} through $+32767_{10}$ for commands that take both positive and negative numerical arguments, and to the range 0 through $+65535_{10}$ for commands that take positive or paired numerical arguments.

Examples

You do some trivial arithmetic.

```
!34+35=$$
```

69

You tell SPEED to increment Variable 4 by 15 and display the result.

```
!15+V4VS4=$$
```

25

You tell SPEED to insert an asterisk 7 plus the value of Variable 3 times.

```
!V3+7<I*$>$$
```

```
*****
```

SPEED inserts 13 asterisks, since the current value of the variable is 6.

-

Symbol

Unary minus operator.

Format

$-n$

Function

You perform subtraction with this operator in SPEED. Remember that SPEED expresses negative numbers in descending order from 65536.

Remember that SPEED is restricted to the range -32768_{10} through $+32767_{10}$ for commands that take both positive and negative numerical arguments, and to the range 0 through $+65535_{10}$ for commands that take positive or paired numerical arguments.

Do not try to follow another arithmetic operator with the minus operator. If you try, for example, the expression $5*-3=$, SPEED displays the message

Error: Syntax error

Examples

You do some trivial arithmetic.

```
!256-16=$$
```

240

You ask SPEED how far you are from the end of the buffer (in numbers of characters).

```
!Z-=$$
```

17325

You do more trivial arithmetic and get SPEED's representation of the number -132.

```
!12-144=$$
```

65404

	<i>Symbol</i>
Current position of the Character Pointer	

Format

Function

SPEED uses the period (.) to represent the position of the Character Pointer in terms of characters from the beginning of the current buffer. You may use this symbol as a numerical argument to the next command.

Examples

- .D\$\$ Delete as many characters forward as there are between the beginning of the buffer and the CP.
- .D\$\$ Delete all characters in the buffer preceding the CP.
- .I\$\$ Insert the ASCII character whose decimal equivalent is the position of the CP, truncated to the last eight bits.
- O,.K\$\$ Kill the buffer up to the CP.
- .M\$\$ Double the distance of the CP from the beginning of the buffer.
- O,.T\$\$ Display the buffer from the beginning up to the position of the CP (= -ZT).
- .\\$\$ Insert the value of the position of the CP.
- .=\$\$ Tell me the value of the CP.
- Z-=\$\$ Tell me how many characters following the CP there are in the current buffer.
- .<x>\$\$ Do the Command Loop x a number of times equal to the value of the position of the CP.

	<i>Symbol</i>
/	Division Operator

Format

m/n

Function

SPEED uses the slash to represent division. Do not try to follow this operator with another, for example minus. If you do, SPEED displays the message

Error: Syntax error

Remember that SPEED is restricted to the range -32768_{10} through $+32767_{10}$ for commands that take both positive and negative numerical arguments, and to the range 0 through $+65535_{10}$ for commands that take positive or paired numerical arguments.

SPEED does only integer or whole number arithmetic. SPEED treats proper fractions as 0, and truncates improper fractions. SPEED treats $n/0$ as n . SPEED does not recognize decimal expressions and handles them as syntax errors.

Examples

You do some trivial arithmetic.

!256/5=\$\$

5/

You ask SPEED what your average line length is by telling SPEED to display the result of dividing the number of characters in the buffer by the number of lines in the buffer.

!Z/VN=\$\$

2/

:

Symbol

Multipurpose Modifier

Format

:command-name

Functions

The effect of this modifier varies with the class of the command name you use it with.

- In the :BG (Buffer Get) command, the colon suppresses the query or ? prompt and you may type the line in directly.
- In the C, N, Q, R, and S search commands, the colon returns a 1 if the command succeeds and a 0 if the command fails. This value can serve to specify the number of times the next command executes if that command takes a numerical argument, and if you specify an arithmetic operator such as + or * connecting the two numbers.

The command :Sq.e.d.\$*3<!!\$>\$\$, for example, will insert three exclamation points only if it finds *q.e.d.*, since $3*1=1$ and $3*0=0$.

Inclusion of a search command in an iteration automatically simulates the : colon modifier.

- In the Put commands :P and :PW, SPEED clears the buffer after writing to the output file. If you omit the colon, SPEED does not clear the buffer.
- In the Case Control :WC command, the colon extends case control to material brought into the buffer from files and other buffers with the CTRL-F and CTRL-B templates. SPEED first converts letters to uppercase and then converts them as the shift and shift-lock characters dictate. SPEED returns a 1 if you extend case control this way, and a 0 otherwise.

- In the Execute :X command, you create a son of SPEED that is a copy of the program you used to enter SPEED. This is usually the CLI, although it may be another SPEED process or some other program.

- In the Conditional Termination ; command, the colon reverses the conditions for terminating command execution. In a Command Loop, ; terminates execution if the previous command *succeeds*, and passes control to the next lower level of the command line. Outside of a Command Loop, ; terminates execution of the command line if the previous command succeeds or its numerical argument is *positive*.

Examples

xxx

!<Sx\$;lo\$>\$\$

xoxoxo

!<Sxo\$;lo\$>\$\$

*xo*xoxo*

The first Command Loop executes until it cannot find another *x*. The second one halts as soon as it finds the first *xo*, and the *l* command never executes.

For additional examples of the uses of this modifier, please see the individual commands that it modifies.

Conditional Termination: terminate execution of a command line or Command Loop upon the failure or success of the last command.

Formats

```
;
n;
;;
n::
```

Function

This command prevents infinite looping in a Command Loop when you do not wish to specify execution of the Command Loop a definite number of times by prefixing the brackets of the Loop with a numerical argument.

This command, when *unmodified* by the colon symbol (:), terminates the execution of a Command Loop within which it occurs if the previous command fails, and turns control over to the next higher level of the command line.

Outside of a Command Loop, this command terminates execution of the command line if the previous command fails or if its numerical argument is 0 or negative. If you have not yet consulted the Command Loop in this dictionary, please do so; see the <command-line> command.

Numerical Arguments

This command takes single positive or negative numerical arguments.

The *n*; command

- terminates execution of a command line if outside a Command Loop
- passes control to the next higher level if within a Command Loop

when

- *n* is 0 or negative and
- the colon symbol (:) does *not* precede the command

This command accepts as arguments integers, more complex numerical expressions, variables and pseudo-variables, and functions which you define on these.

Symbolic Modifier

This command accepts the colon modifier (:). This modifier reverses the effect of the ; command

The :: command

- passes execution within a Command Loop to the next lower level
- terminates execution of the command line outside of a Command Loop

when the previous command *succeeds*. The *n*:: command

- terminates execution of a command line if outside a Command Loop
- passes control to the next higher level if within a Command Loop

when *n* is greater than 0.

SPEED treats an unsuccessful search outside of a Command Loop as an error condition, regardless of whether a semicolon follows the search command. SPEED displays the message

Error: Unsuccessful search

Related Commands

Please consult the <command-line> Command Loop command for its interaction with this command.

You may conditionally execute search commands and certain other commands by modifying them with the colon modifier (:) under certain conditions. See the colon symbol and the entries for the A, C, N, Q, R, and S commands.

; (continued)

Examples

You decide to count periods to get a rough estimate of the number of sentences in the buffer.

```
!JOVS0$<S.$;VIO>VO=$$
```

93

You jump to the beginning of the buffer, set Variable 0 to 0, and issue a Command Loop. The Loop searches for a period; each time it finds one, it increments Variable 0. After it finds the last period in the text, it makes the search once again. When it fails to find a period, the ; command takes you out of the Loop and on to the next command, which displays the number of periods found. If you had not included the semicolon after the search command delimiter, the Command Loop would have begun cycling endlessly.

On another occasion, you can't remember whether you inserted a percent sign at a certain place in the text.

```
!OWP0,.S%$;T$$
```

If you did, you want to see the surrounding text; if you didn't, you set your position mode so that you can continue editing from the current CP position. You use the colon modifier to reverse the usual effect of the ; command.

< x >

Command

Command Loop: execute the command line enclosed in brackets a number of times.

Formats

<command-line>
n<command-line>

Function

This command allows you to execute the command line it controls, *command-line*, a number of times without having to re-enter the commands.

You may specify a definite number of times for the Command Loop to execute, or you may set a condition in the Loop with a Conditional Termination ; command that terminates execution of the Loop and passes control to the next command.

If you have not yet consulted Conditional Termination, please do so; see the ; command.

Numerical Argument

This command accepts positive numerical arguments. The n<x> form of the command specifies that the command line *x* execute *n* times. If *n* is not greater than 0, SPEED skips over the Loop to the command following it.

If you issue the command *without* a numerical argument, it begins cycling endlessly unless the command line within it contains appropriately placed Conditional Termination commands. See the ; command.

Characteristics

This command does not require a delimiter. Be careful, however, to include delimiters for commands requiring them *within* the angled brackets. An insertion loop, for example, should have the form

```
n<Istring$>$$.
```

Inclusion of a search command in an iteration automatically simulates the : colon modifier.

You may embed one Command Loop within another to a depth of ten.

You may embed a Conditional Execution within a Command Loop, and you may use Command Loops in a command line containing O commands and Conditional Executions. Figure 7-3 summarizes some of the possible interactions.

<x> (continued)

Precautions and Error Messages

If you omit the final delimiter of a search or insertion command within the brackets, SPEED displays the message

Error: < with no corresponding >

and treats the right bracket as part of the insertion.

If you have not entered a numerical argument for the Command Loop, and

- the Loop contains search commands (C, N, Q, R, S) which *also* lack arguments, you can prevent endless cycling by following each such search command with a Conditional Execution (;) command.
- the Loop contains no search commands, or only search commands without a following Conditional Termination command, you may have to break the cycling with a CTRL-C, CTRL-A interrupt.

If you exceed the embedding depth of ten, SPEED displays the message

Error: Maximum iteration level exceeded

Even within the legal limits, you may find that deeply embedded Command Loops exhaust available memory.

Do not try to overlap Command Loops and Conditional Executions. You get the error message

Error: < with no corresponding >

for example, if you try to issue a command line such as

```
!<S□$;VD5''GT>VL='$$
```

Do not attempt to use an O command to branch into a Command Loop. SPEED displays the message

Error: Illegal Command

Related Commands

To specify conditions for the execution of a command line, use instead a Conditional Execution; see the n''Xcommand-line' command.

To specify a command line in which you automatically return to earlier commands or skip to later ones, see the Olabel command.

Examples

You decide that you want every line in the buffer inset.

```
!JVN<—$1L>$$
```

You jump to the beginning of the buffer and issue the Command Loop with the numerical argument VN+1 (recalling that the VN command counts the first line as 0). SPEED inserts a tab and moves down one line as many times as the argument indicates.

You wish to change the informal tone of a letter you are writing.

```
!<CBob$Robert$;T>$$
```

With the Command Loop you change every instance of Bob to Robert from the current CP position to the bottom of the buffer. Since you issue no numerical argument to the Loop, you include the semicolon of the Conditional Termination command. When the C command fails to find a match, control passes out of the Loop and the Loop stops cycling.

=

Command

Equals: display or type out the value of the numeric argument *n*.

Format

n=

Function

SPEED can perform simple arithmetic and Boolean calculations, and display the result. The = command displays on the screen or types out at the hard-copy terminal the value of the numeric argument *n*, where *n* may be any numeric argument or expression, the value of a variable or pseudo-variable or any function you can define on it, or any Boolean expression.

Numerical Arguments

None

Symbolic Modifiers

This command takes the commercial at (@) and ampersand (&) modifiers.

n@=

suppresses a line break between the value it displays and the next SPEED prompt. That is, the response to

VN=\$\$

is

30
!_

but the response to

VN@=\$\$

is

*30!*_

If you issue a series of = commands in a single command line, the system response is on the same line for the command following a @= command, and on the next line for the command following a = command.

n&=\$\$

expresses the numerical value in the alternate radix. If, for example, you set your alternate radix as binary with a 2WR command, the system responds to

10 + 10&=\$\$

with

10100

For details concerning numerical and Boolean expressions and operations, see the WR command, the ampersand symbol (&) for taking the alternate radix, the addition (+), subtraction (-), multiplication (*), and division (/) symbols, and the symbols for Boolean operators (↑+, ↑-, ↑*, ↑/).

