Data General

Software Documentation

RDOS/DOS Superedit Text Editor

RDOS/DOS Superedit Text Editor

069-400017-01

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. 069-400017

Revision 01, February 1984

RDOS/DOS
Superedit
Text Editor
069-400017

Revision History:

Original Release - June 1983

First Revision - February 1984

CONTENT UNCHANGED

The content in this revision is unchanged from 069-400017-01. This revision changes only printing and binding details.

Preface

This manual describes Superedit (Speed), a text editor that runs on the Real-Time Disk Operating System (RDOS) and Disk Operating System (DOS). We have organized this manual as follows:

Chapter 1 introduces Speed, tells you how to execute Speed from the RDOS or DOS Command Line Interpreter (CLI), and describes general concepts and terms.

Chapter 2 directs you through a sample editing session and describes a few of the rudimentary Speed commands you will use most often.

Chapter 3 discusses the remaining Speed commands.

Chapter 4 contains nine examples which use most of the Speed commands described in this manual. (In addition, we have included working examples throughout the manual.)

Chapter 5 contains an alphabetical listing and brief definitions of Speed commands.

Chapter 6 contains RDOS/DOS implementation notes and information about memory utilization and error handling.

Appendix A contains an ASCII character set.

Appendix B lists all Speed error messages and their meanings.

We assume that you are operating on-line with an RDOS or DOS system. If you are not or if you have never before used a string-oriented text editor, you should read the appropriate sections of the *Introduction to RDOS* (DGC No. 069-400011).

Related Manuals

The following manuals are part of a series of books that document RDOS and DOS.

Introduction to RDOS (DGC No. 069-400011) describes the fundamentals of using RDOS and summarizes the features, utilities, and capabilities of the operating system.

Guide to RDOS Documentation (DGC No. 069-400012) describes all of the books that comprise the revised documentation set for RDOS and DOS, and lists the previous books that each replaces.

How to Load and Generate RDOS (DGC No. 069-400013) explains how to load, generate, and maintain RDOS. Instructions are provided for preparing hardware, program loading, initializing disks, installing the bootstrap root and starter system, tailoring, and system backup, among others.

How to Generate Your DOS System (DGC No. 093-000222-01) explains how to generate and maintain DOS. Instructions are provided for preparing hardware, initializing disks, installing the bootstrap root and starter system, tailoring, system backup, and optimization, among others.

RDOS/DOS Command Line Interpreter (DGC No. 069-400015) discusses the user interface with the operating system. It covers the Command Line Interpreter (CLI) features and command mechanisms, as well as instructions on how to use CLI commands. It also presents features and operation of the Batch monitor, a CLI utility.

RDOS/DOS Text Editor (DGC No. 069-400016) documents how to load, use, and operate the single-user Text Editor (Edit) or Multi-user Text Editor (Medit) to create and edit text files.

RDOS System Reference (DGC No. 069-400027) describes RDOS system features, system calls, and user device driver implementation for assembly language and high-level language programming.

DOS Reference Manual (DGC No. 093-000201-03) discusses all features of DOS, including system control, memory management, and system calls for system programmers.

RDOS/DOS User's Handbook (DGC No. 093-000105-04) summarizes the commands, calls, and error messages of RDOS and DOS, the Command Line Interpreter (CLI), the text editors (Edit, Medit, and Speed), the Batch monitor, and utility programs.

RDOS/DOS Assembly Language and Program Utilities (DGC No. 069-400019) details the Extended Assembler (ASM), Macroassembler (MAC), Extended Relocatable Loader (RLDR and OVLDR), and Library File Editor (LFE) utilities that aid in programming.

RDOS/DOS Debugging Utilities (DGC No. 069-400020) describes five utilities that assist you in editing, debugging, and patching programs–the Symbolic Editor (SEDIT), Symbolic Debugger (DEBUG), Disk Editor (DSKED), and Patch (ENPAT and PATCH).

RDOS/DOS Sort/Merge and Vertical Format Unit Utilities (DGC No. 069-400021) offers functional information on Sort/Merge (RDOSSORT), which helps you manipulate records in data files, and the Vertical Format Unit (VFU), which enables you to define data channel line printer page formatting.

RDOS/DOS Backup Utilities (DGC No. 069-400022) presents the features and operation of the utilities that perform disk and tape backup. These are BURST/TBURST, DBURST/MBURST/RBURST, DDUMP/DLOAD, FDUMP/FLOAD, and OWNER.

Typesetting Conventions

We use the following conventions for command formats in this manual:

COMMAND *required* [*optional*] . . .

Where	**Means**
COMMAND	Enter the command (or its accepted abbreviation as shown. Uppercase letters indicate the command mnemonic.
required	Enter an argument, such as filename. Lowercase italic letters indicate this argument. Required arguments also appear as: *required*$_1$ | *required*$_2$ In this case, you may choose between the arguments. Do not type the vertical bar; it merely separates the choices.
[*optional*]	Brackets mean that you have the option of entering the argument indicated in lowercase italic letters. Command switches also appear in this format. Do not include the brackets in your code; they only set off the choices.
....	Repeat the preceding entry or entries. The explanation indicates exactly what to repeat.
. . .	Where this symbol appears, the process has continued without incident and you may now take the next action described.

In examples of dialogue, we use:

THIS TYPEFACE TO SHOW YOUR ENTRY

and

THIS TYPEFACE TO SHOW SYSTEM RESPONSES

Additionally, we use certain symbols in special ways:

Symbol	Means
⟨CR⟩	Carriage Return. Press the CR key on your keyboard. **NOTE:** If you have a D100, D200, or G300 terminal, you should press the New Line key (NEW LINE) on your keyboard instead.
□	Include a space at this point. We use this to clarify command lines in some cases. Normally, you can see where to put spaces.
CTRL-	Depress and hold the Control key (CTRL) while you press the character that follows CTRL-.
⟨NL⟩	New Line. Press the NEW LINE key on your keyboard.
R	RDOS/DOS Command Line Interpreter prompt.

Throughout this manual, we show commands that are preceded by the Speed prompt, an exclamation point (!). When we direct you to type a command, you should start typing after the ! prompt.

All numbers are decimal unless we indicate otherwise; for example, to indicate octal 35, we use 35_8.

The keys defined ad DEL and RUBOUT perform the same function. Depending on the console you are using, you will find one of these keys on your keyboard. In this manual, we use DEL to represent that function.

The up arrow symbol (↑) is also executed by different keys, depending on your console. You execute it by pressing SHIFT-6.

We welcome your suggestions for the improvement of this and other Data General publications. To communicate with us, use the postpaid comment form at the end of this manual.

Table of Contents

Figures

Tables

Chapter 1

Introduction and Concepts

The Data General advanced text editor, Superedit (Speed), is a character-oriented text editor that runs under the Real Time Disk Operating System (RDOS) or the Disk Operating System (DOS) system. With Speed you can insert, store, add, delete, change, or copy text. Furthermore, you can do these things with Speed:

- Perform most basic text-editing functions with a few simple commands.
- Experiment with more complex variations and command extensions, which enable you to perform difficult editing tasks easily.
- Edit all types of data files in uppercase or lowercase letters.
- Maintain up to 36 edit buffers simultaneously.
- Maintain and perform input/output (I/O) on multiple files.
- Create and store your own editing programs, using the Speed macro instructions.
- Set conditions for Speed to check before carrying out a task.
- Use numeric variables as arguments in your commands, either alone or in argument expressions to undertake repetitive tasks.

Running Speed

The RDOS Command Line Interpreter (CLI) prompt appears as an *R*. To run Speed on an ECLIPSE computer, at the *R* prompt type:

```
SPEED ⟨CR⟩
```

or

SPEED *filename* ⟨CR⟩

When Speed has started, you get the Speed prompt, which is an exclamation point (!).

If *filename* exists, Speed opens the existing file for input, creates and opens a scratch file for output, and reads the first page of the text file into memory. If *filename* does not exist, Speed creates and opens an output file. Your console displays:

```
CREATING NEW FILE
!
```

If you do not include *filename*, you have to open a file with a file specification command (discussed later).

If you are working on a NOVA® system, the Speed command is preceded by an N. The general formats to start Speed on a NOVA® system are:

```
NSPEED ⟨CR⟩
```

or

NSPEED *filename* ⟨CR⟩

For simplicity, this manual shows only the ECLIPSE Speed version of the command. You may want to create a link entry that allows you to use the command SPEED (instead of mentally substituting NSPEED whenever you see the command SPEED). To create this link you must be in the CLI. At the *R* prompt, type:

```
LINK SPEED.SV NSPEED.SV ⟨CR⟩
```

You can now use the SPEED command as it appears throughout the book.

The Terminal Display

When you enter Speed, it creates a buffer or workspace from which you can display the text you wish to edit. Speed displays the text you are currently editing (whether you type it in from your terminal or bring it into the buffer from an open file) and the command line you type in from the keyboard. When your console displays both text and a command line, the current command line follows the text. Chapter 2 describes how to use the Type-out (T) command to display text. Thoughout the manual we show the T command in examples.

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 indicate a control character with a preceding CTRL-, as in CTRL-A, in examples that you are expected to type. We indicate a control character as an uparrow (↑) followed by the character in responses to commands displayed on the console. For example, CTRL-I is shown in displays as ↑I.

We describe Speed's use of control characters throughout the manual. For your convenience, we list here the most commonly used control characters and their functions.

CTRL-X	Deletes a single line from the command string.
CTRL-A	Aborts an entire command line on one or more text lines.
CTRL-R	Retypes the current line of a command string for verification.
CTRL-L	Inserts a Form Feed.
CTRL-I	Inserts a Tab.
CTRL-M	Inserts a Carriage Return.

Erasing Characters and Lines

Use the RUBOUT (DELETE) key to delete characters when you make a typing error. In this manual, we refer to RUBOUT as DEL. Each time you press DEL, Speed deletes the last character you entered and displays that character on the terminal. For example:

```
IHELLOOLLEH
```

I is the text insertion command described in Chapter 2. HELLOOLLEH is the word HELLO, and the letters your console displays as you press DEL. O is the first letter deleted, L the second, and so on. To remove HELLO from the line below and replace it with BYE, press DEL once for each letter to be deleted, then type in the word BYE. Your console displays:

```
IHELLOOLLEHBYE$$
```

Speed inserts only BYE into the file.

You can delete a single line by pressing CTRL-X. This command is equivalent to pressing DEL for each character in the line.

Character Pointer

The Character Pointer (CP) is an editing cursor (↑) that is always positioned between two characters, never at a character. When you want to modify or examine text, position the CP at the exact location where you want to begin working. For example, if the CP is in this position:

```
ABCD↑F
```

you might insert an *E* to produce the string *ABCDEF*. Notice that you may enter text into the buffer only at the position of the CP.

You can move the CP quickly and precisely to any position in the current buffer by using the various CP commands, which we describe in Chapter 2.

Text

Text consists of a sequence of one or more ASCII characters. The following conventions are used in Speed for accessing and organizing text in an ASCII text file:

Character	A single ASCII alphanumeric. Speed uses the full uppercase and lowercase ASCII character set shown in Appendix A.
String	A sequence of characters. It has no delimiters and can include any ASCII character.
Line	A string of characters up to and including a Carriage Return (shown in this manual as ⟨CR⟩). When you enter lines of text and when Speed types lines of text on your terminal, each string delimited by a Carriage Return character appears on a separate line.
Page	A sequence of characters ending in a New Page character (Form Feed or CTRL-L). A page of text has no size limit. You can create a page by inserting a CTRL-L in the text; you can merge two adjacent pages by deleting the CTRL-L that separates them; and you can change the page length at any time by repositioning CTRL-L. Pages exist merely to control the amount of text you are editing. Speed brings one page at a time into a buffer.
Window	A sequence of characters divided into a specific number of lines. You initially specify (and can later change) the window length by using the window mode commands. Use window mode to organize and control the amount of text that you edit.

Edit Buffers

An edit buffer is a temporary storage compartment shared between your input and output files. Speed lets you use up to 36 edit buffers (coded 0 through 9 and A through Z) to hold text. Speed commands allow you to transfer data from input files to any buffer and from one buffer to another, and to copy buffer contents into output files. A buffer has no fixed length; it is as long as the text you move into it.

You can have only one current buffer. Initially, the current edit buffer is buffer 0. If you want to manipulate text in another buffer, you must activate the buffer. (See Buffer Commands in Chapter 3).

Figure 1.1 depicts the flow of information between the input file and the output file. Note that the edit buffer serves as a temporary storage compartment between the two files.

Input and Output Files

The input file contains the text you want to edit. The output file is the destination of the edited text. If you specify a file in the CLI command that invokes Speed (SPEED*filename*), then Speed opens that file for input and *filename.SC* for output. If you do not specify a file in the CLI command line, then you must use Speed commands, described in Chapters 2 and 3, to open your files.

Escape Character

The Escape or ALT MODE character serves two important functions. First, you press the Escape key (ESC) once (displayed on the screen as $) to separate a command string from the next command character. Second, you type two consecutive ESCs ($$) to signal the end of the command string. The two ESCs (ESC ESC) instruct Speed to execute the command string. We refer to the command delimiter as double ESC or ($$) throughout the rest of this manual.

Command Structure

The general format of a Speed command is

[*num-arg*] COMMAND [*string arg*] $$

[*num-arg* [,*num-arg*]] COMMAND [*string-arg* $ [*string-arg* $]]$$

where:

num-arg	is a decimal number or an expression which evaluates to a decimal number.
COMMAND	is any Speed command.
string-arg	is a string of ASCII characters.

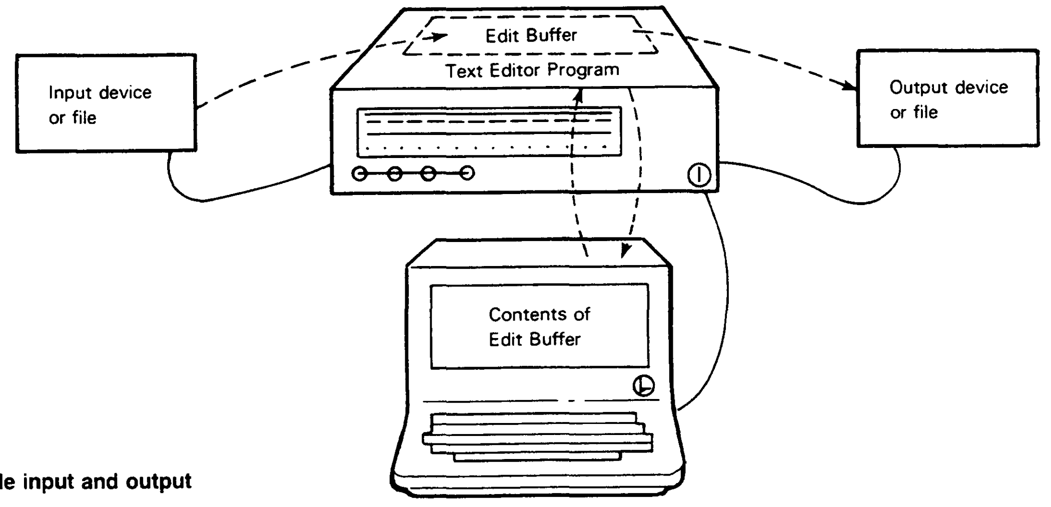

Figure 1.1 File input and output

Note that if you include two numeric arguments in a command string you must separate them with a comma. If you include two string arguments, you must separate them with the ESC character. You must also terminate all Speed commands with two ESC characters.

A Speed command may consist of one letter, without an argument. A Speed command may be preceded by one or two numeric arguments and it may be trailed by one or more string arguments.

