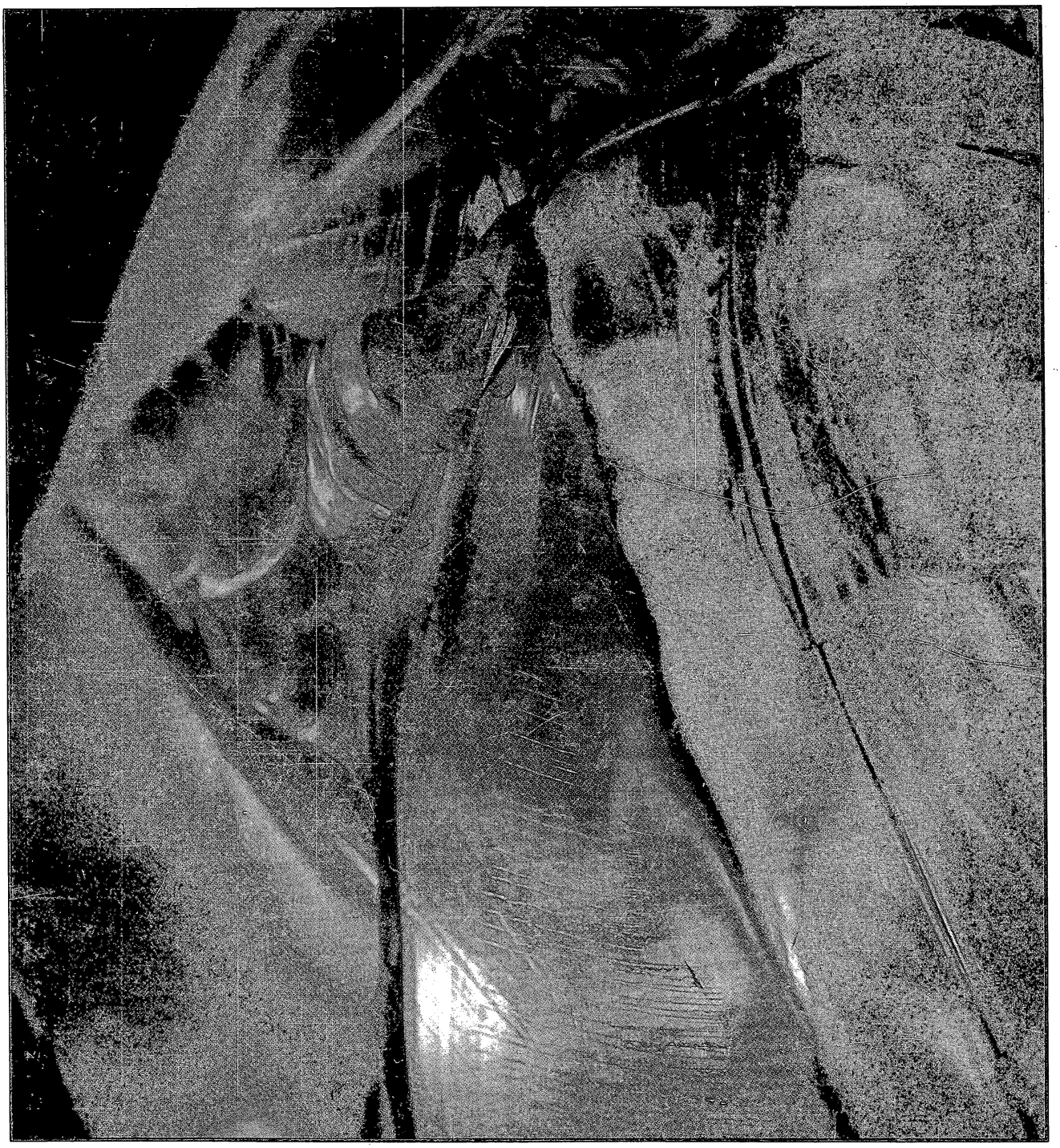
MP/AOS

Speed Text Editor



Data General

MP/AOS

Speed Text Editor

Data General

Notice

Data General Corporation (DGC) has prepared this document for use by DGC personnel, customers, and prospective customers. The information contained herein shall not be reproduced in whole or in part without DGC's prior written approval.

DGC reserves the right to make changes in specifications and other information contained in this document without prior notice, and the reader should in all cases consult DGC to determine whether any such changes have been made.

THE TERMS AND CONDITIONS GOVERNING THE SALE OF DGC HARDWARE PRODUCTS AND THE LICENSING OF DGC SOFTWARE CONSIST SOLELY OF THOSE SET FORTH IN THE WRITTEN CONTRACTS BETWEEN DGC AND ITS CUSTOMERS. NO REPRESENTATION OR OTHER AFFIRMATION OF FACT CONTAINED IN THIS DOCUMENT INCLUDING BUT NOT LIMITED TO STATEMENTS REGARDING CAPACITY, RESPONSE-TIME PERFORMANCE, SUITABILITY FOR USE OR PERFORMANCE OF PRODUCTS DESCRIBED HEREIN SHALL BE DEEMED TO BE A WARRANTY BY DGC FOR ANY PURPOSE, OR GIVE RISE TO ANY LIABILITY OF DGC WHATSOEVER.

IN NO EVENT SHALL DGC BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS DOCUMENT OR THE INFORMATION CONTAINED IN IT, EVEN IF DGC HAS BEEN ADVISED, KNEW OR SHOULD HAVE KNOWN OF THE POSSIBILITY OF SUCH DAMAGES.

Ordering No. 069-400202

Rev. 00, August 1982

Preface

This manual has been written to serve experienced programmers. We assume that the reader is familiar with the MP/AOS Operating System and its Command Line Interpreter (CLI). Those who are not should read *MP/AOS Command Line Interpreter (CLI)* (DGC No. 069-400201).

Organization

The manual comprises four chapters, two appendices, and an index.

Chapter 1, "Speed Concepts and Syntax," covers units of text; buffers and text files; and the Speed command format.

Chapter 2, "Using Speed," discusses opening and closing files, reading text into the buffer, editing, writing text to the output file, and exiting from Speed. Sample sessions are included.

Chapter 3, "Speed Commands and Modifiers," discusses commonly-used and advanced commands. It also shows command modifiers.

Chapter 4, "Command Dictionary," lists commands in alphabetical order.

Appendix A contains the ASCII character set.

Appendix B lists Speed error messages.

Conventions and Abbreviations

Some special conventions used in this manual are:

→	Tab character, typed as CTRL-I.
↓	New-line, typed as CTRL-J.
$	Speed command delimiter, typed with Escape key.
$$	Speed command line terminator, typed as CTRL-D.
^ X or ↑X	Control character, typed by pressing the CTRL key and another character at once. (Depending on the terminal you are using, ^ or ↑ is the ASCII character 136_8.)
COMMAND	Uppercase letters in THIS typeface indicate an instruction mnemonic. You type an instruction mnemonic exactly as it appears.
argument	Lowercase italic letters represent a command's argument. You must replace this symbol with the exact code for the argument you need.
[*optional*]	Brackets denote an optional argument. (Command switches appear in this format as well.) If you use this argument or switch, do not type the brackets into your command line: they only set off the choice.
EXAMPLE LINE	Uppercase letters in THIS TYPEFACE are used for programming examples.
RESPONSE	If the program can respond to the command in the example, the response is shown in uppercase letters in *THIS TYPEFACE*.

Related Manuals

The following manuals also belong to the series of books published on the MP/AOS operating system.

MP/AOS Concepts and Facilities (DGC No. 069-400200) provides a concise but thorough introduction to the MP/AOS operating system for users who want to assess the system's advantages.

MP/AOS System Programmer's Reference (DGC No. 093-400051) documents MP/AOS system structure and provides a complete dictionary of system calls and library routines.

MP/AOS Command Line Interpreter (CLI) (DGC No. 069-400201) describes the interactive CLI program, the user's primary interface to the MP/AOS system. A command dictionary provides command descriptions, formats, and examples.

Loading MP/AOS (DGC No. 069-400207) describes how to install MP/AOS software on ECLIPSE-line computers and how to load tailored systems.

MP/AOS System Generation and Related Utilities (DGC No. 069-400206) describes the generation of an MP/AOS system tailored to specific applications. It also describes the following utilities, including sample dialogues as appropriate:

- SYSGEN, the interactive system generation utility;
- DINIT, the disk initializer;
- FIXUP, the disk repair utility;
- SPOOLER, which controls line printer operations;
- ELOG (error logger), the utility for interpreting the system log file.

MP/AOS Debugger and Performance Monitoring Utilities (DGC No. 069-400205) describes the following utilities, providing a dictionary of debugger commands and sample dialogues as appropriate:

- FLIT, the process debugger;
- PROFILE, which measures execution-time performance;
- OPM, the process monitor that displays current system resource allocation and status.