Characteristics

This command does not require a delimiter.

You may decrement, increment, or set variables and display the result with the same command; issue *VDn*=, *VIn*=, and *VSn*= commands.

The following = commands will help you keep track of your text and editing tasks:

<i>VC</i> =	What is the ASCII decimal equivalent of the next character on the right?
<i>VL</i> =	What line number am I on?
<i>VM</i> =	How many moves is it from the beginning of the line to the CP?
<i>VN</i> =	How many NEW LINES are there in the current buffer?
<i>VP</i> =	What was the previous position of the CP?
<i>WA</i> =	What is my default argument mode?
<i>WC</i> =	What is my case control mode?
<i>WD</i> =	How many lines does my terminal display at a time?
<i>WM</i> =	How many lines does SPEED read into the buffer at a time?
<i>WP</i> =	What is my mode for positioning the CP after an unsuccessful search?
<i>WR</i> =	What is my alternate radix?

= (continued)

- WS= What is my mode for matching case of letters during a search?
- Z= How many characters are there in the current buffer?
- . = How many characters are there from the beginning of the buffer to here?

See each of the commands or symbols used as an argument to the = command above for details of the value returned and displayed.

Precautions and Error Messages

Remember that SPEED is restricted to the range -32768_{10} through $+32767_{10}$ for commands that take both positive and negative numerical arguments, and to the range 0 through $+65535_{10}$ for commands that take positive or paired numerical arguments.

If you inadvertently issue this command with a nonnumerical argument, SPEED informs you that you have made an error in syntax or have issued an illegal command, or sends you the message

Error: Illegal number of arguments to command

Related Commands

To insert the system response in the text, use a Backslash; see the n\ command.

Example

!Z-. = \$\$

2001

You ask SPEED how far in terms of characters you are from the end of the buffer and SPEED tells you.

?

Command

Turn the Trace Mode Toggle on if off, or off if on.

Format

?

Function

This command turns Trace Mode on if it was off, and turns it off if it was on. When you turn Trace Mode on, SPEED displays the characters in the command line as they execute.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

This command does not require a delimiter.

When the toggle is on, SPEED displays each character in the command line, including NEW LINE, form feed, space, arguments to the command, and the first letter of each command. For brevity, the rest of the characters in the command string, such as a long text insert, do not echo. When SPEED has processed the entire command line, it resumes echoing each character.

When you turn the toggle on, you inhibit the automatic display mode, whether you set it with a nWD command or enter SPEED with the D switch. When you turn the toggle off by reissuing the ? command, SPEED will automatically restore your display mode. While the toggle is on, you have to issue T commands for display of text.

The default mode of the toggle is off.

Example

```
?<S;$;VIO>lcolons□ = □$V0\T$$
```

```
< S;VIO > S;VIO > S;VIO > S;VIO > S;IVO \
Tcolons = 4$$
!
```

You turn the Trace Mode Toggle on to observe the execution of the following commands, which search for colons and insert the result in the text. With the toggle on, SPEED gives you a display of the iterative execution of the Command Loop, the two insertion commands, and the display command. It follows that with the text display and then restores your prompt.

@

Symbol

Set a temporary delimiter.

Format

@command-name%

Function

Use this modifier to set a temporary delimiter when you wish to search for or insert the standard delimiter ESC (\$) in the text, and therefore do not want to use ESC as the search string or text string delimiter. SPEED takes the first character following the command name (arbitrarily shown as a percent sign (%) in the format) as the temporary delimiter. The character has the status of a delimiter for the current command only. If a text string or search string ends your command line, remember to insert the temporary delimiter before you type in CTRL-D. Otherwise SPEED displays the message

Error: Unterminated string

and place a CTRL-D echo (\$\$) in the text.

The following commands take the commercial at modifier:

```
□CINQST =
```

The @= command has the idiosyncratic effect of suppressing the NEW LINE after the displayed value. The next character following @T is the temporary delimiter for the following display; if you do not select a temporary delimiter, use ESC (@T\$message\$). See the section "Symbolic Modifiers" under each separate command for other details of the behavior of @. See also CTRL-I for the use of tabs as temporary delimiters in conjunction with @.

Example

```
< Canologue$analog; >
```

```
!@C/;/$/$$
```

```
< Canologue$analog$; >
```

You have typed and stored a faulty command line. You use the commercial at (@) modifier in order to implement the slash as a temporary delimiter. This enables you to change the semicolon to an ESC, semicolon (\$;) sequence. You enter the slash temporary delimiter both at the end of the search string and at the end of the text string.

Backslash: insert a string of ASCII digits into the text.

Format

$n\backslash$

Function

This command inserts the ASCII value of the expression n into the text, where n is a Boolean expression, a numerical expression, the value of a variable or pseudo-variable, or a function you define on these.

Use this command to insert available numerical values in the text automatically rather than calculating them by hand.

Numerical Arguments

None

Symbolic Modifier

This command takes the ampersand modifier (&). The ampersand modifier, when preceding the command name, inserts in the text the numerical value in the alternate radix. If, for example, you set your alternate radix as binary with a 2WR command, the system will respond to

```
!10+10&\$$
```

with

```
10100
```

For details concerning numerical and Boolean expressions and operations, see the WR command, the ampersand symbol (&) for taking the alternate radix, the addition (+), subtraction (-), multiplication (*), and division (/) symbols, and the Boolean operators (^+ ^- ^* ^/).

When you issue the command without the ampersand modifier, SPEED inserts the decimal value of the expression.

Characteristics

This command does not require a delimiter.

This command inserts into the text the *value* of a numerical expression, not the numerical expression itself. If, for example, you issue a $VDO*69\backslash\$\$$

command, where $V0=22$, SPEED will insert 1449 in the text, which is the result of decrementing $V0$ by one and multiplying the result by 69.

Precautions and Error Messages

Remember that SPEED is restricted to the range -32768_{10} through $+32767_{10}$ for commands that take both positive and negative numerical arguments, and to the range 0 through $+65535_{10}$ for commands that take positive or paired numerical arguments.

Remember that SPEED truncates to the last eight bits.

If you inadvertently issue this command with a nonnumerical argument, SPEED informs you that you have made an error in syntax, or that you have issued an illegal command, or displays the message

Error: Illegal number of arguments to command

Related Commands

The \backslash command inserts the system response into the text. To perform calculations *without* inserting the result in the text, use the = command.

To insert the *character* corresponding to an ASCII numerical value, see the nl command.

Examples

You want to record the length of your file at the head of the file.

```
J!This file contains $VN\| lines.$$
```

This file contains 90 lines.

You do so by delimiting the first I command, issuing the VN\ command, and continuing with another I command.

You are taking inventory of stock on hand.

```
!572 cases of 24 cans each = $572*24\| cans.$$
```

*572 cases of 24 cans each = 13728 cans.**

By using the / command, you let SPEED perform the needed calculation instead of doing it yourself by hand.

You wish to insert a series of numbers for use in a measuring scale.

```
!15VS0<VD0+1\I.□$>-1D$$
```

```
15. 14. 13. 12. 11. 10. 9. 8. 7. 6. 5. 4. 3. 2. 1.*
```

You set a variable for the highest value, and apply it as a numerical argument to the Command Loop. Since $VD0+1 = V0$, the first time around, the Command Loop inserts 15, a period, and a space. Successive executions insert decrements from 15. The command following the Loop removes an end space.

^ *

Symbol

Logical AND (Boolean Intersection)

Format

x^*y

Function

SPEED uses the sequence of a caret and an asterisk to represent logical AND or Boolean intersection. The function is defined as

$x^*y = 1$ if $x = 1$ and $y = 1$

$x^*y = 0$ otherwise

This binary operator produces a full word result with each bit representing the Boolean product of the corresponding bits of the operands.

Example

$1^*1 = 1$

$1^*0 = 0$

$0^*1 = 0$

$0^*0 = 0$

 $\wedge +$ *Symbol***Logical Inclusive OR (Boolean Union)**

Format $x \wedge + y$ **Function**

SPEED uses the sequence of a caret and the plus symbol to represent logical inclusive OR or Boolean union. The function is defined as

$$\begin{aligned} x \wedge + y &= 0 \text{ if } x=0 \text{ and } y=0 \\ x \wedge + y &= 1 \text{ otherwise} \end{aligned}$$

This binary operator produces a full word result with each bit representing the Boolean sum of the corresponding bits of the operands.

Example
$$\begin{aligned} 1 \wedge + 1 &= 1 \\ 1 \wedge + 0 &= 1 \\ 0 \wedge + 1 &= 1 \\ 0 \wedge + 0 &= 0 \end{aligned}$$

 $\wedge -$ *Symbol***Logical NOT (Boolean Complement)**

Format $\wedge -x$ **Function**

SPEED uses the sequence of a caret and a hyphen to represent logical NOT or the Boolean complement. The function is defined as

$$\begin{aligned} \wedge -x &= 0 \text{ if } x=1 \\ \wedge -x &= 1 \text{ if } x=0 \end{aligned}$$

This unary operator precedes its operand, just as the unary minus does. The operator produces a full word result with each bit representing the Boolean complement of the corresponding bit in the operand.

Remember that SPEED is restricted to the range -32768_{10} through $+32767_{10}$ for commands that take both positive and negative numerical arguments, and to the range 0 through $+65535_{10}$ for commands that take positive or paired numerical arguments.

Example
$$\begin{aligned} \wedge -1 \& = 1111111111111110 \\ \wedge -0 \& = 1111111111111111 \end{aligned}$$

For the example the alternate radix is set to 2 and the ampersand modifier produces a binary result. Otherwise the result would be

$$\begin{aligned} \wedge -1 &= 65534 \\ \wedge -0 &= 65535 \end{aligned}$$

See the WR command and the ampersand symbol.

^ /	<i>Symbol</i>
Logical Exclusive OR (Boolean Symmetric Difference)	(Boolean Symmetric Difference)

Format

$x \wedge / y$

Function

SPEED uses the sequence of a caret and a slash to represent logical exclusive OR or Boolean symmetric difference. The function is defined as

$$\begin{aligned} x \wedge / y &= 0 \text{ if } x = y \\ x \wedge / y &= 1 \text{ otherwise} \end{aligned}$$

This binary operator produces a full word result with each bit representing the symmetric difference of the corresponding bits of the operands.

Example

$1 \wedge / 1 = 0$ $1 \wedge / 0 = 1$ $0 \wedge / 1 = 1$ $0 \wedge / 0 = 0$

_x	<i>Command</i>
Save the previous command line in a buffer.	

Format

_buffer-name

Function

The only legal buffer names are the digits 0 through 9 and single letters of the alphabet. This command places the last command line longer than 10 characters in the buffer denoted by **buffer-name**. This frees the space the command line occupies for subsequent command strings, but stores the command line for editing or reissue with a **[Bbuffer-name** command.

Numerical Arguments

None

Symbolic Modifiers

None

Characteristics

This command does not require a delimiter, but it must be the first command after the prompt.

The command string terminator, CTRL-D, which echoes as \$\$, counts as two characters for purposes of this command.

If there is no previous command string 10 characters or longer, or if you saved the last such string with a **_x** command, SPEED places the last command line you issued into Buffer *x*.

Precautions and Error Messages

You may not execute a **_x** command from a buffer or from a file, that is, as part of a match to a **[Bbuffer-name** or **[Ffilename** template, or as part of a command file serving as a value to the **/I=** (Invoke) switch. SPEED displays the message

Error: Illegal command

If memory space becomes scarce, SPEED saves the space occupied by the command string you are trying to store in order to avoid a memory space exhaustion error. The effect is the same as if there were no previous commands.