At any instant, Speed is either ready to receive your instructions (in input mode) or is executing them (in execution mode). The prompt (!) indicates that Speed is in input mode. Two Escapes ($$) put Speed into execution mode. The reappearance of the prompt (!) indicates that Speed is back in input mode and is awaiting further commands.

Chapter 2 leads you through a sample editing session in which you use basic Speed file and editing commands.

Chapter 2

Sample Editing Session

The Editing Process

You edit text in Speed by following this procedure:

1. Start Speed.
2. Open files for input and output.
3. Insert text into the buffer from the console or read text from an input file into the buffer.
4. Edit the text in the current buffer.
5. Write the contents of the edit buffer to the output file.
6. Clear the edit buffer.
7. Repeat steps 3 through 6 until you have finished entering and modifying text.
8. Close the input and output files.
9. Exit from Speed.

Step 5 is extremely important. You must issue an output command (described in this chapter and in Chapter 3) to preserve the material you have edited. The rest of this chapter contains a sample editing session to acquaint you with the fundamentals of Speed. Please work through the examples as you read.

Starting Speed

To begin the session, start Speed at the *R* prompt with

```
SPEED ⟨CR⟩
```

Your console displays the Speed prompt (!).

Get a File for Writing (GW)

Use the GW command to create an output file. The general form of the command is:

GW*filename*$

The GW command creates and opens the specified output file. Note that filename must not be a previously existing file. It can be an output device such as the line printer ($LPT). Create a file called SAMPLE with this command:

```
!GWSAMPLE$$
```

At this point, SAMPLE exists but is empty. When we finish making changes to the text, we will show you how to write the contents of the buffer into file SAMPLE.

The Insert Command (I)

Make text insertions by typing the I (Insert) command, followed by the text you want to insert. Insertion commands enter the string of characters specified by your command line into the current buffer, starting at the position of the Character Pointer (CP). If you do not start the text insertion with the I command, Speed does not enter the text in the buffer when you type the double Escape command delimiter–ESC ESC. Instead, Speed interprets the text as a command string, and the console displays an error message. Once Speed inserts the text, it repositions the CP at the space following the last character of the insertion.

Using CTRL-I *text* is similar to the I command except that it inserts a Tab into the edit buffer preceding the text string. For example, if you type:

```
CTRL-ITHIS IS A TAB.$T$$
```

your console displays:

THIS IS A TAB.

Predefined Tab positions occur at intervals of eight spaces such as 1, 9, 17, 25. Each time you type CTRL-I, the CP moves to the next predefined Tab position.

Speed does not accept certain ASCII characters as text in an inserted string. For example, if you try to insert certain control characters into a text string, the control characters signal Speed to perform a function rather than to appear as a character.

Using *n*I inserts a single character *n*, where *n* is the decimal equivalent of any ASCII character listed in Appendix A.

For example, assume that you want to insert an ESC into a string insertion. The number 27 represents an ESC in decimal. For example, if you type:

```
!I000$27I$I222$$
```

I000 inserts three zeroes, 27I inserts an ESC character, and I222 inserts three twos. Your console displays:

```
000$222
```

Using *n* inserts the ASCII representation of the decimal *n* into the buffer at the CP location, suppressing leading zeroes. Examples are shown in Chapter 4.

Using I CTRL-E CTRL- followed by another control character inserts both characters into the line. For example, typing I CTRL-E CTRL-X$$ inserts CTRL-E CTRL-X into the line. This allows you to create, within a buffer, a command line that searches for a control character.

For a list of insertion commands, see Table 3.5.

The Character Pointer

To make changes to the practice text, you need to be able to move your cursor to a specific location. The CP marks the position in the edit buffer at which you are currently editing. Practice using the CP commands along with the Insert (I) command. Remember that the CP always resides *between two characters*, never *at* a character. You can move the CP forward or backward, by characters or by lines. Table 2.1 lists the CP commands.

Command	Description
J or 1J or 0J.	Moves the CP to the beginning of the buffer's first line.
*n*J	Moves the CP to the beginning of line *n* in the current buffer. *n* is always relative to line 1, the first line of a current buffer.
ZJ	Places the CP after the last character in the buffer.
L	Moves the CP to the beginning of the current line.
*n*L	Moves the CP *n* lines forward from its present position across *n* Carriage Returns and places it at the beginning of the line following the *nth* carriage return.
-*n*L	Moves the CP backward to the beginning of the *n*th line preceding the current line.
*n*M or -*n*	Moves the CP *n* characters to the right or left. If *n* ⟩ 0, the CP moves *n* characters to the right. If *n* ⟨ 0, the CP moves *n* characters to the left.

NOTE: You do not need to put an ESC after a CP command in a command string. You do, however, have to terminate the entire command string with ESC ESC.

Table 2.1 Character pointer commands

Type I, followed by the practice text in Figure 2.1. We have included typographical errors in the practice text so that you can correct them, using Speed commands. Follow the text with a double ESC ($$).

```
!ITHE FOLLOWING CHAPTER GUIDE YOU THROUGH A SAMPLE EDITING CESSION <CR>
UNDER SPEED. SINCE OUR INTENTION IS TO MAKE THIS PROCESS AS SIMPLE AND <CR>
PAINLESS AS POSSIBLE, THE TEXT YOU ARE NOW READING WILL SERVE AS THE SAM <CR>
PLE PROGRAM. ALL COMMANDS IN THE SAMPLE EDITING SESSION ARE DIRECTED <CR>
TOWARD THIS TEXT. ERRORS THAT YOU SEE HERE ARE INTENTIONAL AND VAN BE <CR>
CORREECTED BY SIMPLE COMMANDS FROM THE TERNINAL.ONCE SPEED HAS BEEN LOADED <CR>
AND IT DI SPLAYS A PROMPT, YOU ARE READY TO ENTER THE FIRST COMMAND. <CR>
THIS CHAPTER DISCUSSES THE COMMANDS THAT YOU'LL USE MOST OFTEN. <CR>
IN ONE EASY LESSON, YOU'LL LEARN HOW TO OPWN AND CLOSE FILES, INSERT TEXT, <CR>
TEXT, CORRECT TYPING ERRORS, MOVE THE CHARACTER POINTER, AND EXIT FROM <CR>
SPEED. <CR>
```

Figure 2.1 First page of sample text DG-09490

After you have typed in the text in Figure 2.1, you may wish to use the CP commands described in Table 2.1. Brief examples follow.

The ZJ command positions the CP after the word SPEED at the end of the page, and J positions the CP immediately before THE in line 1 of SAMPLE.

The 3J command positions the CP before PAINLESS in line 3.

The L command positions the CP at the beginning of the current line. Starting at line 3, use 3L to position the CP before COREECTED in line 6, and -3L to put it back before PAINLESS.

Again starting at line 3, the 2M command moves the CP to INLESS, and -2M moves it back before PAINLESS.

Use the commands presented so far to make additions and corrections to the text. Continue changing it until you feel completely at ease with all of the commands. When you have finished editing the contents of the buffer, send them to the output file with a P command described in the section called Put (P). Then type #K to clear the buffer.

Additional Text

To insert the contents of the second page into the file SAMPLE, type in the contents of Figure 2.2.

Global Close (GC)

At the moment, output file SAMPLE is open. To close a currently open output file, type:

```
GC$$
```

The GC command closes SAMPLE for output.

```
IIIN CAPTER 3, WE'LL SHOW YOU HOW TO USE THE MORE COMPLEX SPEED <CR>
COMMANDS. CHANCES ARE THAT ONCE YOU RECAME COMPLETELY FAMILIAR <CR>
WITH THE COMMANDS IN THE SAMPLE SESSION, THE OTHERS WILL BE MUCH <CR>
EASIER FOR YOU TO UNDERSTAND.$$
```

Figure 2.2 Second page of sample text DG-09492

Get a File for Reading (GR)

In the remainder of the practice session, you reexamine the contents of SAMPLE. To reexamine and modify an existing file, use the GR command. The general form of the GR command is:

```
!GRfilename$
```

This command opens an existing file, *filename*, but does not read text into the buffer. To read in the first page, you must issue a Y command (see the following section). Remember that to store modified text, you must also open an output file (with the GW command) as well as an input file (with the GR command). For example, to open an output file EXAMPLE along with input file SAMPLE, type:

```
!GRSAMPLE$GWEXAMPLE$$
```

GR opens file SAMPLE as the input file (for reading and modification) and GW opens file EXAMPLE as the output file.

Now you are ready to edit the text with the Search (S), Type-out (T), Change (C), Delete (D), Kill (K), and Put (P) commands described in the next sections.

Search (S)

Speed provides four search commands. Search commands look for a particular string and place the CP after that string when Speed finds it. In this editing session, we use the S and C commands, which search only the current buffer. The other search commands, N and Q, can cross page and buffer boundaries to search through all text not yet output. Chapter 3 describes the N and Q commands.

The general form of the S command is:

```
Stext$
```

Speed begins searching for the specified string at the present CP position. Then it repositions the CP immediately after the last character of the first occurrence of the specified string. If Speed does not find the text before it reaches the end of the buffer, your console displays the error message:

ERROR:UNSUCCESSFUL SEARCH

In this event, Speed positions the CP at the beginning of the buffer and your console displays the Speed prompt.

You can correct the first error in the text, using three of the commands we have discussed. To insert an S at the end of GUIDE, type

```
!JSGUIDE$IS$$
```

J instructs Speed to position the CP at the beginning of the buffer and S instructs Speed to search for the string GUIDE. When Speed finds the string, it places the CP after E and inserts an S. (Note that you separate search commands from other commands with a single ESC.) Then Speed positions the CP after the inserted S.

You can precede the S command with numeric arguments, which allow you to search the buffer selectively.

See Table 2.2 for a list of S commands.

Command String	Description
S*text*$	Searches for a particular string and places the CP after that string.
*n*S*text*$	Searches from the CP through the next *n* Carriage Returns. If the search fails, Speed positions the CP at the beginning of the line following the last line searched.
-*n*Stext$	Searches starting at the *n*th line preceding the current line and ends at the current position of the CP. If the search fails, the CP position remains the same and Speed prints the error message.
0Stext$	Searches for text from the beginning of the current line to the position of the CP. If the string is not found, the CP remains in the original position.
m,*n*Stext$	Searches for text from the (*m* + 1) through the *n*th characters of the buffer. If Speed finds the string, it positions the CP after the last character of the string. If Speed does not find the string, it positions the CP after the *n*th character and prints the error message: *ERROR: UNSUCCESSFUL SEARCH*

Table 2.2 The S command

A few examples of the S commands shown in the table follow. The examples refer to the text in Figure 2.1

Using 2SPROCESS$$ searches two lines after the CP, finds the string, and places the CP after the final S in PROCESS.

Using -1SCHAPTER$$ searches for CHAPTER one line before the CP and positions the CP after the R.

Using 0SING searches for ING and positions the CP after the G of FOLLOWING.

Using 13,150SPROMPT causes an error message because PROMPT is more than 150 characters into the buffer. Speed could not find it, so it positions the CP after the 150th character in the buffer (before the P of POSSIBLE in line 3).

Type-out (T)

A major function of the Type-out (T) command is to type out the current line of text that includes the CP. For instance, the last example in Search places the CP after the 150th character in the buffer. To find the 150th character, type:

```
!T$$
```

Your console displays:

PAINLESS AS ↑ POSSIBLE, THE TEXT YOU ARE READING NOW WILL SERVE AS THE SAM-

If you precede the T command with a number sign (#), Speed types out the entire buffer. To try it, type:

```
!#T$$
```

Your console displays the entire first page of SAMPLE.

Table 2.3 lists all of the T commands. Practice changing the CP position with various search commands and try each T command until you have completely mastered both the S and T functions. Remember that the T commands do not affect the location of the CP.

Mnemonic	Description
T	Types out a line.
:T	Types the text on the line printer instead of the terminal. It can precede any form of the T command.
0T	Types the current line from its beginning to the location of the CP.
*n*T	Types the contents of the current buffer from the location of the CP through the next *n* Carriage Returns.
-*n*T	Types the contents of the *n* lines preceding the current line, plus the contents of the current line up to the CP.
m,*n*T	Types the contents of the current buffer from the (*m* + 1) character (measured from the first character in the buffer) up to and including the *n*th character.
#T	Types out the entire buffer.

Table 2.3 The T command

Change (C)

The C command is a combination search-and-modify command. The general form is:

```
Ctext1$text2$
```

Speed searches for *text1* from the CP position to the end of the buffer. If it finds *text1*, Speed deletes it and inserts *text2* in its place, and positions the CP immediately after the last character in *text2*.

If Speed does not find *text1*, your console displays this error message:

ERROR:UNSUCCESSFUL SEARCH

and positions the CP at the beginning of the buffer. For example, type:

```
!JCGUIDES YOU THROUGH $DEMONSTRATES$T$$
```

Speed positions the CP at the beginning of the buffer, then searches for GUIDES YOU THROUGH. Speed finds the string, inserts DEMONSTRATES in its place, and types out the edited line:

THE FOLLOWING CHAPTER DEMONSTRATES ↑ A SAMPLE EDITING CESSION.

After you make a correction, it is useful to type out the line to check that the text reads the way you want it to. We correct the second error (CESSION) in a later example.

Table 2.4 lists the C command and its variations.

Command String	Description
C*text1*$*text2*$	Searches for *text1* and replaces it with *text2*.
*n*C*text1*$*text2*$	Searches for *text1* from the CP through the next *n* carriage returns and replaces it with *text2*. If the search fails, Speed positions the CP at the beginning of the line following the last line searched.
−*n*C*text1*$*text2*$	Searches for *text1*, starting at the beginning of the *n*th line preceding the current line, ends at the current position of the CP, and replaces it with *text2*. If the search fails, the CP position remains the same, and Speed prints an error message.
0C*text1*$*text2*$	Searches for *text1* from the beginning of the current line to the position of the CP. If it finds *text1*, Speed replaces it with *text2*. If Speed does not find *text1*, the CP remains in its original position and you get the normal error message.
*m,n*C*text1*$*text2*$	Searches for the ($m+1$) through the *n*th characters of the buffer. If Speed finds the string, it replaces it with *text2* and positions the CP after the last character in *text2*. If it does not find *text1*, Speed positions the CP after the *n*th character and outputs an error message.

Table 2.4 The C command

Delete Characters (D)

Use the Delete (D) command to delete a number of characters before or after the CP position. The general form of the D command is:

```
nD$
```

An *n* greater than 0 deletes *n* characters to the right of the CP. An *n* less than 0 deletes *n* characters to the left of the CP. For example, to delete the extra space in DI SPLAYS in line 7 of the sample text, type:

```
!SDI$1D$T$$
```

Speed searches for the first occurrence of DI and positions the CP before the space. Then Speed deletes the space and types out the edited line.