MP/AOS Macroassembler, Binder, and Library Utilities (DGC No. 069-400210) documents the MP/AOS macroassembler and binder as well as the library file editor (LED) and system cross-reference analyzer (SCAN). It includes programming examples and a dictionary of assembler pseudo-ops.

MP/AOS Advanced Program Development Utilities (DGC No. 069-400208) describes the following utilities:

- Text control system (TCS), a method for managing different versions of a single file;
- BUILD, which creates a new version of a file from existing files, thus minimizing effort and errors in program development;
- FIND, which locates occurrences of strings in text files.

MP/AOS SLATE Text Editor (DGC No. 069-400209) documents the features of SLATE, a screen- and line-oriented text editor.

MP/AOS File Utilities (DGC No. 069-400204) describes the following utility programs, providing sample dialogues for each:

- FEDIT, a file editor that permits modification of system files, including program and data files;
- FDISP, which can display the address and data contents of a file or compare two files, displaying the parts that differ;
- SCMP, which can compare two source programs line by line;
- MOVE, which allows the transfer of files among directories;
- AOSMIC, which allows manipulation of MP/AOS and MP/OS disks and files on an AOS system;
- FOXFIRE, which permits the transfer of files among MP/OS, MP/AOS, and AOS systems over asynchronous communication lines.

SP/Pascal Programmer's Reference (DGC No. 069-400203) documents an extended Pascal for system programmers. SP/Pascal has all of the features of MP/Pascal as well as special features targeted for the MP/AOS and AOS operating systems.

Books on three additional programming languages supported by MP/AOS have previously been published as part of the bookset for the MP/OS operating system:

MP/Pascal Programmer's Reference (DGC No. 069-400031) documents for system programmers a Pascal-based language targeted for the MP/OS operating system.

MP/FORTRAN IV Programmer's Reference (DGC No. 069-400033) documents for system programmers a language based on ANSI 1966 standard FORTRAN with extensions.

MP/Basic Programmer's Reference (DGC No. 069-400032) documents for new users a programming language based on ANSI standard Basic with extensions.

MP/OS

For information on Microproducts and a bibliography of documentation on the Microproducts line, see *Introduction to Microproducts* (DGC No. 014-000685).

For information on cross development between MP/OS and MP/AOS, see *MP/OS System Programmer's Reference* (DGC No. 093-400001).

Contents

Figures

Tables

1 Speed Concepts and Syntax

Speed is a character-oriented text editor, and this implementation runs on 16-bit ECLIPSE® computers under MP/AOS.

Using Speed, you can easily create text files, modify existing files, and merge data contained in two or more files. You can also perform very sophisticated editing operations on multiple files by employing the more advanced commands.

This chapter introduces Speed concepts and syntax, and since you will usually use Speed from a console, you will first learn how to correct typing errors.

Correcting Typing Errors

Occasionally, you will make typing mistakes while entering command characters at the console. The system provides you with three methods of correcting these errors.

Erase a Character

You can erase the last character you entered at the console by striking the Delete key. On a video display, the last character disappears, and the cursor moves one space to the left. Hardcopy terminals echo a backslash followed by the deleted character.

If you delete several characters in one line on a hardcopy terminal, the line may become difficult to read. You can retype the entire current line by striking the CTRL and T keys at the same time. (We will use the term CTRL-T to denote this action.)

Erase a Line

You can erase an entire line by typing CTRL-U. On video terminals, the entire line disappears and the cursor moves to the beginning of the line. Hardcopy terminals echo ↑U. In both cases you can immediately type a new line of characters.

Erase Several Lines

A line is defined as a string of characters ending in a New-line, Carriage Return, Form Feed, or null character. You may enter Speed command strings that span two or more lines. If you want to cancel input that spans more than one line, type CTRL-C CTRL-A. Speed cancels all of the characters entered from the prompt (!) on.

Units of Text

We define text as a sequence of one or more ASCII characters. A complete list of the ASCII character set appears in Appendix A. The units of text that Speed recognizes are:

- Character
- String
- Line
- Page
- Window

Character

A character can be any single ASCII alphanumeric character. Each character occupies a single position in the edit buffer, the area in memory where Speed processes your text. Some of the nonalphabetic ASCII characters do not print on your video display (for example, New-line or CTRL-J), and some print as more than one symbol. Nevertheless, they still occupy a single character position in the stored text.

Initially, Speed does not distinguish between uppercase and lowercase alphabetic characters used for command names and search strings. However, you can tell Speed to distinguish between uppercase and lowercase with a case control command.

String

A string or character string is a sequence of any ASCII characters. There are two special-purpose characters which are not usually part of a string, Escape and CTRL-D. Escape is the command delimiter, and CTRL-D is the command line terminator.

Line

A line is a character string ending in the New-line character (CTRL-J). Each line of text you type at the console appears as a separate line on your video display.

Page

A page is a sequence of characters ending in a Form Feed character (CTRL-L). You usually read text from a file into the edit buffer for processing in pages. A page of text is not limited in size, and its size may be altered. You can create a page by inserting a Form Feed into the text, and you can merge two adjacent pages by deleting the Form Feed that separates them.

Window

Besides pages, Speed allows you to read text into the edit buffer in windows. A window is a fixed number of lines of text. It has no delimiter, and can contain any ASCII character including a Form Feed. Window units allow you to read text from a file into the edit buffer in segments whose length you preset. A window length of 20 lines will fit on a console screen, and a window length of 60 lines will fit on a line printer page.

To read text in windows, you must turn Window Mode on and specify the window length (see Chapter 3).

Buffers and Text Files

Speed performs editing operations by reading text from an input file into an edit buffer, modifying the contents of the buffer, and writing the contents of the buffer to the output file (which is therefore the edited version of the input file).

Buffers

Speed has 36 edit buffers, named 0 through 9 and A through Z. You can store text to be modified in as many of these buffers as you require, but at any given time there is only one current buffer. The current buffer is the buffer in which text is currently being modified. The current buffer is initially Buffer 0, but you can issue a Speed command, for example, to change the current buffer from Buffer 0 to Buffer 4.

Character Pointer

Your present position for editing text within the current buffer is indicated by the character pointer (CP). The character pointer is always placed *between* two characters, and it is displayed as a flashing asterisk (*). (On terminals other than 605*x*, the CP is displayed as a circumflex (^).) You can move the CP around in the buffer with character pointer commands (see Chapter 3).

Files

As stated above, Speed reads the text you want to edit from an input file, and then writes the modified text to an output file. When you open files for input or for output, the specific command used determines whether the file is global or local. You can transfer text to or from any of the 36 edit buffers if the file is global. However, a local file can only be accessed from one edit buffer (the current buffer at the time the file was opened).

New Speed users will probably find it easier to work with global files initially.

Speed Command Format

The generalized form of a Speed command is as follows:

![*mod*][*num*]COMMAND[<*str* >*del* [<*str*>*del*]]*term*

where

! is the Speed prompt, indicating Speed will accept commands.

mod is one of the optional command modifiers @, :, or &.

num is one or two numeric arguments.

COMMAND is a Speed command mnemonic.

<*str*> is a string argument.

del is a delimiter (usually Escape with appears as $), required with string arguments.

term is the command line terminator, CTRL-D (which appears as $$).

A Speed command string includes one or more Speed commands. The generalized form of a command string is as follows:

COMMAND[[*del*][...*command*]]...*term*

Commands, Modifiers, and Terminators

A Speed command, as can be seen above, consists of a command mnemonic together with optional arguments and modifiers. Chapter 3 introduces all the Speed commands divided into functional categories, and Chapter 4 describes each command in detail.

Many of the Speed commands accept modifiers which alter the action taken by the command. These modifiers (:, @, and &) are described collectively at the end of Chapter 3, and the descriptions of the relevant commands in Chapter 4 include a full discussion of the effects of modifiers.

A Speed command string consists of one or more commands. Every command line must be terminated by CTRL-D (which is echoed as $$). Speed scans the command line, but will not start to execute commands until it has found the terminator.

If a command string contains more than one command, the commands may be separated by the standard Speed command delimiter, Escape (which is echoed as $). Although it is not necessary to delimit the commands in this way, the beginner will probably find it easier to check that the commands are correct if they are delimited. For example, here is a command string containing two Speed commands (Y and T in this case):

```
!Y$T$$
```

The command string

```
!YT$$
```

has the same effect, but it is not as obvious that there are two commands in the string.

You should note that the exclamation point (!) is the Speed prompt. It tells you that Speed is ready to accept commands.

Numeric Arguments

A numeric argument is a number or an expression which evaluates to a number. Initially, Speed interprets numbers as decimal, but you can specify that any number should be interpreted in an alternate radix (octal, binary, etc.).