_x (continued)

Related Commands

You may reissue a command stored in Buffer *x* with the `_x` command by issuing a `↑Bx` command. See the CTRL-B template in this dictionary.

If you do not wish to save a command line, but know before you issue it that you want a display of its execution, use the Trace Toggle. See the `?` command.

Example

SPEED surprises you unpleasantly when it executes the command line below. You had decided to get a count of periods as a rough count of the number of sentences you had in your text. The result can't be right, so you must have issued the wrong commands.

```
!JOVS0 <S.$;V10> Isentences □ = □ $V0 \ $$
```

```
sentences = 0*
```

```
!_ 1 $BS1 $$
```

You save the command line responsible for the error in Buffer 1 in order to study it carefully, edit it, and reuse it. When you switch to Buffer 1, you see

```
JOVS0 <S.$;V10> Isentences = $v0 \ $$
```

and immediately realize that by omitting the delimiter, you inadvertently set a numerical argument of zero for the Command Loop. You correct that with

```
!@C / < / $ < / $$
```

Next you return to your home buffer and re-execute the command line by issuing a `↑B1` command:

```
BS0 $↑B1 $$
```

```
sentences = 76
```

Success at last!

CTRL- \ character-list CTRL- \ *Template*
Match the first instance of any character from the list between the backslashes.

Format

```
search↑\character-list↑\string
```

Function

When you issue a search command with this template, SPEED matches the search string `a↑\pqr↑\z` with whichever of *apz*, *aqz*, or *arz* it encounters first.

Do not try to place a list *within* a list. If you do, you will inadvertently truncate your lists. SPEED rejects the control characters `↑ E`, `↑ G`, `↑ N`, `↑ T`, `↑ X`, `↑ Y`, and `↑ Z` within this template, and sends you the message

Error: Illegal control character in search string

To place any control character in a `↑\...↑\` list other than `↑W`, precede it with `↑W`. SPEED allows buffer names and filenames within the backslashes (for filenames, remember to issue the command with the commercial at modifier (`@`) in order to end the filename with a delimiter). SPEED will treat each character in the buffer or file as one of the search alternatives.

`↑ N ↑\ pqr ↑\` will match any one character *except* those within the backslashes.

`↑ Y ↑\ 0123456789 ↑\` matches any string of digits. `↑ Y ↑\ aeiou ↑\` matches any sequence of one or more vowels.

Example

```
!OVS6$ <S↑\.?↑\ $;V16$> V6 = $$
```

You make a rough count of the number of sentences in the buffer by counting end punctuation. Using the backslash template, you conduct a single search rather than separate ones for each mark.

CTRL-Bbuffer-name *Expansion*
Expand to the entire contents of a buffer.

Formats

search]Bbuffer-namestring\$
]Bbuffer-name

Function

When you issue a search command with this expansion, SPEED tries to match]Bbuffer-name with the contents of that buffer. The buffer you name must be active, and it must not be the current buffer. The only legal buffer names are the digits 0 through 9 and individual letters of the alphabet.

If the buffer named by buffer-name contains text, you may insert that text into the current buffer with an]Bbuffer-name command. For example, if Buffer A contains the text

Do not read this sentence.

the command

!|Please remember:]BA\$-1L\$\$

will insert

**Please remember: Do not read this sentence.*

in the current buffer.

You do not need an ESC (\$) to close]BA. Unlike the CTRL-F expansion, SPEED treats material following a CTRL-B expansion and a single delimiter as a sequence of commands. (In the example, SPEED positions the CP at the beginning of the line, rather than treating L as further text to insert.)

If the buffer contains a numerical value, you may use CTRL-B as part of a numerical expression. For example, if Buffer 5 contains the value 69, the command]B5-2L\$\$ inserts the letter C in the current buffer, since the ASCII decimal equivalent of C is 69-2=67. Similarly, the command 2*]B5\\$\$ inserts the value 138 in the text of the current buffer.

If the buffer contains commands, you may *execute* the buffer by issuing the expansion]Bbuffer-name\$\$ as if it were a command line. For example, if Buffer C contains

!JOVS0\$<S]\□
]\\$.VIO>VO=\$\$

the simple command]BC\$\$ carries out those commands for the current buffer, giving a rough estimate of the number of words in the buffer by counting spaces and NEW LINES and displaying the value.

You can nest the commands inserted using the CTRL-B and CTRL-F expansions up to 10 levels deep.

To *mention* a CTRL-B expansion when you are constructing a command line that you wish to store and execute at a later date, prefix CTRL-B with CTRL-F or another CTRL-B. If you do not type]B]Bbuffer-name or]F]Bbuffer-name, SPEED attempts to insert or execute the contents of the buffer in the usual way.

Examples

You want to append the contents of file ADDENDA to the end of file AGENDA.

!FOagenda\$BS5BFOaddenda\$
 BCABFUBSOZJI]BA\$\$

You open AGENDA globally for updating. You switch to an available buffer, Buffer 5, and open ADDENDA locally for updating. You copy the contents of ADDENDA to another buffer, Buffer A, and close the local file ADDENDA. You switch back to your home buffer and take the CP to the bottom of the buffer. You use the I command with the CTRL-B template to insert the contents of Buffer A into the current buffer.

You want a scheme for automatically typing line numbers and numbers of characters from time to time.

!VL\I.□\$. \→\$\$

70. 2334 *

!_N\$\$

.

.

!]BN\$\$

80. 2375 *

You execute the command line once, and then realize that you can use it repeatedly. You save the command line by putting it in Buffer N with the very next command. Later on in the session, you execute the buffer whenever you need that command line, rather than retyping the command line itself.

CTRL-C, CTRL-A *Console Control*
Cancel a current command line longer than one line.

Format

↑C↑A

Function

This control key sequence allows you to cancel a command line that extends over several lines, which contains NEW LINES or carriage returns, before you issue a CTRL-D.

To cancel a command on a single line, see CTRL-U.

Example

```
!!Heres□hte□list;)  
Smih)  
jones)  
Brooown↑C↑A
```

!

It is too much trouble clearing up the errors you have already made in this command line; you cancel it in order to start over.

CTRL-C, CTRL-B *Console Control*
Abort this SPEED editing session.

Format

↑C↑B

Function

This control key sequence allows you to abort the current editing session. You return to the parent process, usually the CLI. You lose all editing changes you made in the current session. If you opened the file with Update Mode on, you retain the earlier version of the file and an empty filename.TM. If you opened an output file with Update Mode off (with a BFW, BFNW, FW, or FNW command), the output file will be empty *even if you write to the file*, unless you close it before you issue the abort sequence.

If you are considering an abort because the terminal does not seem to be functioning, you may have inadvertently frozen the terminal with a CTRL-S. Try a CTRL-Q before issuing the abort sequence.

Cancel command lines with CTRL-U or CTRL-C, CTRL-A instead of using this control sequence.

To make an orderly exit from SPEED, use the H command.

Example

↑C↑B

You abort the SPEED editing session, losing all changes you made.

CTRL-D *Console Control*
Terminate and execute the current command line.

Format

command-line\$\$

Function

When you type this control key, SPEED terminates and attempts to execute the current command line. SPEED restores your prompt when it has finished the execution. CTRL-D also acts as a delimiter for a text string if the string is the last entry in the command line; you do not need to precede CTRL-D with ESC for a text string.

SPEED always echoes CTRL-D at the end of your command line with a double ESC (\$\$), which it counts as two characters, and never as ↑D. (To type in ↑D, issue a 4I command.) You cannot, by definition, place a CTRL-D *within* a text string.

Remember to terminate an X command with CTRL-D, rather than with the NEW LINE terminator of the parent process.

If you type a CTRL-D and nothing appears to happen, you may have inadvertently tapped CTRL-S to its left, which freezes the terminal. Try a CTRL-Q.

Example

```
!J$$  
!!Hello!$$  
!ZJ$$  
!!Goodbye!$$  
!#T$$
```

```
!JIHello!$ZJIGoodbye!$#T$$
```

The first command sequence contains five command lines. The second, single command line is equivalent and requires fewer strokes.

CTRL-E *Template*
Match one or more successive tabs and spaces in this position.

Format

search↑Estring

Function

When you issue a search command with this template, SPEED matches **pass**↑E**word** with any string consisting of *pass*, one or more tabs and spaces, and *word*.

This template will match any mixture of tabs and spaces.

↑N↑E matches one or more of anything that is *not* a tab or space.

↑X↑E matches 0 or more tabs and spaces, and is equivalent to ↑T.

↑Y↑E matches one or more tabs and spaces, and is equivalent to ↑E itself.

Example

```
!<C)  
↑E$!  
$;>$$
```

You remove all white space from the beginnings of lines with the ↑E template in a Command Loop.

CTRL-Ffilename

Expansion

Expand to the entire contents of a file.

Formats

`↑Ffilename$$ [command-line]`
`↑Ffilename$ [text-string$]`

where the double dollar sign (\$\$) represents a repeated escape < ESC,ESC > and not, as usual, CTRL-D

Function

When you issue a search command with this expansion, SPEED tries to match ↑Ffilename\$ with the contents of that file. You must have access to the file. If the file you wish to match is not in the current directory, you must specify a complete pathname.

If the file contains text, you may insert that text into the buffer with an ↑Ffilename\$ command. You must follow the filename with a delimiter. If you follow it with a single ESC (\$), SPEED treats the material following the filename as *additional text* and inserts it. If you wish the file to conclude the insertion, but you wish to continue the command line, you must follow the filename with the *double* delimiter ESC ESC (\$\$). SPEED then treats any material following the filename as a sequence of commands.

If the file contains commands, you may *execute* the file by issuing the expansion ↑Ffilename\$\$ as if it were a command line. When you use the expansion this way, SPEED treats the material following the filename and *single* delimiter as a command line.

You can nest the commands inserted using the CTRL-F and CTRL-B expansions in combination up to 10 levels deep. The pathname specified in a CTRL-F command, however, cannot contain an embedded CTRL-F command.

To *mention* a CTRL-F expansion when you are constructing a command line that you wish to store and execute at a later date, prefix CTRL-F with CTRL-B or another CTRL-F. If you do not type ↑F↑Ffilename or ↑B↑Ffilename, SPEED will attempt to insert or execute the contents of the file in the usual way.

Examples

You have stored a routine for numbering lines in the file LINE.

**The next sentence is false.
The next sentence is false.
The next sentence is false.
The first sentence is false.*

↑Fline\$J\$\$

- *1. The next sentence is false.*
- 2. The next sentence is false.*
- 3. The next sentence is false.*
- 4. The first sentence is true.*

Instead of laboriously numbering lines by hand, you tell SPEED to execute file LINE and then jump to the beginning of the buffer. SPEED produces the result you want.

The file DUM contains the first sentence below, and the file DEE contains the second.

↑Fdum\$↑Fdee\$\$-2L\$\$

**Please read the line below.
Please read the line above.*

The ↑ command first inserts the contents of file DUM. Since a single delimiter follows DUM in the command line, SPEED treats what follows as further text to insert. In this case, the additional text is the contents of file DEE. Since a double delimiter follows DEE in the text, SPEED treats the rest of the line as a command string and moves back two lines.

You are devising a command line which will, when you execute it, create a new file by inserting one file, PARADOX, into the buffer and applying the commands in another file, LINE, to it.

!@!%↑F↑Fparadox\$\$↑F↑Fline\$%\$\$
↑FWboth\$:PW\$FC\$\$

You *mention* rather than *use* CTRL-F by repeating it before the filename. You do not apply the commands at this time. Instead, you open a new file, BOTH, copy the command line to it, clear the buffer, and close the file. Now, whenever you wish to insert file PARADOX and execute file LINE, you will simply issue the command ↑Fboth\$.

CTRL-G *SPEED Control*
Put the CP in this position in the
matched string.