Your console displays:

```
AND IT DI↑SPLAYS A PROMPT, YOU ARE READY TO ENTER
THE FIRST COMMAND.
```

If you type this command string:

```
!SCESSION$-7M$1D$IS$T$$
```

Your console displays:

```
ERROR:UNSUCCESSFUL SEARCH !
```

This demonstrates what happens if you fail to position the CP at the beginning of the buffer. In this example, Speed could not find the string. It printed the error message and positioned the CP at the beginning of the buffer. If you type this command string:

```
!JSCESSION$-7M$1D$IS$T$$
```

your console displays:

```
THE FOLLOWING CHAPTER DEMONSTRATES A SAMPLE EDIT-
ING S↑ESSION.
```

An alternate way to edit the same text uses the J and C commands, as follows:

```
!JCCESSION$SESSION$T$$
```

When you use the C command, be certain that the string you wish to change is unique. One of the most common Speed errors is to change valid text within correctly entered search commands. For example, to change VAN to CAN use the L and C commands:

```
!4LCVAN$CAN$T$$
```

Your console displays:

```
TOWARD THIS TEXT, ERRORS THAT YOU SEE HERE ARE IN-
TENTIONAL AND CAN BE
```

You can use an unlimited number of commands in any command string as long as you separate them with a single Escape. Speed carries out the commands from left to right. To continue making several changes with one command string, you can nest three Change commands (divided from each other with a single Escape) as follows:

```
!1LCCOREE$CORRE$CTERN$TERM$T$COPWN$ OPEN$T$$
```

Your console displays:

```
CORRECTED BY SIMPLE COMMANDS FROM THE TERMINAL
ONCE SPEED HAS BEEN LOADED.

IN ONE EASY LESSON, YOU'LL LEARN HOW TO OPEN AND CLOSE
FILES, INSERT TEXT.
```

When you finish correcting the sample text, type:

```
!#T$$
```

Your console displays the buffer contents, and you can proofread them before you put them into the file SAMPLE.

Put (P)

To save your text in SAMPLE, you must use a command that puts (writes) the contents of the current buffer into SAMPLE. The P command writes the entire buffer contents to an output file, terminating them with a New Page character (CTRL-L). To put the entire buffer contents out to the output file, SAMPLE, with the New Page character, type:

```
P$$
```

If you wish to omit the concluding New Page character, use the command

```
PW$
```

You can precede P or PW with a numeric argument using any general format shown below:

*n*P$ or -*n*P$
*n*PW$ or -*n*PW$

If you use a numeric argument, Speed starts at the CP and outputs *n* lines from the CP or -*n* lines before the CP to the output file.

The P and PW commands do not delete characters from the buffer after output. You can add certain modifications, discussed in Chapter 3, to the P commands to clear the buffer automatically. During this session, however, we do not use the shortcuts.

Delete (Kill) Lines (K)

Under most circumstances, you want to clear the buffer after you have sent it to the output file. Otherwise, the buffer imprint remains and is scrambled with any succeeding insertions. To delete a number of lines in a buffer, use the Kill (K) command which has this general format:

*n*K$

n tells the number of lines you want to delete. If you precede the K command with a pound sign (#), Speed deletes all of the lines in the buffer.

Since we have already typed the P command, it is safe to clear the buffer.Type:

```
!#K$$
```

Check to be sure that the buffer is empty. Type:

```
!#T$$
```

Your console displays the Speed prompt only.

To delete a string of characters from the current buffer, use *m*,*n*K. For example, suppose you have the following text:

HOW DO YOU LIKE THIS MANUAL? IS IT EASY TO READ? IF YOU HAVE ANY SUGGESTIONS, PLEASE FILL OUT THE FORM ON THE LAST PAGE.

With the CP at the beginning of the first line, to delete the word HOW, type:

```
!0,3K$T$$
```

The console displays:

DO YOU LIKE THIS MANUAL? IS IT EASY TO READ?

Table 3.7 in Chapter 3 lists K and D command variations.

Figure 2.3 on the following page shows the first practice text, the locations of changes originally needed, and the commands used to make the changes. Continue to make the needed corrections.

Yank a Page or Window into the Buffer (Y)

Once you have opened a file for input, use the Y command to read text from that file into the edit buffer. You can read text into the buffer in units of pages or windows. Chapter 3 describes specific page and window commands.

To yank the first page of SAMPLE into your edit buffer and display it on your terminal, type:

```
!YT$$
```

The text should now be error-free.

When Speed executes a Y command, it clears the current buffer before reading the next page or window. If there are no succeeding pages, Y still clears the buffer and positions the CP at the beginning of an empty buffer. Therefore, if you issue a Y command without first sending the contents of the buffer to the output file, you lose the entire contents of that buffer.

To write the first page of SAMPLE to a file called EXAMPLE and to read in the second page, type:

```
!PY#T$$
```

```
THE FOLLOWING CHAPTER GUIDE [1] YOU THROUGH A SAMPLE EDITING CESSION [2] <CR>
UNDER SPEED. SINCE OUR INTENTION IS TO MAKE THIS PROCESS AS SIMPLE AND <CR>
PAINLESS AS POSSIBLE, THE TEXT YOU ARE NOW READING WILL SERVE AS THE <CR>
SAMPLE PROGRAM. ALL COMMANDS IN THE SAMPLE EDITING SESSION ARE DIRECTED <CR>
TOWARD THIS TEXT. ERRORS THAT YOU SEE HERE ARE INTENTIONAL AND VAN[3] BE <CR>
CORREECTED[4] BY SIMPLE COMMANDS FROM THE TERNINAL[5]. ONCE SPEED HAS BEEN <CR>
LOADED AND IT DI SPLAYS[6] A PROMPT, YOU ARE READY TO ENTER THE FIRST COMMAND. <CR>
THIS CHAPTER DISCUSSES THE RUDIMENTARY COMMANDS THAT YOU'LL USE MOST <CR>
OFTEN. IN ONE EASY LESSON, YOU'LL LEARN HOW TO OPWN[7] AND CLOSE FILES <CR>
INSERT TEXT[8], CORRECT TYPING ERRORS, MANIPULATE THE CHARACTER <CR>
POINTER, AND EXIT FROM SPEED. <CR>
[1]CGUIDE$GUIDE$
[2]CCESSION$SESSION$
[3]CVAN$CAN$
[4]CORREECTED$CORRECTED$
[5]CTERNINAL$TERMINAL$
[6] 1L$SDI$1D$
[7]COPWN$OPEN$
[8]STEXT,$-5D$$
```

Figure 2.3 Sample session and commands

DG-09492

Make any additions or corrections, output the second page, and close the output file by typing:

```
!PGC$$
```

Then, to close the input file, type:

```
!GR$$
```

Before you leave Speed you should type a Global Close command:

```
!GC$$
```

Now you are ready to leave Speed and return to the RDOS CLI.

Home (H)

To leave Speed, type:

```
!H$$
```

The H command closes any open files before returning to the CLI. Speed stops, and you get the RDOS CLI prompt, *R*. At the RDOS prompt, you can only issue CLI commands. Remember that you should issue a Global Close command (GC) before you issue the H command.

This ends the sample editing session. Most of the commands that we describe in Chapter 3 are logical extensions of the commands you already know. If you have questions or doubts about any of the commands outlined in this chapter, go back and work with them, trying them on SAMPLE or on a new file of your own. If you are satisfied with your mastery level at this point, go on to Chapter 3.

Chapter 3

Speed Commands

The Speed commands described in this chapter give additional flexibility and versatility to your editing procedures. They allow you to edit, split, merge, and rearrange your files. If you receive an error message while editing, refer to Chapter 6 for an explanation of error handling.

Numeric Arguments

Numeric arguments may precede some of the commands described in this chapter. Speed always evaluates these arguments to an integer value before executing the command. A numeric argument may be a number or a numeric expression. Numeric expressions may contain:

any digit
any arithmetic operator listed in Table 3.1
any variable

Speed evaluates arithmetic operators from left to right.

Operators	Function
+	Addition
–	Subtraction
*	Multiplication
/	Division

Table 3.1 Arithmetic operators

In this manual we represent single numeric arguments by *n* and double numeric arguments by *m,n*. You must always separate double numeric arguments with a comma and *m* must always be less than or equal to *n*. For example, type:

```
!7,60T$$
```

Speed displays the 8th through the 60th character in the current buffer. If you omit the numeric argument *n* from a command, Speed assumes that it is 0.

Numeric Variables and Commands

Speed contains 10 numeric integer variables (named V0 through V9) that you can use either alone or as numeric arguments to commands. Each of the 10 variables is initially set to 0. You can set each one to a value, or increment or decrement it. When you use a numeric variable as an argument to a command, the variable returns an operand to the next command. For example, if V0 is equal to one, then the following command string:

```
!V0+2D$$
```

deletes three characters following the current position of the Character Pointer (CP).

You may set a variable *v* to value *n* and then use that value with a command. For example:

```
!5VS1$V1+1T$$
```

5VS1 sets V1 to 5: V1 + 1T types out six lines. Table 3.2 describes the different forms of numeric variable commands.

Mnemonic	Description
V*v*=	Represents the current value of variable *v*.
VD*v*	Decrements variable *v* and represents the decremented value.
VI*v*	Increments variable *v* and represents the incremented value.
*n*VS*v*	Sets variable *v* to value *n* and returns that value. You cannot use this command with arithmetic operators in a numeric expression except as the first operand.

Table 3.2 Numeric variable commands

Page and Window Mode

While editing in Speed, you can read text from your file into the buffer in either page or window segments. Page mode is the default mode. If you select window mode, you must specify the window length you desire. You can change the mode at any time during the editing process. Table 3.3 contains a description of page and window mode commands.

Mnemonic	Description
*n*WM	Changes the input mode from page mode to window mode with a window length of *n* lines. (Speed is initially in page mode.)
0WM	Changes the input mode from window mode to page mode.
WM	Returns the value of the data input mode. If WM=0, Speed is in page mode. If WM=*n* (*n* is greater than 0), Speed is in window mode with a window length of *n* lines.

Table 3.3 Page and window mode commands

When you are using page mode, you can access the entire page contained in the buffer, but you can actually view only as much of the page as will fit onto the console display. Therefore, it may be to your advantage to use window mode on long files or files with many CTRL-Ls (also called Form Feeds). Normally, if you are using window mode, you should specify a number of lines that the console can display at once; this allows you to edit text one window length at a time.

In window mode when you insert New Page (Form Feed) characters into the text with a P command, you see CRTL-Ls printed at the appropriate places within the text. You may insert or delete Form Feeds (CTRL-Ls) in window mode.

In page mode, you do not see CTRL-L characters. Speed remembers when the last character read by an input command was a CTRL-L, but it does not place that character into the edit buffer. In page mode you may delete a CTRL-L with the Append (A) command. Append (A) reads a page from the input file and appends it to the current edit buffer.

Examples
To specify a window length of 25 lines, type 25WM$$.

To change input mode from window mode to page mode, type OWM$$.

To put Speed in window mode with a window length of 25 lines, type WM = 25.

Commands to Open and Close Files

You can specify a file as global or local when you open it. If a file is global, you can read or write to it from any of the 36 available edit buffers. If the file is local, it is an input or output file that you open for one particular edit buffer. Only one global input and one global output file can be open at a time.

You may open the same input file as a local file in two or more buffers. You may open the same input file both as a global or local file. This enables you to position different buffers to different places in the same file, a useful feature when you need to rearrange the order of an existing file. Local files are most useful when you want to append parts of separate input files to any output file.

All local file commands begin with the letter B (for example, BGR and BUY). When you open a local file, all commands apply to the local file. For all file input and output commands, a local file *always* takes precedence over a global file. The file is global when you do not precede the file's command with the letter B. See Table 3.10 for a complete list of local file commands.

Many of the commands we discuss in this chapter incorporate the functions of commands (such as GR, GW, and GC) described in Chapter 2. Now we show you commands that combine several functions. We start with variations on the U command.

The UY Command

The UY command opens a file for input (GR), yanks the first page or window from the input file to the edit buffer (Y), and creates a new file for output. The new output file has the same name as the input file, but with the extension .SC appended to it. The .SC file is a temporary scratch file to which you do not have access. The general form of the UY command is:

UY*filename*$

For example,

```
!UYSAMPLE$$
```

opens file SAMPLE for input, creates and opens SAMPLE.SC for output, and yanks the first text page of SAMPLE into the edit buffer. When you open a file with a UY command, you must close the file with either the US or UE command (see below).

You may type the filename immediately after the Speed command in the CLI and omit the UY*filename*$ command. For example, if you start Speed with:

```
R
SPEED SAMPLE ⟨CR⟩
```

that command is equivalent to:

```
R
SPEED ⟨CR⟩
!UYSAMPLE$$
```

The US Command

Use the US command to create a backup file of your original input file and a new, edited file with your original input file's name. The US command completes the following steps:

- Deletes the previous .BU file, if any.
- Renames the input file to *filename*.BU.
- Copies the contents of the current buffer, along with the remainder of the input file, to the output file.
- Closes the input and output files.
- Renames the output file to *filename* (the name of the original input file).

The general form of the US command is:

```
!US$
```

For example, assume you have opened a file for input and yanked the first page or window from the input file to the edit buffer with UYSAMPLE$$. After making modifications to the contents of SAMPLE, type:

```
!US$$
```

US accomplishes these tasks:

- Deletes any previous SAMPLE.BU file.
- Renames SAMPLE to SAMPLE.BU.
- Copies the rest of SAMPLE to SAMPLE.SC.
- Closes SAMPLE and SAMPLE.SC.
- Renames SAMPLE.SC to SAMPLE.

The UE Command

The UE command:

- Copies the current buffer and the remaining input file to the output file.
- Closes the input and output files.

If you opened the input and output files with a UY command, UE also:

- Deletes the input file.
- Renames the output file to the input filename.

The general form of the UE command is:

UE*filename*$

For example, assume you open file SAMPLE with UYSAMPLE$$, make modifications to its contents, and then close it with:

```
UE$$
```

UE accomplishes these tasks:

- Copies the current buffer and the rest of SAMPLE to SAMPLE.SC.
- Closes SAMPLE and SAMPLE.SC.
- Deletes SAMPLE.
- Renames SAMPLE.SC to SAMPLE.

The U? Command

The U? command is the file status command. It lists open global and local input and output files. It lists local files which are relevant to the current buffer. The form of the U? command is:

```
U?$
```

For example, with the command:

```
UYSAMPLE$U?$$
```

your console displays the following information:

```
GLOBAL:
Input file - SAMPLE
Output file -SAMPLE.SC

LOCAL:
Input file - None
Output file - None
```

File Input Commands

Use the file input commands described in Table 3.4 to read from the input file into the current buffer. Speed reads data in either page or window lengths.

Mnemonic	Command Description
A	Appends a page or window length from the input file to the current buffer. The A command has no effect on the CP. In page mode, A deletes CTRL-L between pages.
Y	Clears the current buffer, then reads a page or window length into the buffer. The CP is positioned at the beginning of the buffer. See Figure 2.1.

Table 3.4 File input commands

Insertion Commands

All insertion commands put a string of characters, specified by your command line, into the current buffer at the position of the Character Pointer (CP). After Speed inserts the text, it repositions the CP to follow the last character of the insertion. The simplest form of the I command, I*text*$, is documented in Chapter 3. Table 3.5 lists the remaining insertion commands.

Mnemonic	Command Description
I*text*	Enters *text* (a string of characters).
CTRL-I	Inserts tabs.
ICTRL-ECTRL- (any control character)	Inserts both characters into the line.
CTRL-I*text*	Inserts a tab preceding the text string.
@CTRL-I*text* CTRL-I	Inserts a tab preceding the text string, delimited by CTRL-I.
@I%*text*%	Enters text delimited by the first character after I (in this case, %). The @ modifier changes the command delimiter from ESC ($) to any other character you wish. For a detailed description of the @ modifier, see Special Commands and Command Modifiers in this chapter.
*n*I	Inserts the decimal equivalent (*n*) of any ASCII character listed in Appendix A. Speed does not accept certain ASCII characters as text in an inserted string. For instance, if you try to insert certain control characters (for example, CTRL-B) into a text string, the control characters signal Speed to perform a function rather than to appear as a character. The *n*I command inserts a single character *n*, where *n* is.
n \	Inserts the ASCII representation of the decimal *n* into the buffer at the CP location, suppressing leading zeroes. Use this command in iteration commands to complement numeric variable commands. See Chapter 4 for an example of how to use this command.