Some Speed commands accept a single numeric argument (denoted by *n* in this manual), and some accept two numeric arguments (denoted by *m,n*).*

When a single numeric argument precedes a command that accepts both positive and negative numeric arguments, it may have any value within the range −32768 to 32767. When a numeric argument precedes a command which accepts only positive numeric arguments (J, \ or =), it may have any value within the range 0 to 65535.

In a double numeric argument (*m,n*), either argument (*m* or *n*) may have a value within the range 0 to 65535. Remember that double numeric arguments must always be separated by a comma, and neither of them may be negative. The first argument (*m*) must have a value less than or equal to that of the second argument (*n*).

**If a numeric argument is a mathematical expression, it may contain any digits, numeric operators, special symbols, pseudo variables that represent numeric values, and numeric variables.*

Examples of Speed commands with single numeric arguments are:

```
!10T$$
```

which types 10 lines from the current buffer, starting at the line containing the CP, and

```
!-7D$$
```

which deletes from the current buffer 7 characters preceding the CP.

Examples of Speed commands with two numeric arguments are:

```
!5,9T$$
```

which types out the 6th through 9th characters in the current buffer, and

```
!9,25K$$
```

which deletes the 10th through 25th characters from the current buffer.

Special Symbols and Pseudo Variables

There are several special symbols and pseudo variables (see Table 1.1) which represent specific values associated with the position of the character pointer (CP) within the current buffer. You can use the special symbols and pseudo variables as numeric arguments.

Symbol	Represents
Z	The total number of characters in the current buffer.
.(period)	The number of characters between the beginning of the buffer and the current CP position.
#	A special double argument which is equivalent to 0,Z.
VC	The numeric value of the ASCII character following the CP. If there are no characters after the CP, VC is zero.
VM	The number of characters between the beginning of the current line and the CP.
VN	The total number of lines in the current buffer.
VL	The number of lines between the beginning of the buffer and the CP, including the line containing the CP.
VP	The number of characters between the beginning of the buffer and the CP position just before the last search. If there was no previous search, or if the search started in a previous page, the value of VP is zero.

Table 1.1 Speed pseudo variables

For example,

```
!ZM$$
```

moves the CP Z characters forward in the current buffer (that is, to the end of the buffer).

```
!-VMD$$
```

deletes the VM characters preceding the CP from the current buffer (that is, deletes all the characters between the beginning of the line and the CP).

Numeric Variables

Speed has ten numeric variables, named V0 through V9. Each of the ten variables is initially set to zero, but you can use the numeric variable commands to set the variables individually to values, increment them, or decrement them. You can use numeric variables as numeric arguments to commands, either alone or in argument expressions.

For example, if variable V0 has been set to 5, the command:

```
!V0D$$
```

deletes the five characters following the CP from the current buffer.

Numeric Operators

Table 1.2 lists the numeric operators. None of the operators has precedence over the others, and numeric arguments containing operators are always evaluated from left to right. Speed evaluates numeric arguments before it starts to execute the command.

Operator	Function
+	Addition
-	Subtraction
*	Multiplication
/	Division
↑+	Boolean inclusive OR. This binary operator produces a full word result with each bit representing the Boolean OR of the corresponding bits of the operands.
↑-	Boolean NOT (complement). This unary operator precedes its operand and produces a full word bit-by-bit complement of the operand.
↑*	Boolean AND. This binary operator produces a full word result with each bit representing the Boolean AND of the corresponding bits of the operands.
↑/	Boolean XOR (exclusive OR). This binary operator produces a full word result with each bit representing the Boolean XOR of the corresponding bits of the operands.

Table 1.2 Speed numeric operators

Some examples of the use of the numeric operators follow.

```
!V0+5D$$
```

If V0 has been set to ten, this command deletes the 15 characters following the CP from the current buffer.

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

This command types out every line from the current line to the end of the buffer.

```
!Z-60,ZT$$
```

This command types out the last 60 characters in the current buffer.

String Arguments

A string argument consists of any sequence of ASCII characters followed by a delimiter (usually Escape, which is echoed as $). Some Speed commands accept a single string argument, and some accept two string arguments.

An example of a command with a single string argument is:

```
!SLOGICAL$$
```

which searches for the string LOGICAL, and places the CP after it. Notice that in this case the string argument, LOGICAL, is delimited by CTRL-D, which also terminates the command line.

An example of a command with two string arguments is:

```
!CPROGRAM$MODULE$$
```

which searches for the string PROGRAM, replaces it with the string MODULE, and places the CP after the E. Notice that the first string argument, PROGRAM, is delimited by Escape and the second string argument, MODULE, is delimited by CTRL-D, which also terminates the command line.

Using Speed

2

This chapter discusses the steps you must perform when editing text with Speed. It then describes several sample editing sessions.

A standard edit cycle consists of the following steps:

1. Invoke Speed.
2. Open files for input and output.
3. Read text from the input file into the edit buffer.
4. Edit the text in the buffer.
5. Write the text from the edit buffer into the output file.
6. Close the input and output files.
7. Exit from Speed.

The first step is always to issue a Command Line Interpreter (CLI) command to invoke Speed. The particular form of the command you issue determines whether you simply accomplish Step 1, or Steps 1, 2, and 3.

Invoking Speed

The CLI command line used to invoke Speed is:

XEQ SPEED[/*switch*][*pathname*]

where

XEQ SPEED means execute Speed (this can be abbreviated to X SPEED).

[/*switch*] is one or more of the optional command switches shown in Table 2.1.

[*pathname*] is an optional argument specifying the file you want to edit.

Switch	Action
/D	Turns Display Mode on, setting the Display Mode value, WD, to 10. This means that Speed will display the 20 lines surrounding the CP position. This enables you to see the effects of your commands on the text in the current buffer, without continually having to issue Type-Out commands.
/I=*pathname*	Indicates to Speed that all commands are to be taken from the file specified by *pathname*. When Speed reaches the end of file, it terminates itself. This enables you to write and use edit programs to solve editing problems.

Table 2.1 Optional command switches

The effects of the different variations of the XEQ SPEED command are illustrated in the following examples.

Example 1

In this example you want to edit a text file that already exists.

```
) XEQ SPEED FILE1 ↵
SPEED REV mm.nn
!
```

FILE1 is the name of the file you want to edit. Speed starts up and displays its revision number. It then opens FILE1 for input, opens a new file which it calls FILE1.TM for output, and reads a page of text from the input file into the edit buffer. Finally, it displays the prompt (!) which means that it is ready to accept commands.

If you then give the Filestatus command:

```
!F?$$
```

Speed displays a report that is similar to the following.

```
Global:
    Input File - PATHNAME:FILE1
    Output File - PATHNAME:FILE1.TM
    Update Mode On
  Local:
    Input File - None
    Output File - None
```

This version of the XEQ SPEED command opens FILE1 as a global input file and creates FILE1.TM as a global output file, turning Update Mode on. You will see the significance of this later.

Example 2

In this example you want to enter text into a new file.

```
) XEQ SPEED FILE2 ↓
SPEED REV mm.nn
Create new file? Y↓
!
```

FILE2 is the name of the new file you want to create. Speed starts up and displays its revision number. It then asks you if you want to create the new file. You must confirm by typing Y and a New-line (↓). Speed then creates FILE2 as an output file. (There is, of course, no input file.) Finally, it displays the prompt (!) which means that it is ready to accept commands. If you then issue the Filestatus command:

```
!F?$$
```

Speed responds with a report that is similar to the following:

```
Global:
    Input File - None
    Output File - PATHNAME:FILE2
Local:
    Input File - None
    Output File - None
```

which confirms that FILE2 is the global output file.

Example 3

In this example you want to invoke Speed without opening or creating a file at the same time.

```
) X SPEED ↓
SPEED REV mm.nn
!
```

Speed starts up, displays its revision number, and prompts for commands. The edit buffer is empty, and no files are open. The following section describes the next steps you must take.

Opening Files for Input and Output

If you invoked Speed with an XEQ SPEED *pathname* command, you do not have to issue specific commands to open input and output files. However, if you omitted the *pathname* argument, you will need to open input and output files. As stated earlier, you will probably find it easier to work with global files rather than local files if you are new to Speed. We will therefore discuss only commands that open or close global files.

Three commonly used file-open commands are File Read (FR), File Write (FW), and File Open (FO).

File Read (FR)

This command opens an existing file as the global input file. For example:

```
!FRFILE1$$
```

opens an existing file called FILE1 as the input file. You must issue a file input command to read text from FILE1 into the edit buffer.

File Write (FW)

This command creates and opens a new file as the global output file. For example:

```
!FWFILE2$$
```

creates a new file called FILE2, and opens it as the output file. You must issue a file output command to write text from the edit buffer into FILE2.