Format

search[Gstring

Function

When you issue a search command with this control key, SPEED matches `pass[Gword` with *password*. But instead of positioning the CP at the end of the matched string, as in *password**, SPEED places the CP in the position of the control key, in this case *pass*word*.

On models other than 6052, 6053, D100, or D200, use ↑ instead.

Examples

!`<Scon[Gieve$;Cie$ei$>$$`

*concei*ve*

You correct all occurrences of a common misspelling. The control key positions the CP so that it will be in the right position for the C command in the Command Loop. The example shows the CP after one such change has been made.

!`S[G]`
`$$`

You use the control key to position the CP at the end of the current line rather than at the beginning of the next line.

CTRL-I *SPEED Control*
Insert a string with an initial tab.

Format

][string\$

Function

This control key behaves like the I command, but inserts a tab before the string you specify. The CP repositions at the end of the inserted string.

This control key accepts the commercial at modifier (@). The effect of this modifier is to change the delimiter from \$ (ESC) to a tab, which you may type either with a second CTRL-I or with the TAB key. When you use the control key with the modifier, it does *not* insert a tab before the rest of the string.

Examples

You insert a tab before the title of your column.

!`][Series[A$$`

*Series A**

`< C3.1614$3.1416*; >`

!`@][I$][I$$`

`< C3.1614$3.1416$*; >`

You need to correct a faulty command line. You issue CTRL-I with the modifier in order to insert the missing delimiter. (In the command line, SPEED echoes ↑ I with a simulated tab.)

CTRL-J
NEW LINE

*SPEED Control***Format**

↑J

Function

This control key is equivalent to the NEW LINE key. SPEED always echoes CTRL-J with a new line, and never with ↑ J. In this manual we represent keying in a NEW LINE with ↵.

Example

```
!$$  
!↑J$$  
!0!$$
```

The three commands above have the same effect of inserting a NEW LINE in the text.

CTRL-L
Form Feed

*SPEED Control***Format**

↑L

Function

This control key specifies a form feed within a search or insertion command. SPEED clears the terminal screen when you key in CTRL-L, but restores it when you execute a C or I command, and after a second CTRL-D following search commands.

When SPEED is in page mode (WM is set at 0), it reads text into the buffer from form feed to form feed. The beginning form feed is part of the previous page. (The trailing form feed is also invisible.)

When SPEED is in page mode, it does not place form feeds in the buffer with the rest of the text. It keeps track of each form feed's place and reinserts it when you copy text to an output file. Therefore, if you need to search for or change a form feed, you must specify some nonzero window mode before you open the file. SPEED then leaves form feeds in the text. See the WM command.

When you insert a CTRL-L, SPEED echoes a ↑ L. When you close and reopen the file, SPEED no longer shows the form feed if your window mode is 0.

Example

```
!VN=$$  
  
60  
  
!ZJ$6<|!  
>$!↑L  
$$
```

You are editing a manuscript and wish to simulate the 66 lines of a standard typewriter page. When you ask, SPEED tells you that the buffer contains 60 lines. You jump to the bottom, insert six blank lines and a form feed.

CTRL-M
Carriage Return

*SPEED Control***Format**

↑M

Function

This control key specifies a carriage return within a search or insertion command. When you type it, SPEED will echo a new line. After it is inserted, SPEED echoes ↑ M. This key is equivalent, in a text string, to a carriage return (CR). It is *not* equivalent to the NEW LINE key. See CTRL-J.

Example

Smith thinks he has typed a vertical list.

```
*Hydrogen ↑ MHelium ↑ MLithium ↑ MBeryllium ↑ MBoron
```

SPEED's echo shows Smith that he typed CR instead of NEW LINE.

```
!<C[M$!  
$;>$$
```

```
Hydrogen  
Helium  
Lithium  
Beryllium  
Boron
```

Smith corrects the mistake with a Command Loop and a CTRL-M.

CTRL-N *Template*
Match anything in this position but the next character.

Format

search↑Nstring

Function

When you issue a search command with this template, SPEED matches `pass↑Nsword` with any string of the form `passxword`, where `x` is any character but `s`.

One ↑N does *not* negate another: `pass↑Nsword` and `pass↑N↑Nsword` make the same match.

You may use this template in conjunction with others:

↑N↑E matches one or more of anything that is not a tab or space.

↑N↑X matches the next portion of text that is not 0 or more occurrences of the next character.

↑N↑Y matches text up to the next repetition of the next character: ↑N↑YO successively places the CP after the decimal point on the first search and after the last 7 in `0.000770` on the second.

↑N↑\01234567↑\ matches any next character which is not an octal digit. ↑N↑\!'";?/.,↑\ matches any next character which is not one of the punctuation marks within the paired backslashes.

Examples

```
!Ssep↑Narate$$
```

You search for an instance of a misspelling by placing the ↑N template before the letter you think is misspelled.

```
!OVS3$<S↑N↑\1234567890↑\0$;V13>V3=$$
```

You make a count of leading 0s in your text by incrementing Variable 3 every time SPEED finds a 0 preceded by anything other than a digit.

CTRL-Q*Console Control***Unfreeze the terminal.**

Format

↑Q

Function

This control key allows you to regain control of the terminal. Try this key when your manipulation of the keyboard seems to have no effect. This key also restores roll to the screen if you are displaying text longer than the screen. To freeze the terminal, see CTRL-S.

Example

```
!XTYPE☐schedule$$  
↑S  
↑Q
```

Without leaving SPEED, you check an item in your file SCHEDULE. You use an X command to display an unopened file. As the file rolls by on the screen, you spot the item you want and freeze the screen. When you are through checking, you unfreeze the terminal with CTRL-Q.

CTRL-S*Console Control***Freeze the terminal.**

Format

↑S

Function

This key allows you to freeze the terminal. If the screen is rolling, it will halt. You may continue to type in commands blind, but as long as you maintain the freeze, no command you issue will take effect. To restore roll to the screen and regain control of the terminal, see CTRL-Q.

Example

```
!FOphone$#T$$  
↑S  
↑Q
```

You wish to look for an item in your file PHONE. You open the file and issue a command to display the entire buffer. When you spot the item, you freeze the terminal with a CTRL-S to inspect it closely. When you are through, you unfreeze the terminal with a CTRL-Q.

CTRL-T *Template*
**Match zero or more successive tabs
and spaces in this position.**

Format

search<CTRL-P>|Tstring

Function

Enter this control character from the keyboard by typing the sequence CTRL-P, CTRL-T. Typing in CTRL-T alone has no effect. When you issue a search command with this template, SPEED matches **pass**|Tword with *password*, *pass word*, and *pass* followed by any number of tabs and spaces followed by *word*.

|T has the same effect as the sequence |X|E.

Example

Looking for an abbreviation, you type

!Si.|P|Te.\$\$

The terminal echoes

!Si.|Te.\$\$

And SPEED successively finds both occurrences of the abbreviation in the text.

murdered by Jones, i. e. Smith, i.e.* Brown.*

CTRL-U *Console Control*
Cancel the current command line.

Format

|U

Function

This control key allows you to cancel a command line before you issue a CTRL-D. SPEED will erase the command line from the display when CTRL-U takes effect.

To cancel a series of commands extending over more than one line, use the CTRL-C, CTRL-A sequence.

Example

You begin inserting text and realize that you forgot to type the # I # command name.

!This is the last sentence tha|U

!

You avoid the display and exiting process which your first two letters inadvertently specify by cancelling the line with a CTRL-U.

CTRL-W*SPEED Control***Interpret the next character literally rather than as a template.**

Format

search]Wstring

Function

When you issue a search command with this template, SPEED matches `pass]W]Nsword` with `pass]N sword`; that is, with a string containing the next template character rather than the characters for which it is a template. Without the *CTRL-W*, SPEED would match `pass]Nsword` with `pass`, followed by any character but `s`, followed by `word`. With the *CTRL-W*, SPEED matches the search string with `pass`, followed by `]N`, followed by `sword`. Use this template to carry out searches in other SPEED command lines. To search for `]W` itself, use `]W]W`.

On models other than 6052, 6053, D100, or D200, use `]_` instead.

Example`!Scontain]N]$$$``containi*ng``!_1$BS1$C]W]N$$$BS0$]B1$$``contain]*`

You set a search for an instance of *contain* followed by any suffix such as *-s*, *-ing*, or *-ed*, and SPEED finds an instance. You decide to search instead for instances of *contain* without suffixes. You save the previous command line in Buffer 1, switch to that buffer, and enter a C command. Since the command contains *CTRL-W* before *CTRL-N*, it takes `]N` literally rather than as a negation of the next character. The command finds the template and deletes it. You switch back to your original buffer and execute Buffer 1. SPEED then finds an unaffixed instance of your word (that is not at the end of a line).

CTRL-X*Template***Match zero or more occurrences of the next characters.**

Format

search]Xstring

Function

When you issue a search command with this template, SPEED matches a search string of the form `a]Xbc` with the first expression it encounters of the form `ac`, `abc`, `abbc`, `abbbc`,

`]X]E` is equivalent to `]T`, and matches zero or more tabs and spaces.

`]X]Z` matches everything to the end of the buffer.

Examples`!S0.]X044721$$``0.44721*`

Uncertain of whether there are any 0s in the decimal expression you want to find, you type a *CTRL-X* into the search string. SPEED's first match contains no 0 in that position.

CTRL-Y *Template*
Match a sequence of one or more of the next character.

Format

search]Ystring

Function

When you issue a search command with this template, SPEED matches the search string O.]Y044721 with the first expression it encounters of the form *0.044721*, *0.0044721*, *0.00044721*, *0.000044721*,

]N]Y matches text up to the next repetition of the next character:]N]YO will successively place the CP after the decimal point and after the first 7 in *0.000770*.]Y]N has the same effect as]N]Y.

!S]N]YO\$\$

*0.*000770*

!S]N]YO\$\$

*0.0007*70*

]Y]E is equivalent to]E, and matches any number of tabs and spaces.

]Y]\character-list]\ matches any string consisting only of characters in the list between the backslashes:]Y]\01]\ matches any binary expression, and]Y]N]\01]\ matches any expression other than a binary one.

Example

!<C]Y_\$\$;>\$\$

An editor asks you to remove all underlining from a text. You do so by including a CTRL-Y before the underline in the C command of the Command Loop.

CTRL-Z *Template*
Match any single character in this position.

Format

search]Zstring

Function

When you execute a search command with this template, SPEED matches *pass]Zword* with the string consisting of *pass*, any one character, and *word*.

Remember to count a space, tab, carriage return (CR), NEW LINE, and form feed as single characters.

Remember that]Z must match a single character: *pass]Zword* does not match *password*.

]N]Z prevents a match of anything in the position.

]X]Z matches everything to the end of the buffer.

Examples

!Sc]Zn\$\$

Can maniac n*egate acne?*

SPEED matches in turn the three characters preceding each of the asterisks, but not those in the last word.

*can not, repeat can
not

!<Ccan]Znot\$cannot\$;>\$\$

%%cannot, repeat cannot%%

You delete both spaces and NEW LINES with the template in a Command Loop.

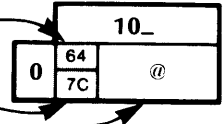
End of Chapter

Appendix A ASCII Character Set

To find the *octal* value of a character, locate the character, and combine the first two digits at the top of the character's column with the third digit in the far left column.