Table 3.5 Insertion commands

Character Pointer and Text Type-out Commands

For a complete description of Character Pointer (CP) and Text Type-out commands, see Tables 2.1 and 2.3, respectively, in Chapter 2.

Search Commands

There are four major search commands: S, C, N, and Q. For each of these, Speed scans the buffer or input file until it finds the exact string of characters you specified. If it finds the string, Speed positions the CP after the last character in the string.

For explanations of the Search (S) and Search and Change (C) commands, see Chapter 2, Tables 2.2 and 2.4, respectively. The Nonstop Search (N) and Quick Search (Q) commands are documented in this chapter.

The N*text* Command

N*text* searches the current buffer for *text*. If it does not find *text*, Speed outputs the current buffer, clears the buffer, yanks in the next page or window length from the input file, and continues the search until it finds *text* or reaches an end of file. If the search is unsuccessful, Speed prints the error message:

```
ERROR:UNSUCCESSFUL SEARCH
```

and positions the CP at the beginning of an empty buffer.

For an example of the N command, return to SAMPLE, our file from the Sample Editing Session. Suppose that you missed the first error on page 2, "CAPTER." Type:

```
!UYSAMPLE$$
!NCAPTER$-5MIH$T$$
```

Speed searches the first page of SAMPLE for CAPTER. Because it cannot find the string, it outputs the first page to SAMPLE.SC and clears the buffer. Speed then yanks in the second page of SAMPLE, finds CAPTER, and positions the CP after the R. The next command (note the ESC delimiter) tells Speed to backup five spaces, insert an H, and type out the line.

Your console displays:

```
IN CH ↑ APTER 4, WE'LL SHOW YOU HOW TO USE THE MORE
COMPLEX SPEED
```

Close the file with a UE command.

The Q*text* Command

Use this command if you access a file with a GR command and you want to output text that resides late in a file. The Q*text* command is similar to the N*text* command except that when Speed does not find text, it executes a Y command, which clears (and scraps) the current buffer and then yanks in the next page. (When Q*text* fails to find text, it does *not* output the buffer.) Do not use this command if you opened the file with a U command and want to save any part of the input file up to *text*.

Using the same example as for N*text*, assume that you want an output file containing only the second page of SAMPLE. Type:

```
!GRSAMPLE$$
!QCAPTER$-5MIH$$
!GWQEXAMPLE$$
!PGC$GR$$
```

In this example, we open the file SAMPLE with a GR command. GR does not create an output file, and the input file, SAMPLE, remains intact no matter what command we issue. Speed searches the first page, without finding CAPTER, so it clears the buffer, yanks in the second page, and carries out the command. We then open the output file, QEXAMPLE, with GW and put the contents of the buffer into that file with P. Finally, we close the output file QEXAMPLE with GC and the input file SAMPLE with GR.

Search Command Modifiers

The @ modifier changes the command delimiter from ESC ($) to any other character you specify (for example, %). You can use this modifier with any S, C, N, or Q command, and it is particularly useful if the string you are searching for contains an Escape character. For example:

```
!@Stext$%
```

is the same as

```
!Stext
```

except that the first character after S (% here) delimits the text so that you can include the ESC ($) in the search.

You can use the colon (:) modifier with output commands P, R , and nR. The :P command is a modified form of the P command, equivalent to P#K$$. It deletes all characters in the current buffer after output. You can use the : modifier with all P and PW variations.

The :R and :*n*R commands are modified forms of R and *n*R. Each returns an operand to the next command; it returns +1 if input is successful and 0 if input fails. No error message appears if the read is unsuccessful.

You can use :R or :*n*R as a numeric argument to the next command. For example:

```
!:RStext$10T$$
```

If the read is successful, Speed searches one line for text and types out the following 10 lines. If the read fails, Speed prints nothing, and the CP resides at the beginning of an empty buffer.

You can also use this modifier as a numeric argument with any S, C, N, or Q command. No error message appears if the search is unsuccessful. In most cases, you will want to use the colon modifier in extended command lines or macro commands. For example:

```
!:Stext$*5T$$
```

If the search is successful, Speed returns a +1 operand to the next command and types five lines from the current position of the CP. If the search is unsuccessful, however, Speed returns a 0 to the *5T command. Since the product of 0*5T equals 0 lines, Speed ignores the T command and proceeds to any further commands in the command string.

You can combine the @ and colon command modifiers; for example, the following command strings are legal:

```
!@:S%text%
```

```
!:@C%text1%text2%
```

See Table 3.9 for a list of colon modifiers used in output commands.

Control Characters in Searches

You can include any of several control characters in a Search command to alter the normal search process.

Using the CTRL-Z character in a Search command string indicates that the position occupied by the CTRL-Z is unimportant, and any character is an acceptable match. For example:

!SAB**CTRL-Z**DE$ matches the string ABCDE, ABBDE, etc.

Using a CTRL-T character in a Search command string matches zero or more Tab or space characters in the buffer. For example:

!SABC**CTRL-T**YZ$ matches the string ABC"""""""XYZ.

Using a CTRL-N*x* in a Search command string indicates that any character is acceptable in the *x* position except for the character *x*. For example:

!SABC**CTRL-NE**E matches the string ABCDE.

!SABC**CTRL-ND**E does not match the string ABCDE.

Using a CTRL-V*x* indicates that any number of *x* characters, including 0, is acceptable in the position occupied by V*x*. For example:

1SABC**CTRL-VD**E matches the strings ABCE, ABCDE, and ABCDDE.

Using CTRL-P in a Search command string positions the CP. If the search is unsuccessful, Speed positions the CP where CTRL-P was in the Search command string, rather than at the end of the matching string. For example:

```
!SNOW IS THE TIME FOR ALL CTRL-P GOOD$T$$
```

The console displays:

```
NOW IS THE TIME FOR ALL ↑ GOOD
```

Using CTRL-E with another control character tells Speed to interpret the character following CTRL- literally, rather than as a special character. You can use this control character to search for other control characters, like CTRL-Z or CTRL-P.

These characters are listed in Table 3.6.

Mnemonic	Command Description
CTRL-E *any control character*	Tells Speed to interpret the control character literally rather than as a special character. Use this control character to conduct a search for other control characters.
CTRL-N*x*	Indicates that any character is acceptable in the *x* position except for the character *x*.
CTRL-P	Positions the Character Pointer (CP). If the search is successful, Speed positions the CP where CTRL-P was in the search string, rather than at the end of the matching string.
CTRL-T	Matches zero or more tab or space characters in the buffer.
CTRL-V*x*	Indicates that any number of *x* characters, including 0, is acceptable in the position occupied by V*x*.
CTRL-Z	Indicates that the position occupied by the CTRL-Z is unimportant, and any character is an acceptable match.

Table 3.6 Control characters in searches

Speed flags any control character in a Search string (except the ASCII representations for Tab (CTRL-I), Carriage Return (CTRL-M), New Page (CTRL-L), Line Feed (CTRL-J), and Vertical Tab (CTRL-K) as an error unless you precede it with a control (CTRL-E). Refer to Table 3.5 for commands that insert ASCII representations.

Deletion Commands

Use the commands described in Table 3.7 to delete characters and lines from the edit buffer. Examples appear in Chapter 2.

Mnemonic	Command Description
D	Deletes a character.
*n*D	Deletes *n* characters to the right of the CP.
−*n*D	Deletes *n* characters to the left of the CP.
K or 0K	Deletes the characters from the CP back to the beginning of the line containing the CP.
*n*K	Deletes the lines in the current buffer from the CP up to and including the next *n* Carriage Returns.
−*n*K	Deletes *n* lines preceding the current line plus the characters up to the CP in the current line.
m,*n*K	Deletes the $(m+1)$ through *n*th characters from the current buffer. Speed positions the CP after the *n*th character.
#K	Deletes all the lines in the current buffer. See Chapter 2.

Table 3.7 Deletion commands D and K

Output Commands

To replace an entire buffer, it is also possible to use the R, *n*R, and E commands.

Using R copies the current buffer to the output file, clears the buffer, and yanks in another page or window length from the input file. If no succeeding pages or window lengths exist, Speed returns the error message:

ERROR: NO MORE CHARACTERS IN INPUT FILE.

You have already output the current buffer, so nothing is lost. The R command is equivalent to the PY command sequence.

Using *n*R performs an R command *n* times.

Using E copies the current buffer and the remainder of the input file to the output file. Speed copies any Form Feeds detected on input to the output file and clears the current buffer. For example, assume that you have opened a file consisting of eight pages, and you wish to correct an error on page 3, leaving the rest of the file intact.

```
!UYSINOZ$2R$$

.
. (modifications)
.

!E$$
```

UY opens SINOZ and yanks in the first page. 2R copies the first and second page of SINOZ to SINOZ.SC, and yanks in page 3.

After modifying the page, E$$ copies the current page, as well as pages 4 to 8 to SINOZ.SC, positioning the CP at the beginning of an empty page 9.

The commands listed in Table 3.8 output data from edit buffers and input files. Output commands do not affect the position of the CP.

NOTE: The P and PW commands do not clear the buffer after output unless you include the colon (:) modifier.

Mnemonic	Command Description
Appended Form Feed Character (CTRL-L)	
P	Writes (but does not clear) the entire contents of the current buffer to the output file, with an appended CTRL-L. See Chapter 3.
*n*P	Writes *n* lines from the position of the Character Pointer in the current buffer, with an appended CTRL-L character.
−*n*P	Writes the preceding *n* lines plus the characters up to the location of the CP in the current line, and appends a CRTL-L character.
0P	Writes the characters on the current line from the beginning of the line to the location of the CP, and appends a CTRL-L character.
m,*n*P	Writes the (*m*+1) to *n*th characters from the current buffer, and appends a CTRL-L.

Table 3.8 Output commands

Mnemonic	Command Description
Without Form Feed Character (CTRL-L)	
PW	Writes the entire contents of the current buffer to the output file without appending a CTRL-L.
*n*PW	Writes (but does not clear) the contents of the current buffer to the output file without a CTRL-L.
−*n*PW	Writes (but does not clear) the preceding *n* lines plus the characters up to the CP without a CTRL-L.
0PW	Writes the current line up to the location of the CP to the output file without a CTRL-L.
m,*n*PW	Writes the *m*+1 to *n* characters to the output file without a CTRL-L.
Replace Entire Buffer	
E	Writes the current buffer and the remainder of the input file to the output file. Speed copies any CTRL-L to the output file and clears the current buffer.
R	Writes the current buffer to the output file, clears the buffer, and yanks in another page or window length from the input file.
*n*R	Performs an *R* command n times.
Colon Modifier in Output Commands	
:P	Deletes (clears) all characters in the current buffer after output. It is a modified form of the P command, equivalent to PK$$. You can use the colon modifier with all P and PW variations.
:R and :nR	Returns an operand to the next command; one if input is successful and zero if input fails. No error message appears if the read is unsuccessful. You can use :R or :*n*R as a numeric argument to the next command.

Table 3.8 Output commands (continued)

Buffer Commands

Speed lets you use up to 36 edit buffers, coded 0 through 9 and A through Z, to hold text. You can specify only one buffer as the current buffer at any time, but you can change the current buffer by issuing a buffer command. Initially, buffer 0 is the current buffer.

The B? command displays your buffer status. Your console lists all active buffers and their lengths in characters. A right angle bracket (〉) indicates which buffer is current. For example, type:

```
!B?$$
```

Your console displays:

```
 BUFFER   0-3977
 BUFFER   2-36
〉BUFFER   A-4662
 BUFFER   G-1903
```

Buffer A is the current buffer.

Using B?*x* displays the status of a specific buffer. The variable *x* is a single-character name that you specify for any one of the 36 edit buffers.

The BA*x* command activates buffer *x*. All buffers except the current buffer are stored on disk unless you delete them with a BK*x* command. You can recover all of the active buffers in subsequent editing sessions by issuing the BA*x* command. The following example will only work if there is no input file. If you have just finished an editing session on a large section of a file in buffer 3, your alternatives for sending the material to the output file are as follows:

1. To access only the text contained in buffer 3, issue a UE or US command. Both commands copy all of buffer 3 to the output file, clear the buffer, and close the input and output files. The US command also creates a backup file from buffer 3, the input file. Note that after either command, you could reopen the file using UY or GR and be at the beginning of an empty buffer 3. To return to another buffer, use a buffer select BS*x* command.

2. If the text in buffer 3 is part of a larger file, issue a P or PW command and then return to your primary buffer. The text in buffer 3 remains intact, though a copy is placed in your output file. You could finish editing in buffer 0, close out the file with a UE or US, and return to the CLI. So far, the instructions in our second example look like this on your terminal:

```
!UYEXAMPLE4$B?$$
〉BUFFER 0-4077
BUFFER 3-3011
!BS3$$
.
.(modifications)
.
!P$$
!BS0$$
.
.(modifications)
.
!UEH$$
```

Assume that you then want to return to Speed from the CLI, to edit the original contents of buffer 0.

```
R
SPEED EXAMPLE4 〈CR〉
!B?$$
〉BUFFER 0-3011
```

Because you have output the contents of buffer 3 with a P command, you now see only the first page of buffer 0 (the original contents of buffer 3). To access the second page, type:

```
!Y$$
```

Unfortunately, by issuing Y instead of R, you lost the first page of the file from buffer 0. In this case, however, the buffer can be retrieved. Type:

```
!BA3$B?$$
〉BUFFER 0-4077
BUFFER 3-3011
!BS3$P$BS0$$
```

Once again, you accessed buffer 3 (BS3), placed it in the output file, and returned to buffer 0 to complete your editing. Buffer 3 remains intact until you either issue a BK3 command or overwrite the buffer with other text.

Using BK*x* deactivates and deletes buffer *x*. Buffer *x* need not be the current buffer. For example:

```
!BS0$B?$$
〉BUFFER 0-3977
BUFFER 1-366
!BK1$B?$$
〉BUFFER 0-3977
```

From the current buffer 0, we delete buffer 1.

The BC*x* command copies the contents of the current buffer to buffer *x* after clearing buffer *x*. The *n*BC*x* command copies to buffer *x* the next *n* lines from the CP position in the current buffer. For example:

```
!BS3$5BCA$$
```

makes buffer 3 current (BS3). Then Speed clears buffer A and copies five lines from buffer 3 to buffer A. The contents and CP position of buffer 3 remain unchanged. Speed positions the CP at the beginning of buffer A.

The -*n*BC*x* clears buffer *x*, then copies to buffer *x* the *n* lines preceding the current line plus the characters on the current line up to the CP.

The *m*,*n*BC*x* command clears buffer *x*, and copies the (+ 1) to *n*th characters (inclusive) from the current buffer to buffer *x*. The contents and CP position of the current buffer remain unchanged. Speed places the CP at the beginning of buffer *x*.

The BS*x* command changes the current edit buffer to edit buffer *x*. Speed saves the status of the CP and of any open local files when it changes the current edit buffer; then it restores them for the new edit buffer. Commands such as Yank (Y) and Append (A) send data from the input file to whatever edit buffer is current.

The BT*x* command is the same as BC*x* except that Speed deletes the characters in the current buffer after moving them to buffer *x*. The *n*BT*X* command and -*n*BT*X* is the same as using *n*BC*x* and -*n*BC*x*, respectively, except that Speed deletes the characters in the current buffer after moving them to buffer *x*.

The *m*,*n*BTX command is the same as *m*,*n*BC*x* except that Speed deletes the characters in the current buffer after moving them to buffer *x*.

Use the commands listed in Table 3.9 to manipulate buffers under Speed.