File Open (FO)

The File Open command opens an existing file as the global input file, reads a page of text from the file into the edit buffer, and creates and opens a global output file. For example:

```
!FOFILE1$$
```

opens an existing file called FILE1 as the global input file and reads the first page of text from FILE1 into the edit buffer. It then creates a file called FILE1.TM and opens it as the global output file.

This command turns global Update Mode on for FILE1. This means that if, after your editing session, you close your input and output files with an FU (File Update) command, Speed closes and deletes the input file FILE1, closes the output file FILE1.TM, and renames FILE1.TM to FILE1. The new FILE1 is simply an edited (updated) version of the original FILE1.

Reading Text into the Edit Buffer

You must read text from the input file into the edit buffer before you can issue editing commands. You can read text into the buffer in window lengths or page lengths. Initially, Speed is in Page Mode and you must issue a Window Mode command before you can read text in window lengths. In the following discussion we assume that Speed is in Page Mode, so text is read from the input file from Form Feed to Form Feed.

If you used the XEQ SPEED *pathname* command to invoke Speed, or if you opened your input and output files with the FO command, the first page of the input file was automatically read into the edit buffer.

Two common commands that you can use to read text into the buffer are Yank (Y) and Read (R).

Yank (Y)

This command reads a page of text from the input file into the edit buffer, overwriting the previous contents of the buffer. If the buffer is not empty and Update Mode is on when you issue a Y command, Speed will ask for confirmation. You should type Y to acknowledge that you want to overwrite the contents of the buffer:

```
!Y$$
Confirm (Y-command) ? Y ↵
!
```

Read (R)

This command writes the contents of the buffer into the output file, then reads the next page from the input file into the buffer:

```
!R$$
!
```

Editing Text

You can now start editing text in the current edit buffer. At all times, Speed marks your current position in the buffer with the character pointer (CP). (You will not see the CP unless Speed is in Display Mode, or unless you issue a text Type-Out command.) When you read text into the buffer, the CP is always positioned before the first character in the buffer. All the commands that modify text use the CP as a reference point.

The following paragraphs will give you an indication of the sorts of editing operations you can perform with Speed. For more information about the various types of Speed commands, see Chapter 3, which divides all the Speed commands into functional categories.

In order to illustrate some edit commands, let us assume that the text in Figure 2.1 has been read into the buffer.

```
The system can be used either
for general purpose applications
oriented toward program development,
or for smaller stand-alone
applications such as rael-time
process control. You can put
all the software in read-only
memory (RAM) to eliminate the need
for mass storage, or you can
take advantage of the powerful
file management system for storing
large amounts of data on disks.
```

Figure 2.1 Text editing example

Text Type-Out

First, we want to read the contents of the buffer. T is the Type-Out command, and # is a special numeric argument (equivalent to 0,Z) which means *all the characters in the buffer.* Speed displays the contents of the buffer.

```
!#T$$
The system can be used either
for general purpose applications
oriented toward program development,
or for smaller stand-alone
applications such as rael-time
process control. You can put
all the software in read-only
memory (RAM) to eliminate the need
for mass storage, or you can
take advantage of the powerful
file management system for storing
large amounts of data on disks.
!
```

Move Character Pointer

Character pointer commands are either character-oriented (M) or line-oriented (J or L). The CP is currently located before the first character in the buffer, and we want to move it four positions to the right so that it is immediately before the *s* in *system.* (Note that a space is counted as a character.)

```
!4M$$
!
```

M is the character-oriented CP Move command, and the argument 4 specifies *four characters to the right.* To see the position of the CP, we can issue the T command without any arguments, which types out the current line:

```
!T$$
The *system can be used either
!
```

If you do not have a 605*x* video display, this line would look like:

```
!T$$
The (^) system can be used either
!
```

The CP location is indicated by (^) rather than by a flashing asterisk.

Insert Text

We now want to insert an extra word at the CP location. To do this we use the I command. We will also type out the current line:

```
!Ioperating $$
!T$$
The operating *system can be used either

!
```

As you can see, Speed has inserted the character string *operating* at the CP location. Notice that we delimited the character string with CTRL-D, which also terminates the command line.

We will now move the CP to the beginning of the buffer:

```
!1J$$
!
```

This line-oriented CP Jump command, J, moves the CP to the beginning of the first line in the buffer.

Search

There is a spelling error in the text in the current buffer: *rael-time* should be *real-time*. We can use the search command C to correct this error:

```
!Crael-time$real-time$$
!T$$
applications such as real-time*
!
```

The C command searches from the CP location (the beginning of the buffer in this case) through the buffer until it finds *rael-time* which it replaces with *real-time*. Notice that Escape ($) delimits the first character string, and that CTRL-D ($$) delimits the second character string and terminates the command line.

We can use another search command, S, to locate the character string *(RAM)* which we will delete from the buffer. First we will move the CP to the beginning of the buffer:

```
!J$$
!S(RAM)$T$$
memory (RAM)* to eliminate the need
!
```

Here, J (which is the same as 1J) moves the CP to the beginning of the first line in the buffer. S(RAM)$ searches from the CP location through the buffer for the character string *(RAM)*, and positions the CP after the last character in the string. T types out the current line.

Notice that we used Escape to delimit the search string *(RAM)*, and that we issued two commands (S and T) in the same command line.

Delete Text

We can use a delete command to erase *(RAM)* from the buffer. The two delete commands are D, which is character-oriented, and K, which is line-oriented (unless it has a double argument). Bearing in mind the current CP position, we use the following command to delete *(RAM)*.

```
!-6D$$
!
```

This command deletes the six characters preceding (that is, to the left of) the CP.

Finally, we will move the CP to the beginning of the buffer, and type out the entire contents of the buffer:

```
!-7L$#T$$
The operating system can be used either
for general purpose applications
oriented toward program development,
or for smaller stand-alone
applications such as real-time
process control. You can put
all the software in read-only
memory to eliminate the need
for mass storage, or you can
take advantage of the powerful
file management system for storing
large amounts of data on disks.
!
```

The command -7L moves the CP to the beginning of the seventh line preceding the CP (that is, to the beginning of the first line in the buffer). Then, #T types out the entire contents of the buffer.

We have again put two commands, separated by Escape, in the same command line. The following command line (without Escape) has the same effect:

```
!-7L#T$$
```

Writing Text to the Output File

When you finish editing the text in the edit buffer, you can write it to the output file using a Put (P) or Read (R) command.

Put (P)

This command writes the text with an appended Form Feed character from the edit buffer into the output file:

```
!P$$
!
```

Read (R)

This command writes the contents of the buffer into the output file, then reads the next page from the input file into the buffer:

```
!R$$
!
```

Closing Input and Output Files

After writing the text from the edit buffer, you can close the input and output files. Speed provides you with a number of commands which close files, including File Update (FU), File Backup (FB) and File Close (FC). If Update Mode is on (that is, you opened an existing file for input with an XEQ SPEED *pathname* command, or you opened the input file with an FO command), you should use FU or FB to close the files (see the section on File Open (FO)).

File Update (FU)

This command writes the contents of the edit buffer to the output file, copies the remainder of the input file to the output file, and closes both files. It then deletes the input file and renames the output file.

For example, if the input file was FILE1 and the output file was FILE1.TM, the command:

```
!FU$$
!
```

copies the buffer and the remainder of FILE1 into FILE1.TM, closes FILE1 and FILE1.TM, deletes FILE1, and renames FILE1.TM to FILE1. The new FILE1 is simply the edited (updated) version of the original FILE1.

File Backup (FB)

This command writes the contents of the edit buffer to the output file, copies the remainder of the input file to the output file, closes both files, and clears the buffer. It then renames the input and output files.

For example, if the input file was FILE1 and the output file was FILE1.TM, the command:

```
!FB$$
!
```

copies the buffer and the remainder of FILE1 into FILE1.TM, closes FILE1 and FILE1.TM, clears the buffer, renames FILE1 to FILE1.BU, and renames FILE1.TM to FILE1. The new FILE1 is the edited version of the original FILE1, and the backup FILE1.BU is the original FILE1.

File Close (FC)

This command closes the input and output files. It does not write the contents of the buffer into the output file.

```
!FC$$
!
```

Exiting from Speed

You exit from Speed with the Halt (H) command. If an input file or an output file is open, or if any text remains in the buffer, Speed will ask for confirmation. You must reply by typing Y before Speed terminates. When Speed has terminated, you will see the CLI prompt ( )) if Speed was executed from the CLI.

```
!H$$
Confirm? Y ↓
)
```

Sample Sessions

Before discussing the complete set of Speed commands in detail, we will further illustrate some text editing functions with three sample editing sessions.