LEGEND:

Character code in decimal
EBCDIC equivalent hexadecimal code
Character



OCTAL	00_		01_		02_		03_		04_		05_		06_		07_	
0	0	NUL	8	BS (BACK-SPACE)	16	DLE P	24	CAN X	32	SPACE	40	(48	0	56	8
	00		16		18		40		4D		F0		F8			
1	1	SOH A	9	HT (TAB)	17	DC1 Q	25	EM Y	33	!	41)	49	1	57	9
	01		05		11		19		5A		5D		F1		F9	
2	2	STX B	10	NL (NEW LINE)	18	DC2 R	26	SUB Z	34	" (QUOTE)	42	*	50	2	58	:
	02		15		12		3F		7F		5C		F2		7A	
3	3	ETX C	11	VT (VERT TAB)	19	DC3 S	27	ESC (ESCAPE)	35	#	43	+	51	3	59	:
	03		0B		13		27		7B		4E		F3		5E	
4	4	EOT D	12	FF (FORM FEED)	20	DC4 T	28	FS \	36	\$	44	, (COMMA)	52	4	60	<
	37		0C		3C		1C		5B		6B		F4		4C	
5	5	ENQ E	13	RT (RETURN)	21	NAK U	29	GS]	37	%	45	-	53	5	61	=
	2D		0D		3D		1D		6C		60		F5		7E	
6	6	ACK F	14	SO N	22	SYN V	30	RS _	38	&	46	. (PERIOD)	54	6	62	>
	2E		0E		32		1E		50		4B		F6		6E	
7	7	BEL G	15	SI O	23	ETB W	31	US ~	39	' (APOS)	47	/	55	7	63	?
	2F		0F		26		1F		7D		61		F7		6F	

OCTAL	10_		11_		12_		13_		14_		15_		16_		17_	
0	64	@	72	H	80	P	88	X	96	\ (GRAVE)	104	h	112	p	120	x
	7C		C8		D7		E7		79		88		97		112	
1	65	A	73	I	81	Q	89	Y	97	a	105	i	113	q	121	y
	C1		C9		D8		E8		81		89		98		113	
2	66	B	74	J	82	R	90	Z	98	b	106	j	114	r	122	z
	C2		D1		D9		E9		82		91		99		114	
3	67	C	75	K	83	S	91	[99	c	107	k	115	s	123	}
	C3		D2		E2		8D		83		92		A2		C0	
4	68	D	76	L	84	T	92	\	100	d	108	l	116	t	124	
	C4		D3		E3		E0		84		93		A3		4F	
5	69	E	77	M	85	U	93]	101	e	109	m	117	u	125	}
	C5		D4		E4		9D		85		94		A4		D0	
6	70	F	78	N	86	V	94	or ^	102	f	110	n	118	v	126	~ (TILDE)
	C6		D5		E5		5F		86		95		A5		A1	
7	71	G	79	O	87	W	95	_ or _	103	g	111	o	119	w	127	DEL (RUBOUT)
	C7		D6		E6		6D		87		96		A6		07	

SD-00217 Character code in octal at top and left of charts.

| means CONTROL

Appendix B

Errors in SPEED

There are two classes of errors in SPEED: Text Editor errors and AOS errors. This appendix lists the Text Editor errors.

SPEED displays up to nine characters of the command line, beginning with the command in error. When the error is in a buffer or command file, the command line typed will be the invocation of the file or buffer in error, and other characters to a total of nine.

If you are a novice user of SPEED, you should realize that on occasions you *want* to receive an error message. For example, if you search for a misspelling in the hope that you don't find one, you receive the message

Error: Unsuccessful Search

with pleasure, not gloom.

SPEED Error Messages

Error: Access denied

The access control list for this file or directory does not permit the access you attempted.

Error: Attempt to delete current buffer

You cannot use a BKx command for the current buffer x.

Error: Attempt to edit permanent file

You attempted to open a file for updating but the file has permanence on.

Error: Attempt to execute current buffer

You cannot use a]Bx command for the current buffer x.

Error: Buffer is inactive.

You attempted to issue a BKx or]Bx command to inactive Buffer x.

Error: File already open

You attempted to open an input file with an FR,FW, or FO command without first closing a previously opened file.

Error: First argument greater than second argument

In commands that have two numerical arguments, the first must not be greater than the second.

Error: Illegal argument to command

You gave a negative argument to a command that accepts only 0 or a positive argument.

Error: Illegal buffer name

A through Z and 0 through 9 are the only legal buffer names. SPEED does not distinguish lowercase from uppercase letters in buffer names; BA and Ba name the same buffer.

Error: Illegal command

You used a character or characters that are not defined as a legal SPEED command.

Error: Illegal control character in search string

A character other than a legal control character appeared in a search command argument.

Error: Illegal number of arguments to command

Commands such as m,nR are not legal.

Error: Illegal variable name

Legal variable names are 0 through 9 only.

Error: Incomplete string in search buffer

You conducted no previous search or the search string was longer than 31 characters.

Error: Insert file too long

File will not fit in available space.

Error: Insufficient memory available

SPEED has exhausted all available memory during command execution or input. SPEED aborted the current command. If you want to save the old command line, type `_x`, where `x` is an available buffer. You can still execute a few commands, but you should issue a file output command very soon.

Error: Label not found

SPEED did not find the `!label!` you specified in the `O` (Over) command.

Error: LINE TOO LONG

A line of the current window exceeds 136 characters.

*** Lower case input encountered ***

Informational message. The `Y` and `A` commands check for lowercase characters in the data they transfer. The message indicates that SPEED encountered lowercase characters while case control was off. If you are at an uppercase terminal and wish to know which characters are lowercase, use a nonzero `WC` command and issue `T` commands for display.

Error: Maximum insert depth exceeded

You exceeded the nesting levels of `↑Bx` and `↑Fx` commands.

Error: Maximum iteration level exceeded

Command Loop nesting level exceeds 10.

Error: No more characters in input file

End of file reached by an `A`, `R`, or `Y` command.

Error: No open file

You attempted to use an `A`, `E`, `FB`, `FU`, `P`, `PW`, `R`, or `Y` command without an open file.

Error: Pathname too long

You have specified a pathname more than 127 characters long.

Error: Renaming error

An error occurred when SPEED attempted to rename the file in an `FB` or `FU` command. You may wish to do an `XFILESTATUS` command to see what has happened. Retry the command.

Error: Search string or < > broken over two levels

A search command or Command Loop starts at one command insert level and ends at another level.

Error: Stack overflow

Existing SPEED may be damaged. Close all files and re-execute SPEED to load a fresh `.PR` image into memory.

Error: Syntax error

You specified an incorrect command format.

Error: String argument too long

The string in your `Ostring$` command exceeds 32 characters.

Error: Unsuccessful search

`A`, `C`, `N`, `Q`, or `S` search command failed to match the search string.

Error: Unterminated string

You omitted the final delimiter in a `@C`, `@I`, `@N`, `@Q`, `@S` or `@T` command.

Error: Update mode on

You tried to issue an `FW`, `FR`, `FNW`, or `FNR` opening command while update mode is on.

Error: < with no corresponding >

You have entered only one angle bracket.

Error: " with no corresponding '

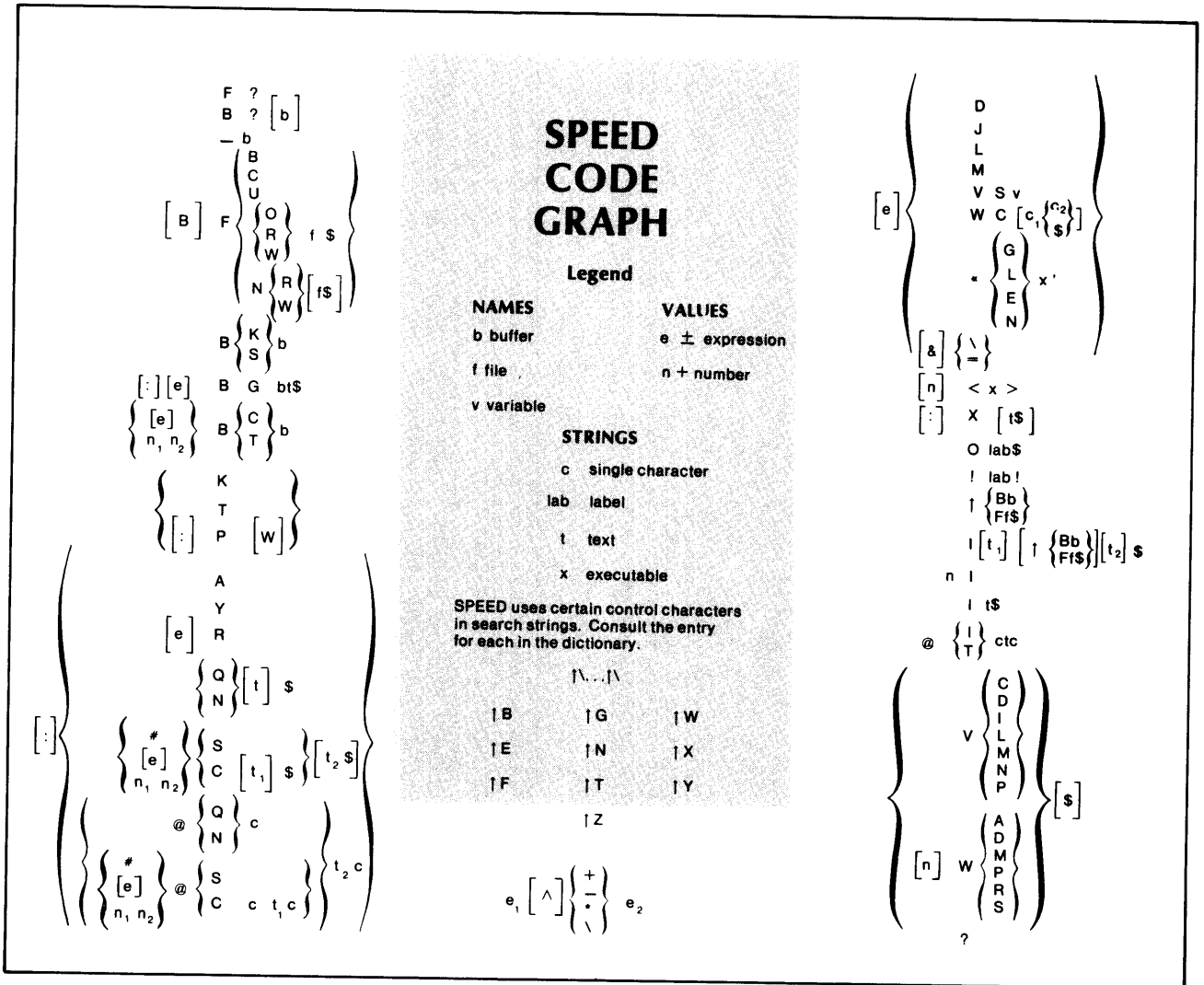
You entered a branch indicator (`"`) but no branch (`'`).

Error: ↑\ with no corresponding ↓

You entered only one `↑\` of the two required to enclose a character list.