Mnemonic	**Command Description**
B?	Lists all active buffers and their lengths (in characters). Uses a right angle bracket (⟩) to indicate which edit buffer is current.
B?*x*	Types out the status of buffer *x*.
BA*x*	Activates buffer *x*.
BC*x*	Clears buffer *x*, writes the contents of the current buffer to buffer *x*, and positions the CP at the beginning of buffer *x*.
*n*BC*x*	Clears buffer *x*, writes the next *n* lines to buffer *x* from the CP position in the current buffer, positions the CP at the beginning of buffer *x*.
−*n*BC*x*	Clears buffer *x*, writes to buffer *x* the *n* lines preceding the current line, plus the characters on the current line up to the CP, and positions the CP at the beginning of buffer *x*. The contents of the buffer and the position of the CP remain unchanged.
m,*n*BC*x*	Clears buffer *x*, writes $(m+1)$ to the *n*th characters from the current buffer to buffer *x*, and positions the CP at the beginning of buffer *x*. The contents of the current buffer remain unchanged.
BK*x*	Deactivates and deletes buffer *x*. Buffer *x* may not be the current buffer.
BS*x*	Changes the current edit buffer to edit buffer *x*.
BT*x*	Clears buffer *x*, writes the contents of the current buffer to buffer *x*, positions the CP at the beginning of buffer *x*, and deletes the characters in the current buffer after moving them to buffer *x*.
*n*BT*x*	Clears buffer *x*, writes the next *n* lines to buffer *x* from the CP position in the current buffer, positions the CP at the beginning of buffer *x*, and deletes the characters in the current buffer after moving them to buffer *x*.
−*n*BT*x*	Clears buffer *x*, writes to buffer *x* the *n* lines preceding the current line, plus the characters on the current line up to the CP, and positions the CP at the beginning of buffer *x*. Deletes the characters in the current buffer after moving them to buffer *x*.
m,*n*BT*x*	Clears buffer *x*, writes the $(m+1)$ to *n*th characters from the current buffer to buffer *x*. The contents of the current buffer remain unchanged. Deletes the characters in the current buffer after moving them to buffer *x*.

Table 3.9 Buffer commands

Local File Commands

When you open an input or output file, you have the option of specifying whether the file is local or global. A local file is an input or output file that you open for one particular edit buffer. You use local files when you want to access parts of other input files. All local file commands are file specification commands preceded by the letter B (for example, BGR and BUY). Therefore, if you do not precede the file specification command with the letter B, the file you specify is global. If you do open a local file, all your commands apply to the local file. Remember, then, to close any local file before you issue global commands. Table 3.10 contains local file commands.

NOTE: It is possible to have both global and local files open for the current buffer. In that case, all the commands apply to the local files.

Mnemonic	Command Description
BGR*[filename]*	Opens or closes local *filename* for the current edit buffer, in accordance with the respective GR, GW, GC, UY, UE, or US command. You can open only one local file for output for each buffer.
BGW*filename*	
BGC	
BUY*filename*	
BUE	
BUS	

Table 3.10 Local file commands

Status Information (U?)

When you type U?, Speed provides a status report of global and local input and output files for the current buffer. For example, using the following procedure, you can create an output file that contains text copied from several different edit buffers containing material from different input files. We assume that Speed is running on your system and that an input file is already open.

1. Change the current edit buffer to a different one (BS*x*).
2. Type the command to open an input file local to that buffer, BGR *filename*. Type in material and edit it.
3. Close the input file with the BGR command, then change the current edit buffer to another one. Repeat step 2.
4. Repeat steps 2 and 3 as many times as necessary, moving material into different edit buffers and closing the input files afterward.
5. Open a global output file (a file that accepts output from all buffers) with a GW command, unless you opened your input file with a UY command. If you opened your input files with UY, an output file with the extension .SC already exists.
6. Move material into the output file from the current edit buffer with output commands (P).
7. Change the current edit buffer to a different one and repeat step 5.
8. Repeat steps 5 and 6 as many times as necessary.
9. Close the output file, which now contains modified material from several edit buffers.

Any input or output commands (for example, Y and P) apply first to local files that are open. You must close a local file before you can access a global file with a particular command.

Command String Insertion Commands

Use these commands to insert the contents of a file or edit buffer into a command string.

The CTRL-G*filename* Command

If you include the CTRL-G*filename* command in a command string, Speed inserts the contents of *filename* into the command string. When Speed executes the command string, the entire contents of *filename* are inserted into the current buffer. The file may contain text you want to insert, commands you want to implement, or both, and must be less than 64 Kbytes long. For example, assume that file EXAMPLE2 contains a program or set of instructons that you can use as a macro command in other programs. In the following example, we insert the macro command at the beginning of the third line of TEST5.

```
!UYTEST5$$
```

```
!3JI**CTRL-G**EXAMPLE2$T$$
```

The CTRL-B*x* Command

If you include the CTRL-B*x* command in a command string, Speed inserts the contents of buffer *x* into the command string. When Speed executes the command string, it evaluates the entire contents of buffer *x* as part of the command string. Buffer *x* may contain text to be inserted, commands to be executed, or both. Buffer *x* cannot be the current buffer.

In the following example, assume that buffer A contains a string of text that you need to insert a number of times into file TEST6. Rather than repeat the insertion every time, use the B*x* command after positioning the CP where you want the insertion to begin.

```
!UYTEST6$B?$$
```

```
⟩BUFFER 0-3977
```

```
BUFFER A-1023
```

```
!JICTRL-BA$$
```

```
24JICTRL-BA$$
```

In this example, we inserted the contents of buffer A twice–once at the beginning of buffer 0 and once after the 24th line.

Nesting

You can nest the commands that CTRL-G*filename* and CTRL-B*x* insert into the command line up to 10 levels deep. However, the filename that you specify in the CTRL-G*filename* command cannot contain an imbedded CTRL-G*filename* command.

When you use CTRL-G*filename* or CTRL-B*x* as the object of a search, Speed treats the characters up to the first ESC ($) in the buffer or the file literally, as text, rather than as commands. For example, if buffer *x* contains:

```
CFRIDAY$$
```

and *filename* contains:

```
T$IWEDNESDAY$Z
```

then the command string:

```
!10⟨CTRL-Bx;⟩ #CTRL-Gfilename$ = $$
```

would be evaluated as:

```
10 ⟨CFRIDAY$$;⟩ #T$IWEDNESDAY$Z = $$
```

In this example, Speed evaluates the command line using the contents of buffer *x* and of file *filename* in place of CTRL-B*x* and CTRL-G*filename*, respectively. Note that while a double ESC sequence terminates input from the console, it does not terminate input from another source, such as a buffer or file.

Iteration Commands

You can execute a command string any number of times by placing the string between angle brackets and specifying the number of iterations with a numeric argument preceding the brackets. A command string within angle brackets is called a command loop. The general form of a command loop is:

n ⟨command string⟩

If *n* is less than or equal to zero, Speed skips the commands within the brackets.

If *n* is greater than zero, Speed repeats the commands *n* times. If you omit *n*, Speed repeats the commands indefinitely.

For example, when you type:

```
5⟨SDATE:$I□JUNE$⟩
```

Speed searches for five occurrences of DATE: and inserts □JUNE after each. If Speed finds less than five occurrences of DATE:, it positions the CP at the beginning of the buffer. You can use the iteration format when correcting repeated misspellings or when changing items used throughout the text. For example, when you type:

```
!10⟨CSIGNALLED$SIGNALED$⟩$$
```

Speed searches for 10 occurrences of SIGNALLED and changes them to the preferred spelling SIGNALED.

Speed treats Search commands and File Input commands (and R) within command loops as though they were modified by the colon modifier. That is, if the command fails, Speed returns a zero operand to the next command and does not print an error message. However, if the command is successful, it returns a + 1 operand to the next command, and command execution within the loop proceeds. The success or failure of the Search or File Input commands nested within command loops does not affect the execution of other commands within the command string.

You can nest command loops up to 10 levels deep.

Semicolon Command (;)

The semicolon command terminates a command loop under one of the following two conditions:

1. If the semicolon is preceded by an argument that is less than or equal to one.

2. If there is no preceding argument and the last search was unsuccessful.

In all other cases, command loop execution continues.

When Speed completes its first unsuccessful search, the semicolon command transfers control out of the current command loop to the command immediately following the loop. If you do not use the semicolon, Speed continues to execute the command loop indefinitely, and does not execute the commands which follow the loop.

In the previous example, we knew that there were five occurrences of the word DATE in the buffer. Suppose you are editing in a much larger buffer and have no idea how many times DATE (or any arbitrary string) occurs. For example,if you type:

```
!(SDATE$; IJUNE$)$$
```

Speed searches for all occurrences of DATE and, each time it finds DATE, inserts JUNE. When Speed finds the last occurrence of DATE in the buffer, it terminates the command loop and returns a prompt to your terminal. The CP is now at the beginning of the buffer. If you type:

```
!(CJUNE$JULY$;)T$$
```

Speed changes all occurrences of JUNE to JULY in the current buffer and then types out the first line in the buffer.

Use the semicolon command only in command loops. When you try to use it outside a command loop, Speed flags the semicolon as an error.

You can use the colon command modifier with the semicolon command. The colon command reverses the action of the semicolon command. That is, control transfers out of the command loop if the last search was successful. The general form of this command modifier is:

```
:;
```

In the example:

```
!(:; SDATE$; IJUNE$L2T)$$
```

if Speed finds one occurrence of DATE, it inserts JUNE, exits from the command loop, and carries out the L2T$ command. If Speed does not find DATE, it positions the CP at the beginning of the buffer.

Case Control Commands

With the Case Control commands, you can create and edit uppercase and lowercase files from an uppercase terminal. Your system must be equipped with an uppercase and lowercase line printer or terminal in order for you to use these commands. The general form of the command is:

*n*WC*xy*

n may be 0 to deactivate Case Control, a positive value for shifting up, or a negative value for shifting down.

x is any character you wish to use as the SHIFT character.

y is any character you wish to use as the SHIFT-lock character.

Use the command variations in Table 3.11 to create and edit text in uppercase and lowercase.

Command	Command Description
0WC	Deactivates case control. Speed reads characters from the terminal exactly as you type them, with no translation.
WC=$$	Returns the value of the case control mode: 0 = deactivated 1 = up SHIFT −1 = down SHIFT
*n*WC*x*	Translates all characters to lower case, except when they are preceded by SHIFT character *x*, which leaves the character in upper case (shifting up).
−*n*WC*x*	Leaves all characters as upper case unless you precede them with the SHIFT character *x*, which translates the character to lower case.
*n*WC*xy* or −*n*WC*xy*	Responds the same as *n*WC*x* and −*n*WC*x* except that *y* is a SHIFT-lock character which translates all characters following it to upper case (if shifting up) or lower case (if shifting down). The SHIFT-lock character remains in effect until you exchange it for another *y*, a case control deactivation (0WC), or a double Escape ($$).

Table 3.11 Case control commands

Case Control Examples

Look over the following examples 1 through 5 to familiarize yourself with the Speed Case Control functions.

Example 1

```
!1WC'$$

!I'HELLO$T$$

'HELLO
```

These commands instruct Speed to do the following:

- Set the control mode to shifting up. Use the apostrophe as the shift character.
- Insert HELLO and type out the line. On typeback, the same SHIFT character precedes any uppercase character, so output appears identical to input.

To insert the SHIFT character itself, precede it with another SHIFT character (which, in this example, is a single quotation mark). Speed treats the two adjacent single quote characters as one.

Similarly, if you want to insert a SHIFT-lock character, precede it immediately by a SHIFT character. Speed inserts the SHIFT-lock character itself into the command string.

Example 2

```
!-1WC#$$

!IS'PE'ED$$"

!0WC$$
```

These commands instruct Speed to:

- Set the case control mode to SHIFT-down. Use the single quotation mark as the SHIFT character.
- Insert SpEeD.
- Deactivate case control.

You can delete SHIFT and SHIFT-lock characters by pressing DEL after the shift or shift-lock character.

If you type a SHIFT character preceding a character other than a letter or another SHIFT character, shifting has no effect.

Case Control changes take effect after the command line containing the WC command. Therefore, as a general rule, you should terminate all Case Control commands with a double ESC ($$) to avoid possible confusion about when the command takes effect.

Example 3

```
!1WC#$$

!IABC#D#E
```

The commands instruct Speed to set the Case Control mode to shifting up with the number sign (#) as the SHIFT character.

Example 4

Once case control is active, it affects everything you type, including WC commands. Assume that you have activated case control with:

```
!-1WC#$$
```

and you want to change the mode to shifting up. Use one of these methods:

- If you change Case Control modes with the same SHIFT character, you must include a double SHIFT character (##) in order for Speed to interpret the command properly. For example:

  ```
  !1WC##$$
  ```

- If you deactivate Case Control before changing modes, the result is the same as in the previous example.

  ```
  !0WC$$

  !1WC#$$
  ```

- If you change modes and change the SHIFT character (from # to %), you will avoid confusion.

  ```
  !1WC%$$
  ```

Example 5

```
!1WC'&$$"

!I'THE &ECLIPSE&-LINE COMPUTERS$$

!T$$

'THE 'E'C'L'I'P'S'E-LINE COMPUTERS
```

These commands instruct Speed to

- Set Case Control mode to shift up, using an apostrophe as the SHIFT.character and an ampersand as the SHIFT-lock character.
- Insert:

  The ECLIPSE-line computers

Flow Control Commands

Using Flow Control commands, you can provide for:

1. Unconditional branching to predefined labels.
2. Command execution based on the evaluation (true or false) of certain conditions.

These features permit you to write Speed programs in order to solve complex editing problems.

Command String Labels

To name locations within a command string, use a command string label, which has the form:

!label!

A label is a string delimited by a pair of exclamation points. You can place it anywhere in a command string except in text string arguments.

Speed ignores labels unless they are specifically referenced by an unconditional branch command. Therefore, you can also use them as comments throughout a Speed program.

The Unconditional Branch Command

The Unconditional Branch command (O) provides unconditional branching. The general form of the O command is:

```
!Olabel$
```

When you issue this command, Speed transfers control to ! *label*! in the command string and continues processing the command string from the command immediately following ! *label*!. The label must contain fewer than 48 characters. The label may not be within a command loop. However, you can use an Unconditional Branch command to exit from a command loop.

Conditional Command Executions

The general form of a conditional command execution is:

nx command string

where:

n	is a numeric argument.
x	is one of the conditions (defined in Table 3.12) which Speed checks against the argument *n*.
command string	is a command string.

If the condition is true, Speed executes the commands before the next apostrophe ('), as well as the commands following the apostrophe. If the condition is false, Speed skips the commands before the next apostrophe and begins execution with the command immediately following the apostrophe. For example, in:

```
!V0"N3L'L2T$$
```

if V0 = 1, Speed executes the 3L command (moves the CP three lines forward) before executing L2T. If V0 = 0, Speed skips the command string and executes only L2T.

Mnemonic	Command Description
n"G*cmd-str*	True if *n* ⟩ 0
n"L*cmd-str*	True if *n* ⟨ 0
n"E*cmd-str*	True if *n* = 0
n"N*cmd-str*	True if *n* ≠ 0

Table 3.12 Conditional branch commands

You can nest conditional command execution commands in command loops, and you can nest command loops within conditional execution commands. When you include conditional execution commands in command loops, be certain to keep the apostrophe (') and its corresponding quotation mark ('') within the same command loop level. That is, when you nest command loops, you cannot issue a conditional execution command that starts at one command loop level and ends at another. For example:

```
!(SLDA$"NL2T)'$$
```

is unacceptable because the apostrophe was placed outside of the command loop.

Exit Commands

Use the commands in Table 3.13 to exit from Speed.

Mnemonic	Command Description
CTRL-C	Interrupts Speed, creates a BREAK.SV file, and returns you to the CLI. Use CTRL-C only as an emergency exit from Speed.
H	Exits from Speed and returns to the CLI. See Chapter 3.