Example 1

In this session we will:

- Create a file and open it for output.
- Insert several lines of text into the edit buffer.
- Type out some of the characters we inserted into the edit buffer.
- Write the contents of the edit buffer to the output file and close the file.
- Terminate Speed.

The entire session is as follows:

```
) XEQ SPEED FILE1 ↓
SPEED REV mm.nn
Create new file? Y ↓
!IABCDE ↑J
FGHIJK ↑J
LMNOPQ$$
!0,13T$$
ABCDE
FGHIJK
!FU$$
!H$$
)
```

XEQ SPEED FILE1 creates FILE1 and opens it as the global output file.

```
!IABCDE↑J
FGHIJK↑J
LMNOPQ$$
```

inserts the character string in the buffer at the CP location. Notice that when you enter a New-line character as part of your character string, the operating system echoes it by starting a new line, not by returning CTRL-J.

The command 0,13T types out the first 13 characters in the buffer (A through the New-line character following K). If we had specified the first 12 characters to be typed out, the display would have been:

```
!0,12T$$
ABCDE
FGHIJK
```

FU writes the contents of the buffer into the output file, and closes the output file. H terminates Speed, and puts you back in the CLI.

Example 2

In this example, we will:

- Open a three-page file for input.
- Create and open an output file.
- Read the first page of the input file into the buffer.
- Edit the contents of the buffer.
- Write the contents of the buffer and the rest of the input file into the output file.
- Close the input and output files.
- Exit from Speed.

The entire session is as follows:

```
) XEQ SPEED ↓
SPEED REV mm.nn
!FRFILE1.IN$$
!FWFILE1.OUT$$
!Y$#T$$
.
.
(Speed displays the contents of the buffer.)
.
.
!Sstring1$Istring2$$
!4L$Cstring3$string4$$
!#T$$
.
.
(Speed displays the contents of the buffer.)
.
.
!E$$
!FC$H$$
)
```

XEQ SPEED invokes Speed.

FRFILE1.IN$$ opens FILE1.IN (a file that contains three pages of text) as the input file. FWFILE1.OUT$$ creates FILE1.OUT and opens it as the output file. Notice that for both of these commands, the string argument was delimited by CTRL-D, which also terminated the command line.

Y reads the first page from the input file into the buffer, and #T types out the entire contents of the buffer. Notice that we have issued two commands in the same command line, separating them with Escape.

Sstring1$ searches from the CP location through the buffer for *string1* and positions the CP immediately after the last character of *string1*. Istring2$$ inserts *string2* into the buffer at the CP location. Notice that we delimited *string1* with Escape and *string2* with CTRL-D, which also terminated the command string.

4L moves the CP to the beginning of the fourth line following its present location.

Cstring3$string4$$ searches from the new CP location through the buffer for *string3* and replaces it with *string4*.

#T types out the entire contents of the buffer.

E writes the contents of the buffer and the remainder of the input file into the output file.

FC closes the input and output files, and H terminates Speed.

Example 3

This final example is much more complicated than the preceding ones, and it demonstrates some of Speed's more advanced commands. It uses nested command loops to change the contents of a file from variable page lengths to fixed page lengths (20 lines per page).

We will perform the following steps:

- Turn on Window Mode and set the window length to 20 lines.
- Open the file for input and output.
- Read a window of text from the input file into the buffer.
- Repeat the following sequence of events (the outer loop) until no more text remains in the input file.

1. Repeat the following sequence of events (the inner loop) until no Form Feeds remain in the current contents of the buffer:

   a. Search for a Form Feed character.

   b. Delete the Form Feed character.

2. Jump to the end of the buffer.

3. Insert a Form Feed character.

4. Write the contents of the buffer into the output file, and read a window of text from the input file into the buffer.

- Close the input and output files.
- Exit from Speed.

The entire session is as follows:

```
) X SPEED )
SPEED REV mm.nn
!10WD$$
!20WM$$
!FOFILE4$$
!<<S↑L$;-1D>ZJ$I↑L$R;>$$
!FBH$$
)
```

X SPEED invokes Speed.

10WD tells Speed to display as much as it can of the 10 lines preceding and the 10 lines following the CP, showing the CP location as a flashing asterisk. All this happens before Speed returns the prompt.

20WM turns Window Mode on and sets the window length to 20 lines.

FOFILE4$$ opens FILE4 as the input file, creates and opens a file called FILE4.TM as the output file, and reads a window of text from the input file into the buffer.

The next command line contains the nested command loops. The inner loop is <S↑L$;-1D>. The first command S↑L$ searches from the CP location (initially the beginning of the buffer) through the buffer for an occurrence of CTRL-L (the Form Feed character), and positions the CP immediately after the first occurrence of CTRL-L. The semicolon tells Speed to terminate the inner loop (and continue execution at the first command following the loop) if the S command fails. -1D deletes the character preceding the CP (CTRL-L in this case). Execution of the command loop continues until the S command fails.

The outer loop is <*Inner-Loop*ZJ$I↑L$R;>. The inner command loop is executed first. When the S command fails, Speed executes the first command following the inner loop (ZJ) which positions the CP after the last character in the buffer. I↑L$ inserts a Form Feed character at the CP location. R writes the contents of the buffer into the output file, clears the buffer, and reads the next window of text from the input file into the buffer. The semicolon command terminates the outer loop if the R command fails (that is, when there is no more text in the input file). Execution of the outer command loop continues until the R command fails.

Notice that when you enter CTRL-L (the Form Feed character) in the command line, the operating system echoes it by starting a new page, not by returning ↑L. So, the nested loops
<<S↑L$;-1D>ZJ$I↑L$R;> display as follows:

```
!<<S
.
.
.
(Form Feed character echoed)
.
.
.
$;-1D>ZJ$I
.
.
(Form Feed character echoed)
.
.
.
$R;>
!
```

Finally, FB closes the input and output files, renames the input file, FILE4, to FILE4.BU, and renames the output file, FILE4.TM, to FILE4. H terminates Speed. Notice that it is not necessary to separate FB from H with the delimiter, Escape.

Speed Commands and Modifiers

3

This chapter presents the Speed commands divided into functional categories. It consists of three sections.

Commonly-used commands are: commands that open and close files, File Input commands, Text Type-Out commands, Character Pointer commands, Search commands, Insertion commands, Deletion commands, Buffer commands, File Output commands, and the exit command.

Advanced commands are: Input Mode commands, Case Control commands, Numeric Argument commands, Numeric Variable commands, Iteration commands, Flow Control commands, Program Execution commands, and the Command Line Recall command.

The three command modifiers are :, @, and &.

Chapter 4, "Command Dictionary," provides more information about each of these commands.

Commonly-Used Commands

This section describes the ten types of Speed commands that you use most commonly in editing.

Commands to Open and Close Files

When you open an input file or an output file, you must specify the file to be local to the current buffer, or global to all 36 buffers. A file that is not associated with a single buffer is global You can access global files regardless of which buffer is the current buffer, but you can access a local file only when its associated buffer is the current buffer.

Each buffer can have only one local input file and one local output file open at a time. Only one global input file and one global output file can be open at a time.

The commands listed in Table 3.1 open and close files. Commands beginning with B, such as BFR and BFW, specify local files, and commands beginning with F, such as FR and FW, specify global files.

Command	Action
BFB	Copies the current buffer and the remainder of the local input file to the local output file, clears the buffer, and closes both files. If Update Mode is on, creates a backup for the input file.*
BFC	Closes any local files associated with the current buffer.
BFNR$	Closes the current local input file without opening a new one. Cannot be used with local Update Mode on.
BFNR*pathname*$	Closes the current local input file and opens the file specified by *pathname* as the local input file. Cannot be used with local Update Mode on.
BFNW$	Closes the current local output file without opening a new one. Cannot be used with local Update Mode on.
BFNW*pathname*$	Closes the current local output file and opens the file specified by *pathname* as the local output file. Cannot be used with local Update Mode on.
BFO*pathname*$	Opens the file specified by *pathname* as the local input file, moves (yanks) a page or window from the local input file to the current buffer, and creates a new local output file (with the same name as the input file, but with the extension .TM). Turns Update Mode on.
BFR*pathname*$	Opens the file specified by *pathname* as the local input file.
BFU	Copies the current buffer and the remainder of the local input file to the local output file, and closes both files. If Update Mode is on, deletes the input file and renames the output file by giving it the input filename.*
BFW*pathname*$	Creates and opens the file specified by *pathname* as the local output file. The file must not already exist.
F?	Displays the status of the global and local files and indicates whether Update Mode is on.