End of Appendix

Appendix C SPEED Code Graph



End of Appendix

SPEED CODE GRAPH

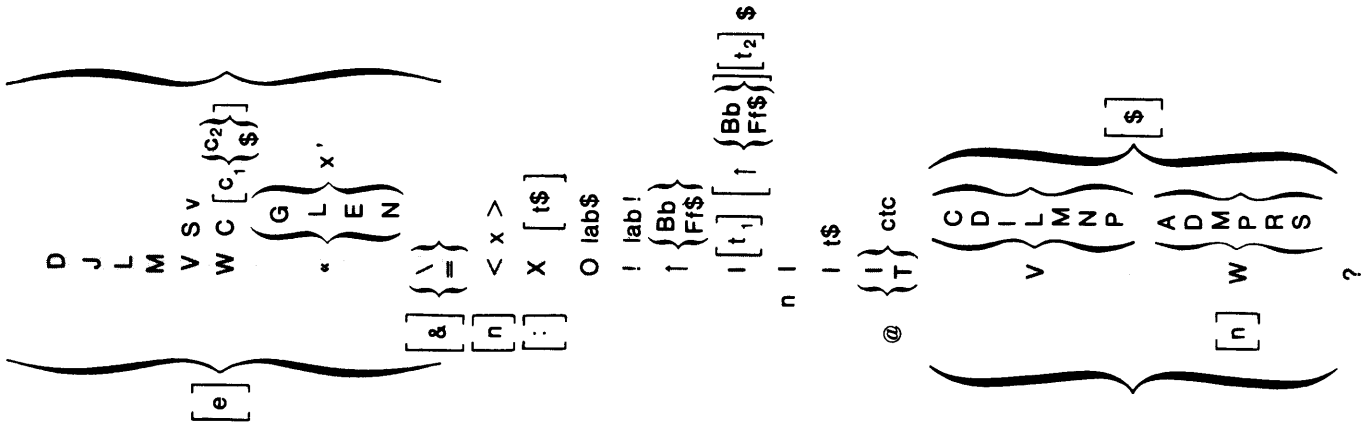
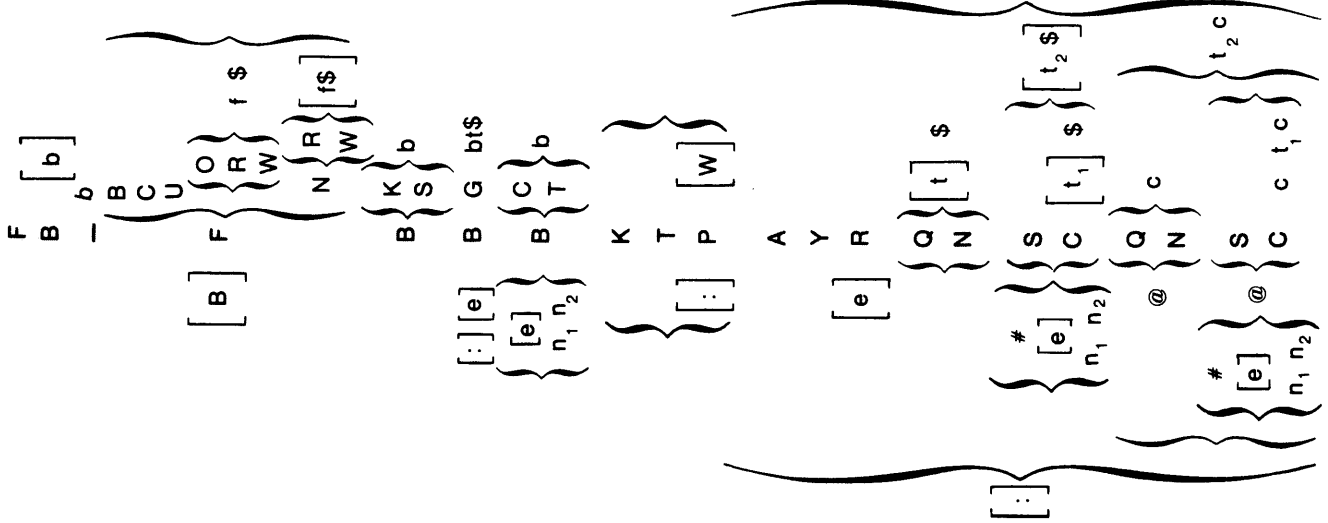
Legend

NAMES	VALUES
b buffer	e ± expression
f file	n + number
v variable	
STRINGS	
c	single character
lab	label
t	text
x	executable

SPEED uses certain control characters in search strings. Consult the entry for each in the dictionary.

1B	1G	1W
1E	1N	1X
1F	1T	1Y
	1Z	

$$e_1 \left[\begin{matrix} \wedge \\ + \\ - \\ \cdot \\ \backslash \end{matrix} \right] e_2$$



SPEED CODE GRAPH

Legend

NAMES
b buffer
f file
v variable

VALUES
e ± expression
n + number

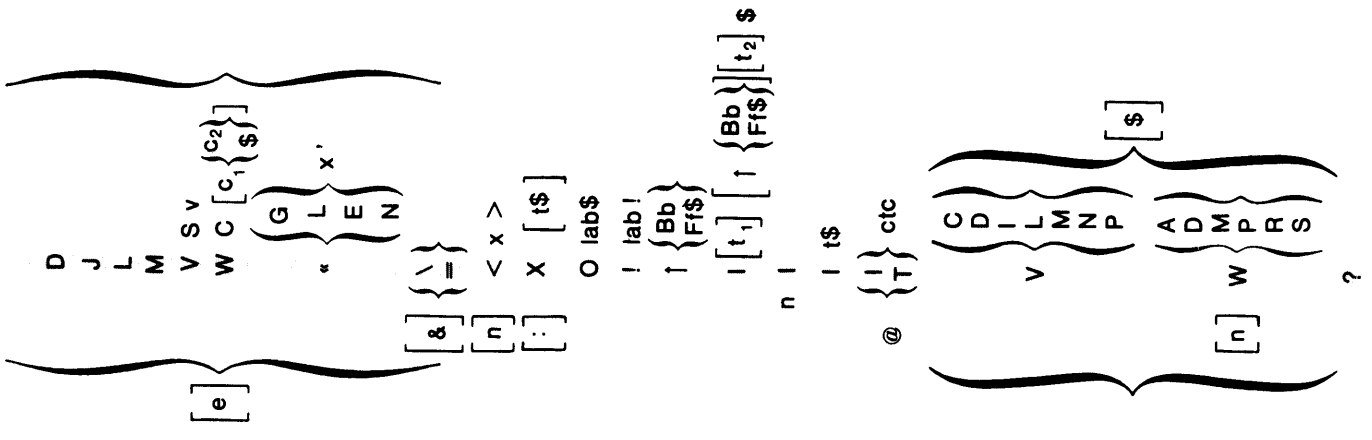
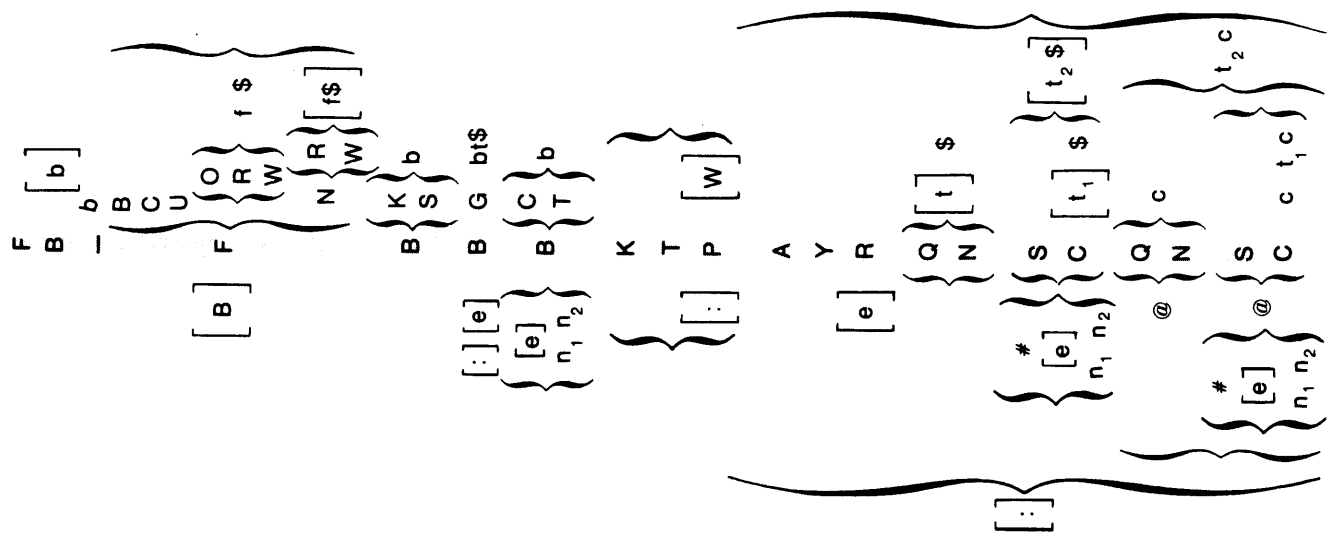
STRINGS

c single character
lab label
t text
x executable

SPEED uses certain control characters in search strings. Consult the entry for each in the dictionary.

1B	1G	1W
1E	1N	1X
1F	1T	1Y
	1Z	

$$e_1 \left[\wedge \right] \left\{ \begin{array}{l} + \\ - \\ \cdot \\ \setminus \end{array} \right\} e_2$$



SPEED CODE GRAPH

Legend

NAMES	VALUES
b buffer	e ± expression
f file	n + number
v variable	

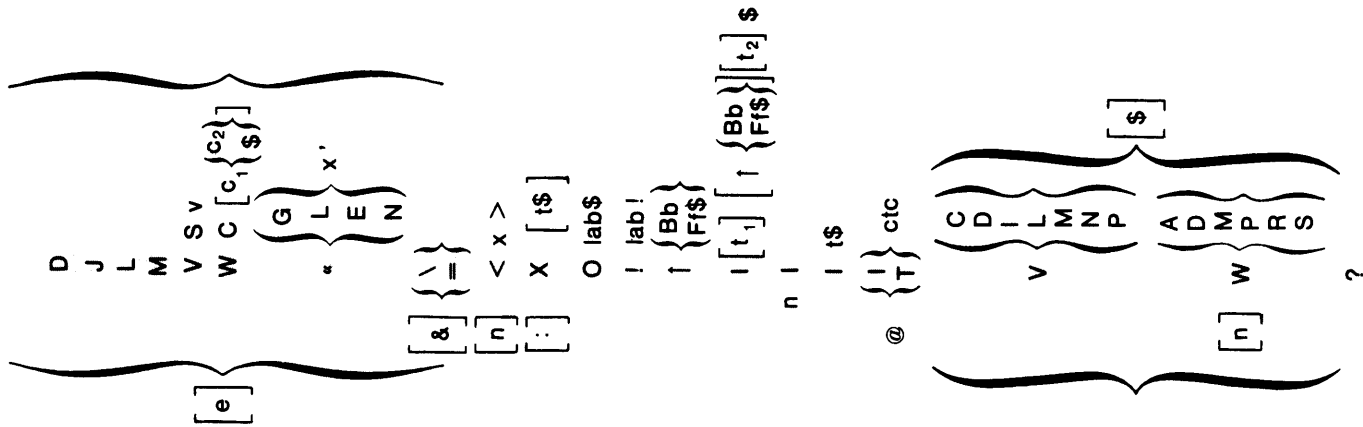
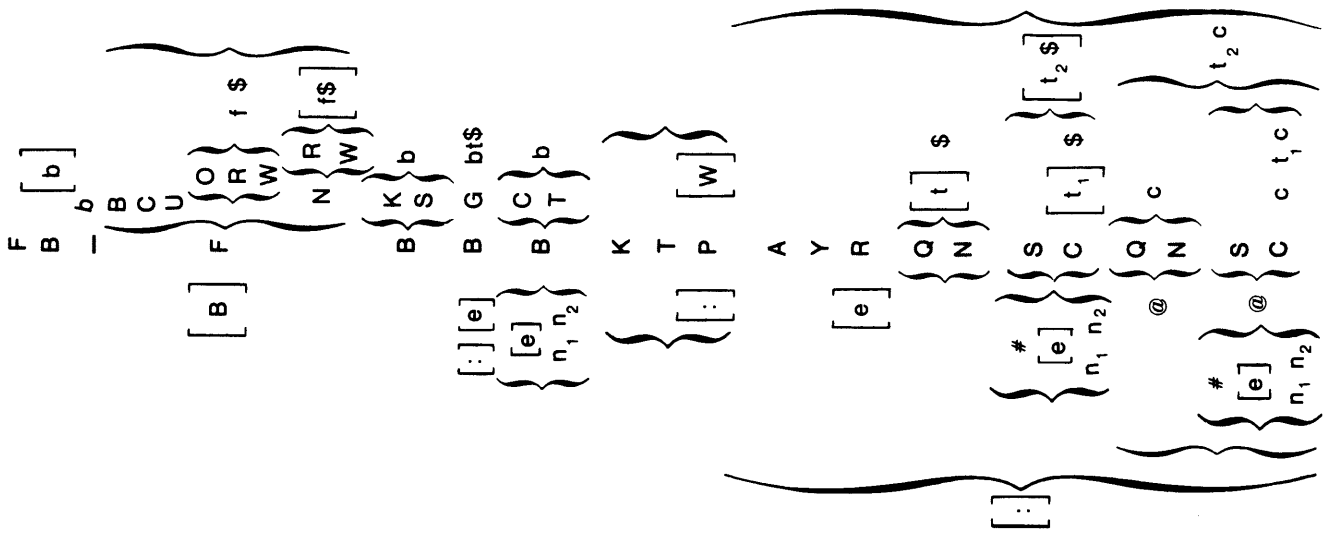
STRINGS

c	single character
lab	label
t	text
x	executable

SPEED uses certain control characters in search strings. Consult the entry for each in the dictionary.