Table 3.13 Exit commands

Special Commands and Command Modifiers

Speed is equipped with several special commands that do not fall under any previous category.

The *n* = command is a numeric Type-out command. It prints the decimal value of *n* on the terminal; *n* may be any numeric argument or expression.

The __*n* command, used as the first and only command after a prompt, places the previous command line in buffer *n*. This command is particularly useful for recovering from an inadvertent CTRL-A (abort) command. If you accidently type something other than __*n* as the first command after the prompt, you can recover by typing a CTRL-X or sufficient DELs to rub out the entire line as long as you have not typed $$.

The period (.) command represents the current CP position. This figure is equivalent to the number which represents all the characters between the beginning of the buffer and the character to the left of the CP. For example:

```
!.,ZT$$
```

types every character from the CP to the end of the buffer. If you type **.=$$**, Speed returns the number of characters from the beginning of the buffer to the CP.

The pound sign (#) command is a special double argument which is equivalent to 0,Z (the entire buffer). Use the pound sign preceding Text Type-out commands and Deletion commands.

The ? symbol is the Trace command. Use it to debug complex Speed macros. The first time you use the ? command, it turns tracing on. Speed prints the first character of each command (and its numeric arguments) as it is executed. The next time you use it, tracing is turned off.

The VL command represents the number of the line which contains the CP. If you type VL=$$, Speed types out the number of that line.

The VN command represents the number of lines in the current buffer. For example:

```
!VN-VLT$$
```

types every line from the current line to the end of the buffer.

If you type VN=$$, Speed types out the number of lines in the buffer.

The X*string* command executes *string* as a CLI command. It only works if CLI.SV and CLI.OL exist or are linked to the current directory. For example:

```
!XGTOD$$%
07/08/77 10:47:33%
```

(GTOD is the CLI command for the current date and time.)

The Z command represents the total number of characters in the buffer. If you type Z=$$, Speed prints the number of characters in the buffer.

We describe these commands in Table 3.14.

Mnemonic	**Command Description**
?	Debugs complex Speed macros. The first appearance of a ? command turns tracing on; Speed prints the first character of each command (and its numeric arguments) as it is executed. Each subsequent **?** command turns tracing off, if it is on; or turns tracing on, if it is off.
n=	Prints the decimal value of *n* on the terminal; *n* may be any numeric argument or expression.
__*n*	Places the previous command line in buffer *n* when used as the first (and only) command after a prompt.
. (period)	Represents the current CP position.
#	Represents a double argument which means the entire buffer.
VL	Represents the number of the line which contains the CP. If you type VL=$$, Speed types out the number of that line.
VN	Represents the number of lines in the current buffer.
X*string*	Executes string as a CLI command but only if CLI.SV and CLJ.OL exist or are linked to the current directory.
Z=	Represents the total number of characters in the buffer. If you type Z=$$, Speed prints the number of characters in the buffer.

Table 3.14 Special commands and special character numeric arguments

Command Modifiers

Two command modifiers alter the normal interpretation of commands. You can use the modifiers individually or in combinations.

The @ Modifier

Use this modifier with Insert (I) and search (S, C, N, Q) commands to change the command terminator/delimiter from an ESC ($) to any other character. You will find this modifier particularly useful for inserting or searching for a single ESC, or when using the CTRL-I *text* command.

The @ modifier defines the new command terminator as the first character following the command (I, S, C, N, or Q).

For examples of the @ modifier, see the insertion and search commands earlier in this chapter.

The Colon Modifier

The colon modifier has several functions:

1. Use it with any search (S, C, N, Q), or file input (A, Y) command or with an R or *n*R command to return a command value that you can then use as an argument to the next command. The command returns an operand of +1 if Speed executed the command successfully, and the operand 0 if the command was unsuccessful. No error message appears if the command was unsuccessful.

2. You can use the colon modifier with the semicolon (;) in iteration commands to reverse the action of the semicolon. That is, to transfer out of the command loop if the last search was successful.

   For examples, see the section on the Semicolon command in this chapter.

3. The colon modifies all variations of the P command to clear the current buffer. For example:

   ```
   !:5,500P$$
   ```

   tells Speed to output the 6th character through the 500th characters in the current buffer, append a Form Feed, and delete the buffer. This command is equivalent to:

   ```
   !5,500P$#K$$
   ```

4. The colon modifies the GW*filename* command.

   ```
   !:GWfilename$$
   ```

   is essentially the same as the GW *filename* command except that Speed creates and opens a random file rather than a sequential file for output.

5. The colon modifies the T command. For example:

   ```
   !:#T$$
   ```

   types the entire buffer on the line printer instead of the terminal.

Conclusion

Many of the commands we have described in this chapter are interchangeable: that is, you can use different command sequences to perform the same modifications. As you gain experience editing with Speed, you will probably discover your own methods for editing efficiently. That is the way Speed is designed to work. You can choose the best command for an easy solution to your editing problem.

If you feel comfortable using the commands in this chapter, you should be able to perform almost any editing task. Chapter 4 gives you an opportunity to test your Speed knowledge by practicing examples using combinations of Speed commands.

Chapter 4

Speed Examples

The examples in this chapter illustrate the use of many Speed commands. We assume that you will practice the commands using these examples as models. The examples do not necessarily represent the most efficient method of solving the stated problems.

Under the Command Description for each example, we explain each command and refer you to the section in Chapter 2 or Chapter 3 for a fuller description. We describe commands only when they first appear in the example. We show an empty line between command lines so that you can read them more easily. It is not necessary to insert an empty line between command lines at your console.

Remember to type a double ESC ($$) to end a command or a command string, and a single ESC ($) to separate commands in a command sequence. *R* is the RDOS/DOS Command Line Interpreter (CLI) prompt.

Example 1

1. Invoke Speed and create a new file.
2. Insert material from your terminal.
3. Instruct Speed to make the console display the file on your so that you can check its accuracy.
4. Close the file and exit from Speed.

The operation sequence is as follows:

```
R

SPEED FILE1 (CR)

CREATING NEW FILE

!Itext..............................$$

!#T$$

!UEH$$

R
```

Command	Description
SPEED FILE1	From the CLI, call Speed and execute a UY command on FILE1. Since FILE1 does not exist, Speed returns the message *CREATING NEW FILE*, then returns the Speed prompt. In this example, the UY command opens a file called FILE1 for input and a file called FILE1.SC for output. We describe other ways to open input and output files in the section Commands to Open and Close Files in Chapter 3.
I*text*...$$	Insert text into the current buffer of FILE1. We describe other kinds of insertion commands under Insertion Commands in Chapter 3.
#T$$	Type the contents of the entire buffer on the terminal. Other Type-out commands type one or more characters or lines from the buffer. See Type-Out Commands in Chapter 2.
UEH$$	Copy the current buffer to FILE1.SC; close FILE1 and FILE1.SC; delete FILE1; rename FILE1.SC to FILE1; and exit from Speed.

Example 2

1. Invoke Speed and open FILE1 for input.
2. Create an output file called OUTFILE.
3. Bring a page of the input into the current buffer and edit it.
4. Move the contents of the input file into the output file; clear the buffer.
5. Close both input and output files.

R

```
SPEED ⟨CR⟩

!GRFILE1$GWOUTFILE$Y$#T$$

!Stext$2K$J25T$$

!R#T$$

!3LCtext1$text 2$$

!E$GC$GR$$

!H$$
```

Command	**Description**
GRFILE1$ GWOUTFILE$	Open FILE1 for input. Create output file OUTFILE.
Y$#T$$	Yank the first page of FILE1 into the edit buffer, then type out the entire contents of the buffer. See Chapter 2 for a description of these commands.
S*text*$2K$J25T$$	Edit commands: S - Search command. See Search Commands in Chapter 2. K - Kill Line command. See Chapter 3. J - Character Pointer command. See Chapter 2.
R#T$$	The R command is a combination of the P and Y commands. It instructs Speed to: • Move the contents of the buffer to the output file. • Clear the buffer. • Move the next page from the input file into the buffer (equivalent to P$Y$$). See Output Commands in Chapter 3.
3LC*text1*$*text2*$$	Edit commands: L - Character Pointer Command. See Chapter 1. C - Change command. See Search and Change Commands in Chapter 2.
E$GC$GR$$	The E command is another combination input/output command. It instructs Speed to: • Move the contents of this buffer and the rest of the input file to the output file. • Clear the buffer. See Chapter 3. • Close the output file (GC). See Chapter 2. • Close the input file (GR). See Chapter 2.
H$$	Exit from Speed.

Examples 3 through 9 use more complex Speed commands.

Example 3

Change a file containing random-size pages to one which contains 50-line pages (method A).

```
!50WM$UYfilename$⟨⟨SCTRL-L$;-1D⟩ZJICTRL-L$R;⟩US$$
```

Command	**Description**
!50WM$	Set window mode for 50 lines long.
UY*filename*$$	Open *filename* for input and output. Yank 50 lines.
⟨⟨**SCTRL-L**$;-1D⟩	Search the buffer for all occurrences of CTRL-L (New Page or Form Feed) and delete them. Exit from the inner command loop when the search fails.
ZJI**CTRL-L**$	Position the CP at the end of the buffer and insert a New Page character.
R;⟩	Output the buffer to the output file. Yank another 50 lines and repeat outer loop. If the R command fails, exit the loop.
US$$	Close the I/O files with backup file.

NOTE: Although Z is a character-oriented argument and J is a line-oriented command, the ZJ combination is acceptable and provides a technique for positioning the CP at the end of the buffer. The combinations ZM and VN + 1J also position the CP at the end of the buffer.

Example 4

Change a file containing random-size pages to one which contains 50-line pages (method B).

```
!50WM$UYfilename$BS1$12I$BSO$$

⟨⟨SCTRL-B1$;-1D⟩ZJICTRL-B1$R;⟩US$$
```

Command	**Description**
!50WM$	Set window mode for 50 lines long.
UY*filename*$	Open filename for input and output. Yank 50 lines.
BS1$12I$BSO$	Put the ASCII decimal representation of Form Feed in edit buffer 1 and return to edit buffer 0.
⟨⟨**SCTRL-B**1$;-1D⟩	Search edit buffer 0 for all occurrences of the contents of edit buffer 1 (Form Feed character, CTRL-L) and delete them. Exit the inner command loop when the search fails.
ZJI**CTRL-B**1$	Position the CP at the end of the buffer and insert a Form Feed character from buffer 1.
R;⟩	Output the buffer to the output file. Yank another 50 lines and repeat the outer loop. If the R command fails, exit the loop.
US$$	Close the I/O files with backup files.

Example 5

Put consecutive page numbers at the top of each page in a file (method A).

```
!UYfilename$

0VS00VS10VS2

!PAGE!7⟨CTRL-I$⟩IPAGE□$

VI0-10"E0VS0VI1-10"E0VS1V12$' '

48+V2I48+V1I48+V0II

$:R"NOPAGE$'US$$
```

Command	Description
UY*filename*	Open *filename* for input and output. Yank the first page.
0VS00VS10VS2	Set the units', tens', and hundreds' counters to zero (numeric variables V0, V1, and V2, respectively). See Chapter 3 for numeric variables.
!PAGE!7⟨**CTRL-I**$ ⟩IPAGE□$	Set a *label* !PAGE!. Insert seven tabs and insert PAGE.
VI0-10"E	Increment the units' counter and subtract 10. If the result is not equal to 0, skip the commands before the matching apostrophe.
0VS0VI1 -10"E	If the units' counter reached 10, place a zero in the units' counter, increment the tens' counter and, subtract 10. If the result is not equal to zero, skip the commands before the matching apostrophe.
0VS1VI2$' '	If the tens' counter reached ten, place a zero in the tens' counter and increment the hundreds' counter.
48+V2I48+ V1I48+V0II	Insert the ASCII decimal equivalent of the integers stored in the hundreds', tens', and units' counters followed by a Carriage Return.
$:R"NOPAGE$ 'US$$	Output the page and yank another page. If the yank was successful, jump to location PAGE in the macro. Otherwise, close the I/O files with backup files. The colon returns an operand of plus one if the *R* command is successful and 0 if it fails. The N command tests this result.

Example 6

Put consecutive page numbers at the top of each page in a file (method B).

```
!UY filename$0VS0$⟨7⟨CTRL-I$⟩IPAGE□$VI0\I

$R;⟩US$$
```

Command	Description
!UY*filename*$	Open *filename* for input and output. Yank a page.
0VS0$	Set variable 0 to 0.
⟨7⟨**CTRL-I**$⟩ IPAGE□$	Space over seven tabs and insert PAGE.
VI0\I	Increment variable 0, convert the integer to its decimal ASCII representation, and insert a Carriage Return.
$R;⟩	Output the page to the output file and yank another page. If the yank is successful, repeat the loop. Otherwise, exit the loop.
US$$	Close the I/O files with backup files.

Example 7

Create a file (FILE.3) composed of rearranged pages from two other files (FILE.1 and FILE.2).

File.1	File.2	File.3
Page 1	Page 1	FILE.1
		Page 3
Page 2	Page 2	FILE.1
		Page 2
Page 3	Page 3	FILE.2
		Page 3
		FILE.1
		Page 1
		FILE.2
		Page 1
		FILE.2
		Page 2

```
!GWFILE.3$GRFILE.1$3⟨Y⟩P$

GRFILE.1$2⟨Y⟩P$GRFILE.2$3⟨Y⟩P$

GRFILE.1$YPGRFILE.2$YRP$GR$GC$$
```

Command	Description
!GWFILE.3$	Create a new output file FILE.3.
GRFILE.1$3⟨Y⟩P$	Open FILE.1 for input, get the third page, and output it to FILE.3.
GRFILE.1$2⟨Y⟩P$	Close and reopen FILE.1, get the second page, and output it to FILE.3.
GRFILE.2$3⟨Y⟩P$	Close FILE.1 and open FILE.2 for input. Get the third page and output it to FILE.3.
GRFILE.1$YP	Close FILE.2 and open FILE.1 for input. Yank the first page and output it to FILE.3.
GRFILE.2$	Close FILE.1 and open FILE.2 for input.
YRP$	Yank the first page and output it. Yank the second page and output it.
GR$GC$$	Close FILE.2 and FILE.3.

Example 8

Create a macro of Example 6 in a buffer which can be called for reuse by a CTRL-B*x* command (where *x* is the buffer containing the macro).

```
!BS1$@I%0VS0$⟨7⟨CTRL-I$⟩IPAGE□$VI0/I⟨CR⟩

$R;⟩US%BS0$$

!UYfilename$CTRL-B1$$
```

Command	Description
!BS1$@I%...%	Make buffer 1 current and insert the command string delimited by the % character.
BS0$$	Return to buffer 0.
!UY*filename*$**CTRL-B**1$$	Open *filename* for input and output and execute the macro in buffer 1. The I/O files are closed by the macro. This command line may be repeated for any filename without entirely retyping the macro contained in buffer 1.

Example 9

Create a macro of Example 6 in a file which can be called for reuse by a CTRL-G *filename*$ command. Saving the file lets you use it in subsequent editing sessions.

```
!GWMACRO.1$@I%0VS0$⟨7⟨CTRL-I$⟩IPAGE□$

VI0 \ I$R;⟩US%PWUE$$

!UYfilename$CTRL-GMACRO.1$$
```

Command	Description
!GWMACRO.1$	Open the file MACRO.1 for output.
@I%...%	Insert the command string delimited by the % character.
PWUE$$	Write the contents of the buffer to the output file and close the output file. Use PW to avoid the Form Feed, which would be interpreted as an erroneous command in the file.
!UY*filename*$**CTRL-G** MACRO.1$$	Open *filename* for input and output and execute the commands in file MACRO.1.