Table 3.1 Commands to open and close files

**If local Update Mode is not on, the local output file is not renamed. If there is no local input file, only the buffer is copied.*

Command	Action
FB	Copies the current buffer and the remainder of the global input file to the global output file, clears the buffer, and closes both files. If Update Mode is on, creates a backup for the input file. *
FC	Closes all global files.
FNR$	Closes the current global input file without opening a new one. Cannot be used with global Update Mode on.
FNR*pathname*$	Closes the current global input file and opens the file specified by *pathname* as the global input file. Cannot be used with global Update Mode on.
FNW$	Closes the current global output file without opening a new one. Cannot be used with global Update Mode on.
FNW*pathname*$	Closes the current global output file and opens the file specified by *pathname* as the global output file. Cannot be used with global Update Mode on.
FO*pathname*$	Opens the file specified by *pathname* as the global input file, moves (yanks) a page or window from the global input file to the current buffer, and creates a new global output file (with the same name as the input file, but with the extension .TM). Turns Update Mode on.
FR*pathname*$	Opens the file specified by *pathname* as the global input file.
FU	Copies the current buffer and the remainder of the global input file to the global output file and closes both files. If Update Mode is on, deletes the input file and renames the output file by giving it the input filename. *
FW*pathname*$	Creates and opens the file specified by *pathname* as the global output file. The file must not already exist.

Table 3.1 Commands to open and close files (continued)

** If there are no global files, this command acts on local files. If there is no global input file but there is a global output file, then the buffer is copied to the global output file and the file is closed but not renamed.*

File Input Commands

The File Input commands listed in Table 3.2 read data from an input file into the current buffer. The commands read pages or windows of data into the buffer, depending on whether or not Window Mode has been turned on.

The Y and A commands accept the colon (:) modifier. This causes the command to return a 1 if the command was successful, and a 0 if the command failed.

When you issue file input commands, local files take precedence over global files; that is, the command applies to an open local file if one exists; otherwise, it applies to an open global file.

Command	Action
A	Reads one page or window from the input file and appends it to the end of the current buffer.
BFO*pathname*$	Opens the file specified by *pathname* as the local input file, moves (yanks) a page or window from the local input file to the current buffer, and creates a new local output file (with the same name as the input file, but with the extension .TM). Turns Update Mode on.
FO*pathname*$	Same as BFO, except that the input and output files are global instead of local.
R	Copies the current buffer to the output file, clears the buffer, and moves (yanks) a page or window from the input file into the current buffer.
*n*R	Performs the R command *n* times.
Y	Clears the current buffer and yanks a page or window from the input file into the current buffer.

Table 3.2 File Input commands

Text Type-Out Commands

The Text Type-Out commands listed in Table 3.3 allow you to examine part or all of the current buffer, or display data typed at the console. The commands do not move the character pointer (CP).

The *n*= command accepts the @, :, and & modifiers. The @ modifier suppresses the New-line character from the display; the : modifier sends the output to the line printer instead of the console; and the & modifier causes the value of *n* to be interpreted or displayed in the alternate radix, instead of in decimal.

Command	Action
T	Types the current line, indicating the CP position with a flashing asterisk (or as a circumflex (^) on some terminals).
0T	Types the current line from the beginning to the CP.
*n*T	Types the current buffer from the CP through *n* New-lines.
-*n*T	Types the *n* lines preceding the current line and the characters between the beginning of the current line and the CP.
m,*n*T	Types characters in the current buffer from the (*m* + 1)th character through the *n*th character.
@T%*string*%	Types the character string. A delimiter (% in this case) must appear between T and the string.
n=	Types the decimal value of the numeric argument *n* at the console.

Table 3.3 Text Type-Out commands

Character Pointer Commands

The CP marks your present position in the current buffer. The CP always resides between characters: before the first character in the buffer, between two characters in the buffer, or after the last character in the buffer.

You can move the CP by using the commands listed in Table 3.4. The J commands are line oriented and move the CP relative to the first line in the buffer. The L commands are also line oriented, but they move the CP relative to its current position. The M command is character oriented.

To display the current position of the CP in the buffer, use the T command. This displays the line containing the CP, showing the CP as a flashing asterisk (or as a circumflex (^) on some terminals).

Command	Action
J 0J 1J	Moves the CP to the beginning of the buffer.
*n*J	Moves the CP to the beginning of line *n*.
L 0L	Moves the CP to the beginning of the current line.
*n*L	Moves the CP to the beginning of the line following the *n*th subsequent New-line character.
-*n*L	Moves the CP to the beginning of the *n*th line preceding the current line.
*n*M	Moves the CP across *n* characters. If *n*>0, the CP moves to the right. If *n*<0, the CP moves to the left.

Table 3.4 Character Pointer commands

Search Commands

Speed executes a search command by attempting to match, character by character, the character string in the command line with portions of the current buffer. Character strings being searched for may not overlap page or window boundaries.

The commands listed in Table 3.5 cause Speed to scan through the current buffer until it finds the character string specified, and then position the CP after the last character in the string.

If the search reaches the end of the buffer (or the end of the specified range when arguments are used) without success when executing an S or a C command, Speed prints an error message on the terminal and positions the CP at the beginning of the buffer. For both N and Q commands, if Speed searches the last page in the input file without success, you are left with an empty buffer.

Command	Action
C*string1*$*string2*$	Searches from the current CP position to the end of the buffer for *string1* and replaces it with *string2*.
*n*C*string1*$*string2*$	Searches from the CP through the next *n* New-line characters for *string1* and replaces it with *string2*.
-*n*C*string1*$*string2*$	Searches through the preceding *n* lines and from the beginning of the current line to the CP for *string1* and replaces it with *string2*.
m,*n*C*string1*$*string2*$	Searches from the (*m* + 1)th character through the *n*th character in the current buffer for *string1* and replaces it with *string2*.
Any of the above C commands which omit *string1*, for example, C$*string2*$	Searches for an occurrence of the previous search string and replaces it with *string2*.
N*string*$	Searches the current buffer for *string*. If unsuccessful, copies the buffer to the output file, clears the buffer, and reads a new page or window from the input file. This continues until *string* is found or the end of the input file is reached.
N$	Same as the N*string*$ command, except that it searches for the previous search string.
Q*string*$	Same as N*string*$, except that it does not copy the buffer to the output file.
Q$	Same as N$, except that it does not copy the current buffer to the output file.
S*string*$	Searches from the current CP position to the end of the buffer for *string*, placing the CP at the end of *string* if the search succeeds.
*n*S*string*$	Searches from the current CP position through the next *n* New-line characters for *string*, placing the CP at the end of *string* if the search succeeds.
-*n*S*string*$	Searches through the preceding *n* lines and from the beginning of the current line up to the CP for the *string*, placing the CP at the end of *string* if the search succeeds.
m,*n*S*string*$	Searches from the (*m* + 1)th character through the *n*th character in the current buffer for *string*, placing the CP at the end of *string* if the search succeeds.
Any of the above S commands which omit *string*, for example, S$	Searches for an occurrence of the previous search string.

Table 3.5 Search commands

You can use the @ modifier with all Search commands. It changes the command delimiter from Escape ($) to the character immediately following the search command mnemonic S, N, C, or Q. For example, the command @S%*string*% is the same as S*string*$, except that the first character after S (% in this case) delimits the string.

You can also use the colon (:) modifier with all of the Search commands. When you precede a command with the : modifier, the command returns a value of 1 if it succeeds and 0 if it fails. No error message appears if the search was unsuccessful. The command can be used as a numeric argument to the next command.

You can combine the @ and the : command modifiers. For example, the following commands are legal:
@:S%*string*%
:@C%*string1*%*string2*%

Control Characters in Searches

You can include certain control characters in Search commands to alter the normal search process. Any number of these control characters may appear in the same search string.

The control characters listed and described in Table 3.6 act as templates in search strings. The examples in the table indicate how the templates act individually. However, if you combine search string templates, they can provide a very powerful matching mechanism. (Note that we represent CTRL-X as ↑X in the examples in the table.)

Some examples of the use of various combinations of search string templates are as follows:

↑N↑E which matches one or more of anything apart from space or tab. ↑X↑Z which matches anything to the end of the buffer. ↑\0123456789↑\ which matches any digit in this position in the search string. ↑N↑\0123456789↑\ which matches anything in this position in the search string except a digit. ↑Y↑01234567↑\ which matches any octal string.

Template	Action
CTRL-E	Matches one or more spaces or tabs. For example, A↑EA will match: A A and A A but not AA
CTRL-N	Matches any character except the character following the CTRL-N. Multiple occurrences of CTRL-N such as, ↑N↑Nx) have the same meaning as a single CTRL-N. For example, A↑NA will match: AF and A A but not AA
CTRL-T	Matches zero or more spaces or tabs. For example, A↑TA will match: A A and A A and AA
CTRL-X	Matches zero or more occurrences of the next character. For example, A↑XB will match: A and AB and ABBBBB
CTRL-Y	Matches one or more occurrences of the next character. For example, A↑YB will match: AB and ABBBBB but not A
CTRL-Z	Matches any single character.
CTRL*string*CTRL\\	Matches any one of the characters in the search string. For example, X↑\ABC↑\ will match: XA and XB and XC

Table 3.6 Search string templates

The control characters listed in Table 3.7 affect the CP position after a successful search. (The value of Position Mode affects the CP position following an unsuccessful search. See the section, "Speed Modes Affecting Searches").