1B	1G	1W
1E	1N	1X
1F	1T	1Y
	1Z	

$$e_1 \left[\begin{matrix} \wedge \\ + \\ - \\ \cdot \\ \backslash \end{matrix} \right] e_2$$



Appendix D

Functional Analysis of SPEED Commands

	F	B	F	B	F	F	N	N	F	B	F	N	B	F	N	F	N	P	:	P	:	N	Q	S	C	I
	O	O	R	R	R	R	W	W	W	W	A	R	Y	E	P	W	P	W								
Close input file	-	-	-	-	+	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Close output file	-	-	-	-	-	-	-	-	+	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Copy text to buffer	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Put in output file with ^L	-	-	-	-	-	-	-	-	-	-	-	+	-	+	+	-	+	-	-	-	-	+	-	-	-	
Put in output file without ^L	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-	+	-	-	-	-	-	-	
Shift buffers	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Open input file	+	+	+	+	+	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Clear buffer	+	+	-	-	-	-	-	-	-	-	-	+	+	+	-	-	+	+	-	-	-	+	+	-	-	
Append to buffer	+	+	-	-	-	-	-	-	-	-	+	+	+	-	-	-	-	-	-	-	-	+	+	-	-	
Open output file	+	+	-	-	-	-	+	+	+	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Move CP	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+	+	
Match text	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+	+	
Delete characters	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	-	-	-	-	+	-	
Delete lines	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	-	-	-	-	-	-	
Insert text	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	
Convert and insert text	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Type out or display	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Get variable value	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Get mode value	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Shift command control	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Iterate commands	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Conditional terminate level	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Update file	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Backup file	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Apply to local file	-	+	-	+	-	+	-	+	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Execute CLI	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Exit from SPEED	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	

Explanation:

- Read *across* to see which commands have which properties.
- Read *down* to see how the command works.
- Find the *intersection* of a command name and a property to see whether that command has that property or not.
- + Yes, command has this property
- No or does not apply

\	-x	D	K	B K	B T	B C	B G	B S	J	L	M	T	?	=	B ?	F ?	V x	W x	0	<x>	n"x)	;	F U	B U	F B	B B	F C	B C	H	X
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+	+	+	+	-	
-	+	-	-	-	+	+	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+	+	+	-	
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+	-	-	-	
-	-	-	-	-	-	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+	+	-	+	
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
-	-	-	-	-	-	-	-	-	+	+	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
-	-	+	+	+	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
-	-	-	-	-	-	+	-	-	-	-	-	+	+	+	+	+	-	-	-	-	-	-	-	-	-	-	-	-	-	
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-	
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-	-	-	
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+	+	-	-	-	
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-	
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	

(concluded)

End of Appendix

Index

Nonalphabetic entries precede all others and follow the ASCII sequence. Page numbers in italics indicate major entries. The letter "f" following a page entry means "and the following page;" "ff" means "and the following pages."

↑B 3-3, 4-6, 6-2, 7-101
 ↑C↑A 1-3, 7-102
 ↑C↑B 1-3, 7-102
 ↑D 1-3, 7-103
 ↑E 4-7, 7-103
 ↑F 4-6f, 6-3, 7-104
 ↑G 4-7, 7-105
 ↑I 1-3, 7-105
 ↑J 1-3, 7-106
 ↑L 1-3, 7-106
 ↑M 1-3, 7-107
 ↑N 4-7, 7-107
 ↑O 1-3
 ↑P 1-3
 ↑Q 1-3, 7-108
 ↑S 1-3, 7-108
 ↑T 4-7, 7-109
 ↑U 1-3, 7-109
 ↑W 4-8, 7-110
 ↑X 4-8, 7-110
 ↑Y 4-8, 7-111
 ↑Z 4-8, 7-111
 \$ see ESC (escape)
 \$\$ see ↑D
 ↑\ ... ↑\ 4-8, 7-100
 ↑↑ see ↑G
 ↑- see ↑W
 ! see Prompt
 !label! 5-2, 7-49, 7-82
 " see Conditional Execution command
 # see Entire buffer
 & 7-85
 ‘ see Conditional Execution command
 (↑) see Character Pointer
 * (blinking) see Character Pointer
 * see also Multiplication operator 4-1
 + see also Addition operator 4-1
 - see also Unary Minus operator, Subtraction operator 4-1
 . see Current Position of Character Pointer
 / see also Division operator 4-1

/D see Display switch
 /I= see Invocation switch
 : see Multipurpose modifier
 ; see Conditional Termination command
 < x > see Command Loop
 = see Equals command
 ? see Trace Mode Toggle
 @ see Commercial at modifier, temporary delimiter
 see Backslash command
 ^* see Logical AND
 ^+ see Logical Inclusive OR
 ^- see Logical NOT
 ^/ see Logical Exclusive OR
 -x see Save command

A

A see Append command 2-1
 Aborting SPEED see also ↑C↑B 1-7
 Addition operator 7-87
 ALPHA LOCK 1-3, 4-5
 ALT see ESC (escape)
 Alternate radix see also Ampersand modifier and Window Radix command 4-5, 7-85
 Ampersand modifier 7-85
 AND see Logical AND
 Apostrophe 5-2, 7-83
 Append command 7-5
 Arguments see Numerical arguments, Symbolic modifiers, Strings
 Arithmetic in SPEED 4-1
 ASCII character set A-1
 ASCII digit insertion see Backslash command
 At modifier see Commercial at modifier

B

Backslash command 4-1, 7-96
 B? see Buffers? command
 BC see Buffer Copy command
 BFB see Buffer File Backup command
 BFC see Buffer File Close command
 BFNR see Buffer File New Read command
 BFNW see Buffer File New Write command
 BFO see Buffer File Open command
 BFR see Buffer File Read command

BFU *see* Buffer File Update command
 BFW *see* Buffer File Write command
 BG *see* Buffer Get command
 BK *see* Buffer Kill command
 Boolean Complement *see* Logical NOT
 Boolean Functions in SPEED 4-2
 Boolean Intersection *see* Logical AND
 Boolean Symmetric Difference *see* Logical Exclusive OR
 Boolean Union *see* Logical Inclusive OR
 Branch over to label 5-2
 Branching *see also* Over command, Conditional Execution command 5-2
 BREAK ESC *see* ESC (escape)
 BS *see* Buffer Set command
 BT *see* Buffer Take command
 Buffer 3-1ff
 default 7-20
 definition 1-1, 1-5
 Buffer Copy command 3-2, 7-6
 Buffer File Backup command 7-8
 Buffer File Close command 3-4, 7-9
 Buffer File New Read command 7-10
 Buffer File New Write command 7-11
 Buffer File Open command 7-12
 Buffer File Read command 7-14
 Buffer File Update command 7-15
 Buffer File Write command 7-16
 Buffer Get command 6-1, 7-17
 Buffer Kill command 3-4, 7-19
 Buffer name 3-2
 Buffer Set command 3-3, 7-20
 Buffer Take command 3-3, 7-21
 Buffers? command 3-5, 7-23
 Building a command line 1-6

C

C *see* Change command
 Capitals, convention on 7-5
 Carriage return 1-3, 1-4, 7-107
 Case control mode *see also* Window Case command 4-4f
 Change command 2-3, 7-24
 Changing text 2-3
 Character 2-2
 ASCII A-1
 CTRL (control) 1-3
 definition 1-4
 lowercase 4-4
 positions 1-4
 shift-down 4-4f
 shift-lock 4-4f
 shift-up 4-4f
 uppercase 4-4
 Character Pointer 1-2, 7-86
 Characteristics (of commands) 7-3

Check
 buffer status 3-5
 CP position 2-4
 file status 2-5
 Choosing the SPEED modes 4-4
 Close a file 2-5
 Code graph C-1f
 Colon modifier *see* Multipurpose modifier
 Command file *see also* Macros 6-3
 Command, inadvertent 1-7
 Command line 1-1, 1-6
 canceling 1-7
 Command Loop 5-1, 7-50, 7-83, 7-91
 Conditional control of 5-1f, 7-90, 7-91
 Nesting of 5-2, 7-91
 Numerical control of 5-1, 7-91
 Command name 1-6, 7-4
 Command precedence 3-5, 7-5
 Commercial at modifier 4-3f, 6-1, 7-95
 Complement *see* Logical NOT
 Conditional Execution command 5-2, 7-50, 7-82, 7-91
 Execution conditions 5-2
 Conditional Iteration 5-3
 Conditional Termination command 5-1, 7-90, 7-91
 Conditionalizing the Next Command 4-3
 Confirm? 1-7, 7-39
 Q command 1-7, 4-9, 7-55
 Y command 1-7, 3-2, 7-79
 Control characters and templates 4-6
 Copy to a buffer *see* Buffer Copy command
 Copy to a buffer and delete *see* Buffer Take command
 Copy to a file *see* Write to a file
 Copying to another buffer 3-2
 Correcting text 2-2
 Console control keys 1-2, 1-3
 CP *see* Character Pointer
 CR *see* Carriage return
 Create a file 2-1
 Create new file? 7-1
 Creating and using SPEED macros 6-2
 Current Position of Character Pointer 2-4, 7-88
 Cursor 1-2

D

D *see* Delete command
 Default argument mode *see also* Window Argument command 4-4
 Default record length 7-17
 Default values *see also* Window Argument command, Window Position command
 DEL 1-7
 Delete command 2-3, 7-27
 Deleting characters 2-3
 Delimiter *see also* |D 1-5, 2-1, 7-4
 in a file expansion *see* |F
 standard 1-5, 7-4, 7-80
 temporary 4-3f, 7-95

Difference *see* Logical Exclusive OR
Digit insertion *see* Backslash command
Display mode *see also* Window Display command 4-5
Display switch 2-4, 7-2, 7-28
Display, terminal 1-1
Display text 2-1
Division operator 7-88

E

E *see* End command
E (equal to zero) *see also* Conditional Execution command 5-2, 7-82
End command 3-2, 7-19
Ending a SPEED session 2-5
Entering SPEED 1-7, 7-1
Entering with a filename 7-1
Entering without a filename 7-1
Entire buffer 7-85
 Total number of characters 7-80
 Total number of lines 7-65
Entry sequence (in dictionary) 7-5
Equals command 4-1, 7-93
Error messages *see also* Correcting errors
 B-1
 suppression *see* Multipurpose modifier
Errors 1-7, B-1
ESC (escape) 1-3, 1-6, 2-1, 7-80
Examples (of commands) 7-3
Exclusive OR *see* Logical Exclusive OR
Execute command 7-77
Executing a buffer 6-2
Executing commands across windows or pages 4-8f
Executing files 6-3
Executing the CLI *see also* Execute command 4-9
Exit Command *see also* Halt command 7-2
Exiting a Loop *see* Command Loop *and* Conditional Termination command
Exiting SPEED *see also* Halt command 1-7, 2-5, 7-2
Expansion *see* ↑B *and* ↑F
Expansions to buffers and files *see also* ↑B *and* ↑F 4-6f, 7-101, 7-104