Chapter 5

Command Summary

This chapter contains an alphabetical list of Speed commands and command modifiers with a brief description of what each does. The list includes arithmetic operators and control characters to which Speed responds.

The list starts with alphabetized special characters (graphic characters that are neither letter, digit, nor space characters), special character numeric arguments, and conditional branch commands. Alphabetized letter commands follow, with the simplest form of the command appearing first, then command formats using special character modifiers and/or numeric arguments.

Command and Symbol Descriptions

*	(asterisk) Multiplies (as arithmetic operator).
@	(at sign) Modifies Insert and Search commands, changing the text delimiter to any character appearing immediately after the command.
:	(colon) Modifies search, file input, and R commands (A, Y, R, S, N, Q, C), returning an operand to the next command (1 if the search or input succeeds and 0 if it fails). Modifies type commands (*n*T, -*n*T, *m*,*n*T), sending output to the line printer. Modifies output commands (P, *n*P, *m*, *n*P, PW, *n*PW, *m*, *n*PW), clearing the buffer after output. Modifies the GW*filename* command, creating a randomly organized output file.
:;	(colon with semicolon) Jumps out of the command loop if the last Search command was successful.
!*string*!	(exclamation point delimiting a string) Delimits a label named string within a command string. Must be referenced by *Olabel*. Treats !*string*! as a comment when used without an Unconditional Branch command.
–	(minus or negative, depending upon use) Subtracts (as arithmetic operator); counts back (as part of numeric argument to a command).
–*n*	(negative numeric argument) Places the previous command string in buffer *n*. Must enter it as the first command after the prompt appears.
n\	(numeric argument with backslash) Inserts the ASCII representation of the decimal number *n* into the buffer at the CP location.
n"E*command string*'	(numeric argument, two apostrophes, E, command string, and apostrophe) Executes the command string if *n* equals 0.
n"G*command string*'	(numeric argument, two apostrophes, G, command string, and apostrophe) Executes the command string if *n* is greater than 0.
n"L*command string*'	(numeric argument, two apostrophes, L, command string, and apostrophe) Executes the command string if *n* is less than 0.
n"N*command string*'	(numeric argument, two apostrophes, N, command string, and apostrophe) Executes the command string if *n* equals 0.
–*n*⟨*command string*⟩	(negative numeric argument with command string in brackets) Skips command within the brackets.
n⟨*command string*⟩	(numeric argument with command string in angle brackets) Performs enclosed commands *n* times.
n \	(numeric argument with backslash) Inserts the ASCII representation of the decimal *n* into the buffer at the CP location, suppressing leading zeros.
n=	(numeric argument with equal sign) Types out the value of numeric argument *n*.
n;	(numeric argument with semicolon) Jumps out of the command loop if *n* is less than 0.

_n	(underscore with numeric argument) Places the previous command string in buffer *n*. Must be the first command after the prompt appears.
.	(period) Represents the current CP position.
#	(pound sign) Applies command to entire edit buffer. (Equivalent to the double arguments O,Z.)
+	(plus) Adds (as arithmetic operator).
;	(semicolon) Jumps out of the command loop if the last Search command failed.
/	(slash) Divides (as arithmetic operator).
?	(question mark) Turns trace mode on (if off) and off (if on).
A	Reads one page from the input file and appends it to the current edit buffer.
B?	Lists all active buffers and their lengths (in characters). Uses a right angle bracket ()) to indicate which edit buffer is current.
B?*x*	Types out the status of buffer *x*.
BA*x*	Activates buffer *x*.
BC*x*	Clears buffer *x*, writes the entire contents of the current buffer to buffer *x*, and positions the CP at the beginning of buffer *x*.
*m,n*BC*x*	Clears buffer *x*, writes the (*m* + 1) through *n*th characters from the current buffer to buffer *x*, and positions the CP at the beginning of buffer *x*.
−*n*BC*x*	Clears buffer *x*, writes the preceding *n* lines plus the characters up to the CP in the current line to buffer *x*, and positions the CP at the beginning of buffer *x*.
*n*BC*x*	Clears buffer *x*, writes the next n lines from the CP position in the current buffer to buffer *x*, and positions the CP at the beginning of buffer *x*.
BGC*filename*	Closes the input file local to the current edit buffer.
BGR*filename*	Closes the input file for the current edit buffer and opens *filename* for input.
BGW*filename*	Creates and opens a sequentially organized local output file for the current buffer.
BK*x*	Deletes buffer *x*.
BS*x*	Changes the current buffer to buffer *x*.
BT*x*	Clears buffer *x*, writes the entire contents of the current buffer to buffer *x*, and deletes the characters moved to buffer *x* from the current buffer.
*m,n*BT*x*	Clears buffer *x*, writes the (*m* + 1) through *n*th characters from the current buffer to buffer *x*. Deletes the characters moved to buffer *x* from the current buffer.
−*n*BT*x*	Clears buffer *x*, writes the preceding lines plus the characters up to the CP in the current line to buffer *x*. Deletes the characters moved to buffer *x* from the current buffer.
*n*BT*x*	Clears buffer *x*, writes the next *n* lines from the CP position in the current buffer to buffer *x*. Deletes the characters moved to buffer *x* from the current buffer.
BU*x*	Activates buffer *x*.
BUE	Transfers remainder of local input file to the output file and closes the input and output files. Deletes the input file and renames the output file to the input filename when file has been opened with a BUY command.
BUS	Transfers remainder of local input file to the output file. Closes the input/output files. Creates backup file when file has been opened with a BUY command.
BUY*filename*	Opens local *filename* for input, opens *filename.SC* for output, and yanks in a page.
C*text1*$*text2*$	Searches for *text1* and replaces it with *text2*.
@C%*text1*%*text2*%	Searches for *text1* delimited by the first character after C (arbitrarily shown here as %) and replaces it with *text2*.
@*m,n*C%*text1*%*text2*%	Searches (*m* + 1) through *n*th characters for *text1*, delimited by the first character after C (arbitrarily shown here as %), and replaces it with *text2*.
@ − *n*C%*text1*%*text2*%	Searches the preceding *n* lines plus the characters up to the CP on the current line for *text1* delimited by the first character after C (arbitrarily shown here as %), and replaces it with *text2*.
@*n*C%*text1*%*text2*%	Searches the next *n* lines for *text1*, delimited by the first character after C (arbitrarily shown here as %) and replaces it with *text2*.
@0C%*text1*%*text2*%	Searches the current line from the beginning of the line up to the CP for *text1*, delimited by the first character after C (arbitrarily shown here as %) and replaces it with *text2*.

:C*text1*$*text2*	Searches for *text1*, replaces it with *text2*, and returns an operand to the next command (1 if the search succeeds and 0 if it fails). No error message is displayed after an unsuccessful search.
:*m*,*n*C*text1*$*text2*	Searches (*m* + 1) through *n*th characters for *text1*, replaces it with *text2*, and returns an operand to the next command. No error message is displayed after an unsuccessful search.
: − *n*C*text1*$*text2*	Searches the next *n* lines for *text1*, replaces it with *text2*, and returns an operand to the next command. No error message is displayed after an unsuccessful search.
:*n*C*text1*$*text2*	Searches the next *n* lines for *text1*, replaces it with *text2*, and returns an operand to the next command. No error message is displayed after an unsuccessful search.
:0C*text1*$*text2*$	Searches the current line from the beginning of the line up to the CP for *text1*, replaces it with *text2*, and returns an operand to the next command. No error message is displayed after an unsuccessful search.
@:C%*text1*%*text2*%	Searches for *text1* delimited by first character after C, replaces it with *text2*, and returns an operand to the next command. No error message is displayed after an unsuccessful search.
@:*m*,*n*C*text1*$*text2*$	Searches (*m* + 1) through *n*th characters for *text1* delimited by first character after C, replaces it with *text2*, and returns an operand to the next command. No error message is displayed after an unsuccessful search.
@: − *n*C%*text1*%*text2*%	Searches the preceding *n* lines plus the characters up to the CP on the current line for *text1* delimited by first character after C, replaces it with *text2*, and returns an operand to the next command. No error message is displayed after an unsuccessful search.
@:*n*C%*text1*%*text2*%	Searches from the CP through the next *n* Carriage Returns for *text1* delimited by first character after C, replaces it with *text2*, and returns operand to the next command. No error message is displayed after an unsuccessful search.
@:0C%*text1*%*text2*%	Searches the current line from the beginning of the line up to the CP for *text1* delimited by first character after C, replaces it with *text2*, and returns an operand to the next command. No error message is displayed after an unsuccessful search.
m,*n*C*text1*$*text2*$	Searches (*m* + 1) through *n*th characters for *text1* and replaces it with *text2*.
− *n*C*text1*$*text2*$	Searches the preceding *n* lines plus the characters up to the CP on the current line for *text1* and replaces it with *text2*.
*n*C*text1*$*text2*$	Searches from the CP through the next *n* Carriage Returns for *text1* and replaces it with *text2*.
0C*text1*$*text2*$	Searches the current line from the beginning of the line up to the CP for *text1* and replaces it with *text2*.
CTRL-B*x*	Inserts the contents of buffer *x* (text, commands, or both) into the command string in the position of the CTRL-B*x* command. Evaluates the contents of buffer *x* as part of the command string. Buffer *x* may not be the current buffer.
CTRL-C	Returns to CLI from Speed.
CTRL-E *any control* character	Interprets the next character literally, rather than as a special character.
CTRL-G*filename*	Inserts the contents of *filename* (text, commands, or both) into the command string in the position of the CTRL-G*filename* command. Evaluates the contents of *filename* as part of the command string.
CTRL-I	Inserts a Tab.
CTRL-I*text*$	Inserts a Tab plus text delimited by CTRL-I and ESC ($).
CTRL-N*x*	Accepts any character except *x* in the position occupied CTRL-N.
CTRL-P	Positions the CP at the position occupied by CTRL-P in the Search string if the search is successful.
CTRL-T	Accepts zero or more space or Tab characters in this position.
CTRL-V*x*	Accepts any number of the character *x* in the position occupied by CTRL-V.
CTRL-Z	Accepts any character in the position occupied by CTRL-Z.
− *n*D	Deletes *n* characters to the left, starting at the CP position.
*n*D	Deletes *n* characters to the right, starting at the CP position.
E	Outputs the buffer and the rest of the input file.
GC	Closes the current global output file.
GR	Cioses the current global input file.

GR*filename*	Closes the current input file and opens *filename* for input.
GW*filename*	Creates and opens an output file in sequentially organized format.
:GW*filename*	Creates and opens an output file in randomly organized format.
H	Returns to the CLI from Speed.
I*text*$	Inserts text delimited by I and ESC ($) at the current position of the CP.
ICTRL-E *any control* character	Inserts both characters into the line at the current position of the CP.
@I%*text*%	Inserts text delimited by the first character after I (in this case, %) at the current position of the CP.
@CTRL-I*text*CTRL-I	Inserts text delimited by CTRL-I at the current position of the CP.
CTRL-I*text*$	Inserts a Tab into the edit buffer preceding the text string.
n \	Inserts the ASCII representation of the decimal number *n* into the buffer at the CP location.
*n*I	Inserts the character whose ASCII decimal equivalent (from Appendix A) is *n* at the current location of the CP.
J	Moves the CP to the beginning of the buffer.
*n*J	Moves the CP to the beginning of the *n*th line in the buffer.
ZJ	Moves the CP to follow the last character in the buffer.
0J	Moves the CP to the beginning of the buffer.
K	Deletes the characters from the beginning of the current line to the CP position.
−*n*K	Deletes the preceding *n* lines and the characters up to the CP on the current line.
m,*n*K	Deletes the (*m*+1) through *n*th characters.
*n*K	Deletes the lines starting at the CP position through *n* Carriage Returns.
#K	Deletes all lines in the current buffer.
0K	Deletes the characters from the beginning of the current line to the CP position.
L	Moves the CP to the beginning of the current line.
−*n*L	Moves the CP to the beginning of the *n*th line preceding the current line.
*n*L	Moves the CP to the beginning of the line following the *n*th Carriage Return.
0L	Moves the CP to the beginning of the current line.
−*n*M	Moves the CP *n* characters to the left.
*n*M	Moves the CP *n* characters to the right.
N	
N*text*$	Searches the rest of the input file for text. For each page, outputs the buffer to the output file.
@N%*text*%	Searches the rest of the input file for text delimited by first character after N.
@:N%*text*%	Searches the rest of the input file for text delimited by the first character after N and returns an operand to the next command (1 if search succeeds and 0 if it fails). No error message is displayed after an unsuccessful search.
:N*text*	Searches the rest of the input file for text and returns an operand to the next command (1 if search succeeds and 0 if it fails). No error message is displayed after an unsuccessful search.
O*label*	Transfers control to label string (*!label!*) and continues processing command string from command immediately following *!label!*.
P	Outputs the edit buffer (with an appended New Page character (CTRL-L)) to the output file.
:P	Outputs the edit buffer (with an appended New Page character (CTRL-L)) to the output file and clears the buffer after output.
m,*n*P	Outputs the (*m*+1) through *n*th characters (with an appended New Page character (CTRL-L)) to the output file.
−*n*P	Outputs the preceding *n* lines plus the characters up to the CP (with an appended New Page character (CTRL-L)) to the output file.
*n*P	Outputs *n* lines from the CP (with an appended New Page character (CTRL-L)) to the output file.
0P	Outputs the characters on the current line from the beginning of the line to the location of the CP (with an appended New Page character (CTRL-L)) to the output file.

PW	Outputs the edit buffer to the output file.
m,*n*PW	Outputs the (*m*+1) through *n*th characters inclusive to the output file.
−*n*PW	Outputs the preceding *n* lines plus the characters up to the CP on the current line to the output file.
*n*PW	Outputs *n* lines from the CP to the output file.
0PW	Outputs the current line from the beginning of the line to the CP to the output file.
Q*text*	Searches file for text. Does not output text to the buffer.
@Q%*text*%	Starts at CP and searches for text delimited by first character after Q (arbitrarily shown here as %) without outputting text to the buffer.
@:Q%*text*%	Searches file for text delimited by first character after Q (arbitrarily shown here as %) without outputting text to the buffer. Returns an operand to the next command (1 if the search succeeds and 0 if it does not). No error message is displayed after an unsuccessful search.
:Q*text*	Searches file for text without outputting text to the buffer, and returns an operand to the next command (1 if search succeeds and 0 if it does not). No error message is displayed after an unsuccessful search.
R	Outputs the contents of the edit buffer, clears the buffer, and yanks the next page from the input file.
:R	Outputs the contents of the edit buffer, clears the buffer, yanks in the next page, and returns an operand to the next command.
:*n*R	Outputs the contents of the edit buffer, clears the buffer, yanks in the next page from the input file *n* times, and returns an operand to the next command.
*n*R	Outputs the contents of the edit buffer, clears the buffer, and yanks in the next page from the input file *n* times.
S*text*$	Starting at the CP, searches for text delimited by S and ESC ($) in the current buffer, and positions CP after text.
@S%*text*%	Starting at the CP, searches for text delimited by the first character after S (arbitrarily shown here as %), and positions CP after text.
@*m*,*n*S%*text*%	Searches the (*m*+1) through *n*th characters for text delimited by the first character after S (arbitrarily shown here as %), and positions CP after text.
@−*n*S%*text*%	Searches the preceding *n* lines plus the characters up to the CP on the current line for text delimited by the first character after S (arbitrarily shown here as %), and positions CP after text.
@*n*S%*text*%	Searches from the CP through the next *n* Carriage Returns for text delimited by the first character after S (arbitrarily shown here as %), and positions CP after text.
@0S%*text*%	Searches the current line from the beginning of the line up to the CP for text delimited by the first character after S (arbitrarily shown here as %), and positions the CP after text.
@:S%*text*%	Starting at the CP, searches for text delimited by the first character after S (arbitrarily shown here as %), returns an operand to the next command, and positions the CP after text. No error message is displayed with an unsuccessful search.
@:*m*,*n*S%*text*%	Searches the (*m*+1) through *n*th characters for text delimited by the first character after S (arbitrarily shown here as %), returns an operand to the next command, and positions the CP after text. No error message is displayed with an unsuccessful search.
@:−*n*S%*text*%	Searches the preceding *n* lines plus the characters up to the CP on the current line for text delimited by the first character after S (arbitrarily shown here as %), returns an operand to the next command, and positions the CP after text. No error message is displayed with an unsuccessful search.
@:*n*S%*text*%	Searches from the CP through the next *n* Carriage Returns for text delimited by the first character after S (arbitrarily shown here as %), returns an operand to the next command, and positions the CP after text. No error message is displayed with an unsuccessful search.
@:0S%*text*%	Searches the current line from the beginning of the line up to the CP for text delimited by the first character after S (arbitrarily shown here as %), returns an operand to the next command, and positions the CP after text. No error message is displayed with an unsuccessful search.
:S*text*$	Searches for text and returns an operand to the next command (1 if the search succeeds and 0 if it does not), and positions the CP after text. No error message is displayed with an unsuccessful search.