Control Character	Action
CTRL-G	Positions the CP at the location of the ↑G in the search string after a successful search
CTRL-↑	Positions the CP at the location of the ↑↑ character in the search string after a successful search. (This character cannot be generated from a 605*x* keyboard.)

Table 3.7 Control of the CP after a search

The control characters listed in Table 3.8 indicate that the character following them should be interpreted literally. Therefore, you can use them to search for other control characters.

Control Character	Action
CTRL-W	Causes the next character to be interpreted literally rather than as a special character. Thus, you can use the template to search for search string templates and control characters. For example, A↑W↑EA will match: A↑EA and not A A
CTRL-_	Causes the next character to be interpreted literally rather than as a special character. Thus, you can use the template to search for search string templates and control characters. (This character cannot be generated from a 605*x* keyboard.) For example, A↑_↑EA will match: A↑EA and not A A

Table 3.8 Templates for literals in searches

Speed Modes Affecting Searches

Two of the Speed modes have direct relevance to Search commands: Position Mode and Search Case Match mode.

The value of Position Mode affects the positioning of the CP after an unsuccessful search, as shown in Table 3.9. You can set the value of Position Mode with the *n*WP command, or find the status of Position Mode with the WP command. (See Chapter 4 for further details.)

Search Command Format	CP Position if WP=0	CP Position if WP<>0
S	Beginning of buffer.	Position before search.
*n*S	Position before search plus *n* lines.	Position before search.
-*n*S	Position before search.	Position before search minus *n* lines.
m,*n*S	After the *n*th character in the buffer.	After the *m*th character in the buffer.

Table 3.9 Effects on searches of Position Mode

The value of Search Case Match mode determines whether Search commands will match alphabetical characters regardless of their case (uppercase or lowercase). The *n*WS command sets the value of Search Case Match mode, and the WS command returns the value of Search Case Match mode. (See Chapter 4 for further details.) If the value of Search Case Match mode is zero, matches will be case independent (both uppercase and lowercase will match either uppercase or lowercase); this is the initial (default) status. If the value of Search Case Match mode is not zero, searches will be case dependent (uppercase will match only uppercase, and lowercase will match only lowercase).

Insertion Commands

All of the Insertion commands listed in Table 3.10 insert a string of characters, specified in the command, into the current buffer at the CP position. The CP is then repositioned to follow the last character of the insertion.

You can use the @ modifier with the I command. It changes the command delimiter from Escape ($) to the character immediately following the I. For example, @I%*string*% is the same as I*string*$, except that the first character following the command mnemonic (% in this case) delimits the string. For the effects of modifiers on the C commands, see the section on Search Commands.

You can use the & modifier with *n*I and *n*\ commands. It causes the value of argument *n* to be interpreted in the alternate radix, instead of as a decimal value.

Command	Action
C*string1*$*string2*$	Searches from the CP to the end of the buffer for *string1* and replaces it with *string2*.
*n*C*string1*$*string2*$	Searches from the CP through the next *n* New-line characters for *string1*, and replaces it with *string2*.
-*n*C*string1*$*string2*$	Searches through the preceding *n* lines and from the beginning of the current line to the CP for *string1*, and replaces it with *string2*.
m,*n*C*string1*$*string2*$	Searches from the (*m* + 1)th through the *n*th characters in the current buffer for *string1*, and replaces it with *string2*.
Any of the above C commands which omit *string1*, for example, C$*string2*$	Searches for an occurrence of the previous search string and replaces it with *string2*.
I*string*$	Inserts *string* at the current CP location.
*n*I	Inserts the character whose ASCII decimal equivalent is *n* at the current CP location.
CTRL-I*string*$	Inserts a tab followed by *string* at the current CP location.
n\	Inserts the ASCII representation of the decimal number *n* at the current CP location.

Table 3.10 Insertion commands

Character String Insertion

The two Include-File references summarized in Table 3.11 allow you to insert characters into a command string. The characters inserted may form text, commands, or both.

Reference	Action
CTRL-B*x*	Replaces ↑B*x* in the command string with the contents of buffer *x*.
CTRL-F*pathname*$	Replaces ↑F*pathname*$ in the command string with the contents of the file specified by *pathname*.

Table 3.11 Include File Insertion commands

Deletion Commands

The commands listed in Table 3.12 delete characters and lines from the current buffer.

For the effects of command modifiers on the C command, see the section on Search Commands.

Command	Action
C*string1*$*string2*$	Searches from the CP to the end of the buffer for *string1*, and replaces it with *string2*.
*n*C*string1*$*string2*$	Searches from the CP through the next *n* New-line characters for *string1*, and replaces it with *string2*.
-*n*C*string1*$*string2*$	Searches through the preceding *n* lines and from the beginning of the current line to the CP for *string1*, and replaces it with *string2*.
m,*n*C*string1*$*string2*$	Searches from the (*m* + 1)th through the *n*th character in the current buffer for *string1*, and replaces it with *string2*.
Any of the above C commands which omit *string1*, for example, C$*string2*$	Searches for an occurrence of the previous search string and replaces it with *string2*.
*n*D	Deletes the *n* characters following the CP from the current buffer.
-*n*D	Deletes the *n* characters preceding the CP from the current buffer.
K 0K	Deletes all the characters between the beginning of the current line and the CP.
*n*K	Deletes *n* lines, starting at the CP through *n* New-line characters.
-*n*K	Deletes the preceding *n* lines and the characters between the beginning of the current line and the CP.
m,*n*K	Deletes the (*m* + 1)th through the *n*th characters from the current buffer.

Table 3.12 Deletion commands

Buffer Commands

You may store text to be modified, command strings, and so on in up to 36 edit buffers (named 0 through 9 and A through Z). Only one buffer, however, can be the current buffer at any time. You can use the commands listed in Table 3.13 to transfer data between buffers, and to manipulate buffers.

The BG*x* commands accept the colon modifier, which inhibits Speed from displaying a ? prompt.

Command	Action
BA*x*	Activates buffer *x*.
BC*x*	Copies the entire contents of the current buffer to buffer *x*, after clearing buffer *x*.
0BC*x*	Copies the characters between the beginning of the current line and the CP to buffer *x*, after clearing buffer *x*.
*n*BC*x*	Copies the next *n* lines, starting at the CP, to buffer *x*, after clearing buffer *x*.
-*n*BC*x*	Copies the *n* lines preceding the current line and the characters between the beginning of the current line and the CP into buffer *x*, after clearing buffer *x*.
m,*n*BC*x*	Copies the (*m*+1)th through the *n*th characters from the current buffer into buffer *x*, after clearing buffer *x*.
BG*x* 0BG*x*	Gets a line of characters typed at the console, and stores it in buffer *x*.
*n*BG*x*	Gets *n* characters typed at the console and stores them in buffer *x*.
BK*x*	Deactivates and deletes buffer *x*.
BS*x*	Makes buffer *x* the current buffer.
BT*x*	Copies the entire contents of the current buffer to buffer *x*, after clearing buffer *x*. Deletes all the characters copied from the current buffer.
0BT*x*	Copies the characters between the beginning of the current line and the CP to buffer *x*, after clearing buffer *x*. Deletes all the characters copied from the current buffer.
*n*BT*x*	Copies the next *n* lines, starting at the CP, to buffer *x*, after clearing buffer *x*. Deletes all the characters copied from the current buffer.
-*n*BT*x*	Copies the *n* lines preceding the current line and the characters between the beginning of the current line and the CP into buffer *x*, after clearing buffer *x*. Deletes all the characters copied from the current buffer.
m,*n*BT*x*	Copies the (*m*+1)th through the *n*th characters from the current buffer into buffer *x*, after clearing buffer *x*. Deletes all the characters copied from the current buffer.
B?	Lists all the active buffers and indicates the current buffer.
B?*x*	Displays the status of buffer *x*.

Table 3.13 Buffer commands

File Output Commands

The commands listed in Table 3.14 copy data from the current buffer to the output file. The CP does not move in any output command.

The colon modifier causes the P and PW commands to delete from the buffer all the characters that have been copied to the output file. The colon modifier causes the R commands to return a value of 1 if the input was successful and a 0 if the input failed.

When you issue File Output commands, a local file takes precedence over a global file — that is, the command applies to an open local file if one exists; otherwise, it applies to a global file.