F

F? *see* Files? command
FB *see* File Backup command
FC *see* File Close command
File 1-5
 input 1-5, 3-3 (Table 3-1)
 global 1-5, 3-1
 handling 3-5 (Table 3-2)
 local 1-5, 3-4f
 commands 3-4
 output 1-5,, 3-3 (Table 3-1)

File Backup command 2-5, 6-3, 7-30
File Close command 2-5, 7-31
File expansion *see* ↑F
File New Read command 3-1, 7-32
File New Write command 3-1, 7-33
File Open command 3-1, 7-34
File Read command 2-1, 7-36
File Update command 3-2, 6-3, 7-37
File Write command 2-1, 7-38
Files? command 2-5, 7-38
Flow control *see* Over command, Conditional Execution command
FNR *see* File New Read command
FNW *see* File New Write command
Form feed 1-4, 7-106
Format (of commands) *see also* Rules for commands 7-2
FO *see* File Open command
FR *see* File Read command
FU *see* File Update command
Function (of commands) 7-2
Functional analysis of SPEED commands 7-5, D-1
FW *see* File Write command

G

G (greater than zero) *see also* Conditional Execution command 5-2, 7-82

H

H *see* Halt command
Halt command 2-5, 7-39

I

I *see* Insert command
Inadvertent command *see* Command, inadvertent
Inclusive OR *see* Logical Inclusive OR
Insert
 character by ASCII value 4-2
 control keys 6-2, 7-2
 digits *see* Backslash command
 from a buffer *see* ↑B
 from a file *see* ↑F
Insert command 2-1, 2-4, 7-40
Insert text 2-1
Interacting with SPEED *see also* Buffer Get command 6-1
Iteration *see also* Command Loop *and* Conditional Execution command
Intersection *see* Logical AND
Invocation switch 6-3, 7-2, 7-42

J

J *see* Jump command
Jump command 2-2, 7-43

K

K *see* Kill command
Keyboard 1-2
Keys
 as characters 1-2
 as command names 1-2
 as control keys 1-2, 1-3
Kill command 2-3, 7-44
Killing a buffer 3-4
Killing lines 2-3

L

L *see* Line command
L (less than zero) *see also* Conditional Execution
 command 5-2, 7-82
Label *see* !label!, Over command
Last Character 2-4, 7-80
Line (of text) 1-5
Line command 2-2, 7-46
Line feed *see* NEW LINE
Line printer listings, a note about 7-2
Logical AND 4-3 (Table 4-1), 7-97
Logical Exclusive OR 4-3 (Table 4-1), 7-99
Logical Inclusive OR 4-3 (Table 4-1), 7-98
Logical NOT 4-3 (Table 4-1), 7-98
Loop *see* Command Loop
lowercase *see* Window Case command, Window Shifts
 command
Lowercase character *see* Character, lowercase

M

M *see* Move command
Macros *see also* Command file 1-1, 6-1, 6-3
Manipulating local and global files 3-4
Match case *see also* Window Shifts command 4-7f
Match text string
Modifying commands 4-3
Move
 CP
 from line to line 2-2
 across characters 2-3
 text to another buffer *see* Buffer Copy command,
 Buffer Take command
 text to an output file
Move command 2-3, 7-47
Multiplication operator 7-86

N

N *see* Nonstop command
N (not equal to zero) *see also* Conditional Execution
 command 5-2, 7-82
Nonstop command 4-8, 7-48
NOT *see* Logical NOT
Nulls, strip 7-35
Numerical arguments 1-6, 4-1, 7-2, 7-3
Numerical expressions as text 4-1
NEW LINE 1-3, 1-4, 7-106

O

O *see* Over command
Open a file 2-1
Operators
 arithmetic 4-1
 Boolean 4-2
OR *see* Logical Exclusive OR, Logical Inclusive OR
Organization of entries (in dictionary) 7-2f
Over command 5-2f, 7-49, 7-91

P

P *see* Put command
Page (of text) 1-5
Parent process 7-77
Permanence 7-1, 7-12, 7-34
Position mode *see also* Window Position command 4-5
Precautions and error messages (for commands) 7-3
Precedence *see* Command precedence
Prompt 1-2, 7-81
Put command 2-5, 7-51
Put in a file 7-51
Put Without command 7-53
PW *see* Put Without command

Q

Q *see* Quick command
Quick command 4-9, 7-54
Quotation mark 5-2, 7-82

R

R *see* Read command
Radix *see* Ampersand modifier *and* Window Radix
 command
Read command 3-2, 7-56
Read a page or window into the buffer 7-5, 7-56, 7-79
Related commands 7-3

Repeat key *see* REPT
REPT 1-3 1-7
RUBOUT *see* DEL
Rules for commands 1-6

S

S *see* Search command
Save command 6-1, 7-99
Saving a command line *see also* Save command 6-1
Search command 2-3, 7-57
Search strings and text strings 7-4
Search templates and control characters 4-7
Searching for text 2-3
Setting a temporary delimiter 4-3f
Son process 7-77
Shift sensitive mode *see also* Window Shifts command 4-6
Shifting case *see* Window Case command, Window Shifts command
Space (character) 1-4
Status 1-7
 of buffers *see also* Buffers? command 2-4
 of CP position *see* Current Position of Character Pointer
 of files *see* Files? command
 of lines and characters 2-4
Storing numerical expressions 4-1
Strings *see also* !label!, Filename 1-5
 executable strings *see also* !B, !F, Command line C-1
 search strings 1-5, 1-6, 7-4
 text strings 1-5, 1-6, 7-4
Structure of SPEED commands 7-3ff
Subtraction operator 7-87
Switches *see also* Display switch, Invocation switch 7-2
Symbolic modifiers 1-6, 7-3, 7-4
 Ampersand modifier 4-4, 7-85
 Commercial at modifier 4-3f, 6-1, 7-95
 Multipurpose modifier 2-5, 4-3, 4-9, 5-2, 7-89, 7-90
Symmetric Difference *see* Logical Exclusive OR

T

T *see* Type command
TAB *see also* !I 1-3, 1-4, 7-105
Template 1-2
Terminating a Loop *see* Command Loop *and* Conditional Termination command
Terminating SPEED *see* Exiting SPEED, Halt command
Terminator *see also* !D 1-5, 1-6, 7-5
Text, units of 1-4f
Trace Mode Toggle 6-1, 7-94
Tracing a command line *see also* Trace Mode Toggle 6-1

Type
 a line *see* Commercial at modifier *and* Type command
 digits *see* Equals command
Type command 2-1, 2-4, 7-59
Typing out text 2-4

U

Unary minus operator 7-87
Union *see* Logical Inclusive OR
Update Mode 3-1, 7-12, 7-15, 7-30, 7-34, 7-37, 7-55, 7-78, 7-79
Units of text *see* Text, units of
Uppercase *see* Window Case command, Window Shifts command
Uppercase character *see* Character, uppercase
User Data Area 7-13
Using decimal equivalents of characters 4-2
Using global files 3-1f
Using local files 3-4f
Using multiple buffers 3-2
Using the alternate radix 4-4
Using the /I= switch *see also* Invocation Switch 6-3

V

V *see* Value command
Value command 7-61
Value Character command 7-62
Value Decrement command 4-1, 7-63
Value Increment command 4-1, 7-64
Value Line command 2-4, 7-65
Value Moved command 2-4, 7-66
Value Number command 2-4, 7-67
Value Position command 7-68
Value Set command 4-1, 7-69
VC *see* Value Character command
VD *see* Value Decrement command
VI *see* Value Increment command
VL *see* Value Line command
VM *see* Value Moved command
VN *see* Value Number command
VP *see* Value Position command
VS *see* Value Set command

W

WA *see* Window Argument command
WC *see* Window Case command
WD *see* Window Display command
Window (of text) 1-5
Window Argument command 4-4, 7-70
Window Case command 4-4f, 7-71

Window Display command 4-5, 7-72
Window Mode command 4-5, 7-73
Window Position command 4-5, 7-74
Window Radix command 4-5, 7-75
Window Shifts command 4-6, 7-76
WM *see* Window Mode command
WP *see* Window Position command
Write to a file *see also* File, output 2-5, 3-2
Writing SPEED commands 6-1f
WR *see* Window Radix command
WS *see* Window Shifts command

X

X *see* Execute command

Y

Y *see* Yank command
Yank command 3-1, 7-79

Z

Z *see* Last Character symbol

How Do You Like This Manual?

Title _____ No. _____

We wrote the book for you, and naturally we had to make certain assumptions about who you are and how you would use it. Your comments will help us correct our assumptions and improve our manuals. Please take a few minutes to respond.

If you have any comments on the software itself, please contact your Data General representative. If you wish to order manuals, consult the Publications Catalog (012-330).

Who Are You?

- EDP Manager
- Senior System Analyst
- Analyst/Programmer
- Operator
- Other _____

 What programming language(s) do you use? _____

How Do You Use This Manual?
(List in order: 1 = Primary use)

- _____ Introduction to the product
- _____ Reference
- _____ Tutorial Text
- _____ Operating Guide
- _____ _____

Do You Like The Manual?

Yes	Somewhat	No	
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Is the manual easy to read?
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Is it easy to understand?
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Is the topic order easy to follow?
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Is the technical information accurate?
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Can you easily find what you want?
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Do the illustrations help you?
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Does the manual tell you everything you need to know?

Comments?
*(Please note page number and paragraph where applicable.)***From:**

 Name _____ Title _____ Company _____
 Address _____ Date _____

CUT ALONG DOTTED LINE

FOLD

FOLD

TAPE

TAPE

FOLD

FOLD



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

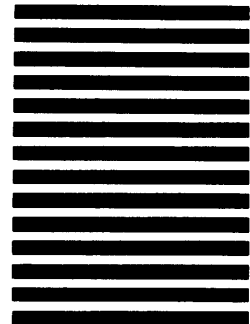
Postage will be paid by addressee:

 **Data General**

ISD User Documentation, M.S. E-111

4400 Computer Drive

Westborough, Massachusetts 01581



DataGeneral Users group

Installation Membership Form

Name _____ Position _____ Date _____
 Company, Organization or School _____
 Address _____ City _____ State _____ Zip _____
 Telephone: Area Code _____ No. _____ Ext. _____

1. Account Category

- OEM
 End User
 System House
 Government

5. Mode of Operation

- Batch (Central)
 Batch (Via RJE)
 On-Line Interactive

2. Hardware

M/600
 C/350, C/330, C/300
 S/250, S/230, S/200
 S/130
 AP/130
 CS Series
 N3/D
 Other NOVA
 microNOVA
 Other _____
 (Specify) _____

Qty. Installed	Qty. On Order

6. Communications

- RSTCP CAM
 HASP 4025
 RJE80 Other
 SAM

Specify _____

7. Application Description

○ _____

3. Software

- AOS RDOS
 DOS RTOS
 SOS Other

Specify _____

8. Purchase

From whom was your machine(s) purchased?

- Data General Corp.
 Other
 Specify _____

4. Languages

- Algol Assembler
 DG/L Interactive
 Cobol Fortran
 ECLIPSE Cobol RPG II
 Business BASIC PL/1
 BASIC Other

Specify _____

9. Users Group

Are you interested in joining a special interest or regional Data General Users Group?

○ _____

CUT ALONG DOTTED LINE

FOLD

FOLD

TAPE

TAPE

FOLD

FOLD



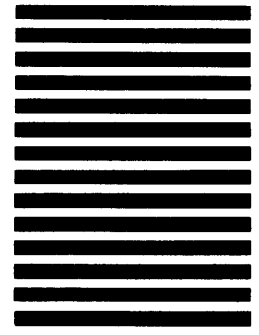
NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:



ATTN: Users Group Coordinator (C-228)
4400 Computer Drive
Westboro, MA 01581



—

—

—