:*m*,*n*S*text*$	Searches the (*m* + 1) through *n*th characters for text, returns an operand to the next command, and positions the CP after text. No error message is displayed with an unsuccessful search.
: − *n*S*text*$	Searches the preceding *n* lines plus the characters up to the CP on the current line for text, returns an operand to the next command, and positions the CP after text. No error message is displayed with an unsuccessful search.
:*n*S*text*$	Searches from the CP through the next *n* carriage returns for text, returns an operand to the next command, and positions the CP after text. No error message is displayed with an unsuccessful search.
:0S*text*$	Searches the current line from the beginning of the line up to the CP for text, returns an operand to the next command, and positions the CP after text. No error message is displayed with an unsuccessful search.
m,*n*S*text*$	Searches the (*m* + 1) through *n*th characters for text and positions the CP after text.
− *n*S*text*$	Searches the preceding *n* lines plus the characters up to the CP on the current line for text and positions the CP after text.
*n*S*text*$	Searches from the CP through the next *n* Carriage Returns for text and positions the CP after text.
0S*text*$	Searches the current line from the beginning of the line up to the CP for text and positions the CP after text.
T	Types the current line showing the position of the CP.
:T	Types text on the line printer instead of the terminal.
m,*n*T	Types the (*m* + 1) through *n*th characters inclusive.
− *n*T	Types the *n* lines preceding the current line plus the characters on the current line up to the CP.
*n*T	Types from the CP through the next *n* carriage returns.
0T	Types from the beginning of the current line up to the position of the CP.
#T	Types out the contents of the buffer.
:*m*,*n*T	Outputs the (*m* + 1) through *n*th characters to the line printer instead of displaying them at the console.
: − *n*T	Outputs *n* lines preceding the current line plus the characters on the current line up to the CP to the line printer instead of displaying them at the console.
:*n*T	Outputs characters from the CP through the next *n* Carriage Returns to the line printer instead of displaying them at the console.
:0T	Outputs the current line, showing the position of the CP, to the line printer instead of displaying it at the console.
U?	Types global and local file status.
UE	Transfers the remainder of the input file to the output file. Closes the input/output files. Deletes the input file and renames the output to the input *filename* when the file has been opened by a UY command.
US	Transfers the remainder of the input file to the output file. Closes the input/output files. Creates a backup file when the file has been opened with a UY command.
UY*filename*	Opens *filename* for input, *filename.SC* for output, and yanks a page.
V*v*	Represents the current value of variable *v*.
*n*VS*v*	Sets the variable *v* to value *n* and returns that value.
VD*v*	Decrements the value of variable *v* and represents the decremented value.
VI*v*	Increments the value of variable *v* and represents the incremented value.
VL	Represents the number of the line which contains the CP.
VN	Represents the number of lines in the current buffer.
WC	Returns the value representing the case control mode (0, 1, or -1).
0WC	Deactivates case control.
− *n*WC*x*	Activates shift-down using *x* as the shift character.
*n*WC*x*	Activates shift-up using *x* as the shift character.
− *n*WC*xy*	Activates shift-down using *x* as the shift character and *y* as the shift-lock character.
*n*WC*xy*	Activates shift-up using *x* as the shift character and *y* as the shift-lock character.

WM=	Returns the value of the data input mode (0 or *n*).
*n*WM	Changes from page to window mode with an *n* line window length.
0WM .	Changes from window to page mode.
X*string*	Executes string as a CLI command.
Y	Clears the buffer and reads one page from the input file.

Chapter 6

Implementation Notes

Memory Utilization

Initially, Speed allocates 10 Kwords of memory, allowing room for approximately 14,000 characters in ECLIPSE®-line computers, and 13,000 in the NOVA®-line or microNOVA® computers. This space is divided, as needed, between the program, the current edit buffer, and the command line input buffer. If Speed needs more room, it obtains 1 Kword more memory, and the message

CORE*

is displayed on your terminal. Each additional 1 Kword of memory allows storage space for 2,048 characters.

If Speed exhausts all available memory during command execution, your console displays this error message:

ERROR:MEMORY SPACE EXHAUSTED

and the current command is aborted. Enough room is left at this point to execute some commands, but you should issue a text output command shortly.

If you are typing a command line and you exhaust all available space, your console displays this error message:

ERROR:MEMORY SPACE EXHAUSTED

and issues a new prompt. You can then save the old command line by issuing the __*n* command (see Chapter 3, Table 3.14).

Error Handling

There are two types of Speed errors: editing errors and RDOS/DOS errors. We describe Speed errors in Appendix B. The system stores a text file, Speed.ER, which contains the text of all these error messages. Errors appear in the form:

ERROR:MESSAGE

MESSAGE is the error text shown in Appendix B.

If the Speed.ER file is not present in the directory where you are running Speed, your console displays this message:

WARNING: EXPANDED ERROR TEXT NOT AVAILABLE

before the first prompt. Editor errors are of the form:

ERROR:Snn

S*nn* is the numeric error code described in Appendix B.

RDOS/DOS errors are described in the appropriate system reference manual. If an RDOS/DOS error occurs, Speed types:

ERROR:Rnn

nn is the number of the error code.

After any error, Speed types up to nine characters of the command line. The first character is the erroneous command. The only exception occurs when Speed encounters an error while executing a buffer or insert file (CTRL-B*x* or CTRL-G*filename* command). If such an error occurs, Speed displays the correct command string insertion command. For example, if you are executing a macro in buffer A and an error occurs, Speed types:

↑BA

instead of flagging the actual error in buffer A.

The only other error that can occur is a fatal internal error. If this error occurs, Speed immediately returns control to the CLI. The CLI then returns the error message:

UNKNOWN ERROR CODE = xxxxxx

xxxxxx is the address of the error. Such an error can only be caused by a malfunction in RDOS or Speed, so it is unlikely that you will see this error message. However, if you do, please save all details on how to recreate the error and notify your local Data General Applications Engineer.

RDOS/DOS Channel Usage

For most editing functions, eight channels are more than you will ever need. However, if you are doing complex editing, you may require more. (With 36 buffers, Speed has the potential to use more than the limit of 64 channels, since it needs one channel for each open input, output, command, and insertion file.)

When Speed runs out of channels, your console displays the error code:

ERROR:NO MORE CHANNELS AVAILABLE (error code 21)

You can change the number of channels by modifying location 413 in the save file. Use the OEDIT command (the octal editor). The right byte at location 413 contains the number of channels initialized for Speed.

Changing the Escape Character

The Speed Escape character is the same as the ASCII ESC character (octal 33). If you have an older model keyboard,it may generate another code from the ESC (or ALT MODE) key. In order to use Speed from such a terminal, you must use OEDIT and change location 45 in the save file so that it contains the correct octal code for the ESC character as generated by your terminal.

Appendix A

ASCII Character Set

DECIMAL	OCTAL	HEX	KEY SYMBOL	MNEMONIC
0	000	00	↑@	NUL
1	001	01	↑A	SOH
2	002	02	↑B	STX
3	003	03	↑C	ETX
4	004	04	↑D	EOT
5	005	05	↑E	ENQ
6	006	06	↑F	ACK
7	007	07	↑G	BEL
8	010	08	↑H	BS (BACKSPACE)
9	011	09	↑I	TAB
10	012	0A	↑J	NEW LINE
11	013	0B	↑K	VT (VERT. TAB)
12	014	0C	↑L	FORM FEED
13	015	0D	↑M	CARRIAGE RETURN
14	016	0E	↑N	SO
15	017	0F	↑O	SI
16	020	10	↑P	DLE
17	021	11	↑Q	DC1
18	022	12	↑R	DC2
19	023	13	↑S	DC3
20	024	14	↑T	DC4
21	025	15	↑U	NAK
22	026	16	↑V	SYN
23	027	17	↑W	ETB
24	030	18	↑X	CAN
25	031	19	↑Y	EM
26	032	1A	↑Z	SUB
27	033	1B	ESC	ESCAPE
28	034	1C	↑\	FS
29	035	1D	↑]	GS
30	036	1E	↑↑	RS
31	037	1F	↑—	US

DECIMAL	OCTAL	HEX	KEY SYMBOL
32	040	20	SPACE
33	041	21	!
34	042	22	" (QUOTE)
35	043	23	#
36	044	24	$
37	045	25	%
38	046	26	&
39	047	27	' (APOS)
40	050	28	(
41	051	29	)
42	052	2A	*
43	053	2B	+
44	054	2C	, (COMMA)
45	055	2D	-
46	056	2E	. (PERIOD)
47	057	2F	/
48	060	30	0
49	061	31	1
50	062	32	2
51	063	33	3
52	064	34	4
53	065	35	5
54	066	36	6
55	067	37	7
56	070	38	8
57	071	39	9
58	072	3A	:
59	073	3B	;
60	074	3C	<
61	075	3D	=
62	076	3E	>
63	077	3F	?
64	100	40	@

DECIMAL	OCTAL	HEX	KEY SYMBOL
65	101	41	A
66	102	42	B
67	103	43	C
68	104	44	D
69	105	45	E
70	106	46	F
71	107	47	G
72	110	48	H
73	111	49	I
74	112	4A	J
75	113	4B	K
76	114	4C	L
77	115	4D	M
78	116	4E	N
79	117	4F	O
80	120	50	P
81	121	51	Q
82	122	52	R
83	123	53	S
84	124	54	T
85	125	55	U
86	126	56	V
87	127	57	W
88	130	58	X
89	131	59	Y
90	132	5A	Z
91	133	5B	[
92	134	5C	\
93	135	5D	]
94	136	5E	↑ OR ∧
95	137	5F	← OR —
96	140	60	` (GRAVE)

DECIMAL	OCTAL	HEX	KEY SYMBOL
97	141	61	a
98	142	62	b
99	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6A	j
107	153	6B	k
108	154	6C	l
109	155	6D	m
110	156	6E	n
111	157	6F	o
112	160	70	p
113	161	71	q
114	162	72	r
115	163	73	s
116	164	74	t
117	165	75	u
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7A	z
123	173	7B	{
124	174	7C	|
125	175	7D	}
126	176	7E	~ (TILDE)
127	177	7F	DEL (RUBOUT)

Appendix B
SPEED Error Messages

01 ILLEGAL FILE NAME
Filename is incorrect.

02 SYNTAX ERROR
Format of command is incorrect (e.g., 2#T).

03 ILLEGAL VARIABLE NAME
Legal variable names are zero through nine only.

04 ILLEGAL NUMBER OF ARGUMENTS TO COMMAND
Commands such as *m,n*R are not legal.

05 ILLEGAL BUFFER NAME
Zero through nine and A through Z are the only legal buffer names.

06 BUFFER IS INACTIVE
You attempted to use the BK*x* or CTRL-B*x* command on an inactive buffer.

07 MAXIMUM ITERATION LEVEL EXCEEDED
Command loop nesting level is greater than 10.

10 NO OPEN FILE
You attempted to use the A, Y, P, PW, R, E, UE, or US command without an open file.

11 FILE ALREADY EXISTS
You attempted to create an existing file using the GW command.

12 FILE DOES NOT EXIST
You attempted to open a nonexistent file.

13 FILE ALREADY OPEN
You attempted to open an output file with a GW or UY command without first closing a previously opened output file.

14 NO MORE CHARACTERS IN INPUT FILE
An A, Y, or R command has reached an end of file.

15 UNSUCCESSFUL SEARCH
The string that you are searching for was not found.

16 MAXIMUM INSERT DEPTH EXCEEDED
The nesting level of CTRL-B*x* and CTRL-G*filename*$ has been exceeded.

17 SEARCH STRING OR ⟨ ⟩ BROKEN OVER TWO LEVELS
The search command or command loop starts at one command insert level and ends at another level.

20 INSERT FILE TOO LONG
The file is larger than 64 Kbytes.

21 NO MORE CHANNELS AVAILABLE
All I/O channels in use. See Chapter 3.

22 INPUT LINE TOO LONG
The maximum line length is 132 characters. A line longer than 132 characters has been read using an A, Y, R, E, or UY command. This is a warning message only; execution of the command string continues.

23 ATTEMPT TO DELETE CURRENT BUFFER
A BK*x* command may not be used for the current buffer.

24 PARITY ERROR
The character in error is replaced by a backslash (\ ). This is a warning message only; execution of the command string continues.

25 STACK OVERFLOW
Internal Speed error.

26 MEMORY SPACE EXHAUSTED
No more memory is available. See Chapter 6.

27 ATTEMPT TO EXECUTE CURRENT BUFFER
A CTRL-B*x* command may not be used for the current buffer.

30 UNTERMINATED STRING
The @S or @I command needs a second delimiter.

31 ⟨ WITH NO CORRESPONDING ⟩

32 " WITH NO CORRESPONDING '

33 LABEL NOT FOUND
The label referenced by an unconditional jump (0) command cannot be found.

34 UNABLE TO OPEN $LPT
No line printer in system.

35 STRING ARGUMENT TOO LONG
In O*string*$ command, the string is greater than 48 characters.

36 *FIRST ARGUMENT GREATER THAN SECOND ARGUMENT*
In commands which have two arguments, the first must be less than or equal to the second (for example, *m*,*n*T).

40 *RENAMING ERROR*
Improper file handling detected during UE/US execution.

41 *ILLEGAL COMMAND*
A character or characters were used which are not defined as a legal Speed command.

42 *ILLEGAL ARGUMENT TO COMMAND*
A command which accepts only a positive argument was given a negative argument.

43 *ILLEGAL CONTROL CHARACTER IN SEARCH STRING*
A character other than those described in Table 2.2 in a search command string.

44 *FILE READ PROTECTED*
You attempted to open a read-protected input file.

45 *FILE WRITE PROTECTED*
You attempted to write or append to a write-protected output file.

46 *FILE OPENED WITH UY*
You attempted to use a GR or GC command to close a file opened with a UY command. Use US or UE instead.

47 *FILENAME TOO LONG*
You specified a filename containing more than 49 characters. RDOS checks only the first 10 characters in a filename for uniqueness and up to 2 characters in the filename extension.

53 *DIRECTORY SPECIFIER UNKNOWN*
You attempted to access a nonexistent or unlinked directory.

Index

Data General Corporation, Westboro, MA 01580

069-400017-01