Command	Action
BFB	Copies the current buffer and the remainder of the local input file to the local output file, clears the buffer, and closes both files. If Update Mode is on, creates a backup for the input file.
BFU	Copies the current buffer and the remainder of the local input file to the local output file, clears the buffer, and closes both files. If Update Mode is on, deletes the input file and renames the output file to the input filename.
E	Copies the buffer and the remainder of the input file to the output file.
FB	Copies the current buffer and the remainder of the global input file to the global output file, clears the buffer, and closes both files. If Update Mode is on, creates a backup for the input file.
FU	Copies the current buffer and the remainder of the global input file to the global output file, clears the buffer, and closes both files. If Update Mode is on, deletes the input file and renames the output file to the input filename.
P	Copies the current buffer to the output file, and appends a Form Feed character.
0P	Copies from the beginning of the current line to the CP into the output file, and appends a Form Feed character.
*n*P	Copies the next *n* lines, from the CP through *n* New-line characters, into the output file, and appends a Form Feed character.
-*n*P	Copies the preceding *n* lines and the characters between the beginning of the current line and the CP into the output file, and appends a Form Feed character.
m,*n*P	Copies the (*m* + 1)th through the *n*th characters from the current buffer into the output file, and appends a Form Feed character.
PW	Same as P, without appending a Form Feed character.
0PW	Same as 0P, without appending a Form Feed character.
*n*PW	Same as *n*P, without appending a Form Feed character.
-*n*PW	Same as -*n*P, without appending a Form Feed character.
m,*n*PW	Same as *m*,*n*P, without appending a Form Feed character.
R	Copies the current buffer to the output file and moves (yanks) a new page or window into the buffer.
*n*R	Repeats the R command *n* times.

Table 3.14 File Output commands

The Exit Command

You use the H command, described in Table 3.15, to exit from Speed.

Command	Action
H	Exits from Speed to the parent process. If the current buffer is not empty, or if any global files or the local files for the current buffer are still open, Speed displays the message: *Confirm?* Type Y if you wish to exit. Typing any other character causes Speed to continue.

Table 3.15 Exit command

Advanced Commands

This section describes the eight types of Speed commands that offer you advanced editing features.

Input Mode Commands

Three categories of command can change the way in which Speed reads data into the current buffer, displays data and commands, and reads data to the output file: Window Mode commands, Display Mode commands, and the Trace Mode command.

Window Mode Commands

Speed reads data from an input file to the current buffer either in page lengths (Form Feed to Form Feed) or in window lengths (a fixed number of lines). Initially, Speed is in Page Mode, and any file input command reads page lengths into the current buffer. The commands listed in Table 3.16 allow you to switch from one data input mode to the other.

Command	Action
WM	Returns the data input mode value. If WM=0, Speed is in Page Mode. If WM is equal to any other value, Speed is in Window Mode and the value is the number of lines per window.
*n*WM	Changes data input mode to Window Mode, with *n* specifying the number of lines in a window.
0WM	Changes the data input mode to Page Mode.

Table 3.16 Window Mode commands

Data read from an input file in Window Mode will include any Form Feed characters embedded in the file. In Page Mode, Speed remembers when the last character read by an input command was a Form Feed, but does not place the Form Feed into the current buffer. You determine the page size by inserting Form Feeds in text.

Display Mode Commands

Normally, Speed does not display text unless you issue the appropriate Type-Out command. However, if Display Mode is turned on, Speed will display a specified number of lines preceding and following the current CP location before it gives the next prompt (!). The current CP location is indicated on the display by a flashing asterisk.

The commands listed in Table 3.17 turn Display Mode on, return the status of Display Mode, or turn Display Mode off.

Command	Action
WD	Returns the current value of display mode. If WD=0, display mode is off. If WD is equal to any other value, Display Mode is on and Speed displays that number of lines preceding and following the CP.
*n*WD	Turns Display Mode on, and sets WD to the value specified by *n*. (If *n*>10, WD is set to 10.)
0WD	Turns Display Mode off.

Table 3.17 Display Mode Commands

Trace Mode Command

When trace mode is turned on (see Table 3.18), Speed displays the characters in the command string as they are executed. Specifically, each character, including New-line, Form Feed, space, arguments to a command, and the first letter of the command, is echoed. (For brevity, the rest of the characters in the command string, for example, a long insert, are not echoed.) Once the entire command has been processed, Speed resumes the process of echoing each character.

Command	Action
?	Turns Trace Mode on if it was off, and turns it off if it was on.

Table 3.18 Trace Mode command

Case Control Commands

The Case Control command and its variations let you create and edit uppercase and lowercase files from an uppercase terminal. The general form of the command is:

*[n]*WC*[x[y]]*

where

n may be 0 to deactivate case control, positive for up-shifting, or negative for down-shifting.

x is the shift character.

y is the shift lock character.

Table 3.19 lists the Case Control commands.

The unmodified case control commands affect characters as they are being evaluated in the command line, if they were entered from the keyboard. However, the colon modifier extends case control as follows.

The :*n*WC*x[y]* commands affect all of the characters in the command line, regardless of whether they were entered from the keyboard or expanded from character string insertion references (see Character String Insertion).

The :WC command returns a 1 if case control has been extended to include input from buffers and files, and returns a 0 otherwise.

Command	Action
*n*WC*x*$	Turns Case Control Mode on and designates *x* as the shift-up character when *n* is positive. Designates *x* to be the shift-down character when *n* is negative.
*n*WC*xy*	Same as *n*WC*x*$, except that *y* is the shift lock character. When *n* is positive, *y* shifts all the characters it precedes to uppercase until the next occurrence of *y* or the command line terminator (CTRL-D). When *n* is negative, *y* shifts all the characters it precedes to lowercase.
0WC	Deactivates case control.
WC	Displays the current status of case control when used as an argument to the = command.

Table 3.19 Case Control commands

Numeric Argument Commands

Two commands can set or change the interpretation of numeric arguments to other commands: the Default Argument command and the Alternate Radix command.

Default Argument

Table 3.20 lists the variations of the WA command that sets or returns the Default Argument value for D, J, K, L, and M commands. Initially, the Default Argument value is 0.

Command	Action
*n*WA	Sets the Default Argument to 1 if *n* is not equal to zero, and sets it to 0 if *n* is equal to 0.
WA	Returns the current Default Argument value when used as an argument to the = command.

Table 3.20 Numeric Argument commands

Alternate Radix

Table 3.21 lists the variations of the WR command that sets or returns the Alternate Radix value. The Standard Radix is always 10 (decimal), and the Alternate Radix is initially 8 (octal). The ampersand (&) modifier switches from decimal to the Alternate Radix for the single argument or command it precedes.

Command	Action
*n*WR	Sets the value of the Alternate Radix to *n* (where *n* can be any number from 2 through 36).
WR	Returns the value of the Alternate Radix when used as an argument to the = command.

Table 3.21 Alternate Radix commands

Numeric Variable Commands

Table 3.22 lists the commands that manipulate the numeric integer variables (named V0 to V9). Since each command returns a value, you can use the commands alone, in numeric expressions, or as numeric arguments to other commands.

Command	Action
VD*x*	Decrements the value of variable *x* by 1 and returns its new value.
VI*x*	Increments the value of variable *x* by 1 and returns its new value.
*n*VS*x*	Sets the value of variable *x* to *n* and returns its new value.
V*x*	Returns the current value of variable *x*.

Table 3.22 Numeric Variable commands

Iteration Commands

Speed will execute a command string any number of times when you place the command string between angle brackets. (The enclosed command string, together with the angle brackets, is known as a command loop.) Table 3.23 lists the two forms of the iteration command.

Command	Action
<command_string>	Repeats the command string indefinitely.
n<command_string>	Repeats the command string *n* times. If *n* is less than or equal to zero, skips the command string.

Table 3.23 Iteration commands

You can use the semicolon command to terminate a command loop prematurely (see Table 3.24). When a semicolon command terminates a command loop, control transfers to the command immediately following the command loop. The colon modifier reverses the action of the semicolon command. This is also shown in Table 3.24.

Command	Action
;	Terminates the command loop if the previous Search command was unsuccessful.
:;	Terminates the command loop if the previous Search command was successful.
n;	Terminates the command loop if *n* is less than or equal to zero.
n:;	Terminates the command loop if *n* is greater than zero.

Table 3.24 Command Loop terminators

Searches in Command Loops

Speed treats all Search commands (and the file input commands R, A, and Y) within command loops as if they were modified by the colon modifier. That is, the command returns a 1 if it succeeds and a 0 if it fails, suppressing error messages.

Even if a search fails, command execution within the command loop continues. Therefore, you should always write Search commands within command loops so that they act as a numeric argument to the next command. Alternatively, you can combine the search command with a semicolon (;) command. This will transfer control out of the loop if the preceding Search command failed. Note that the semicolon command (without an argument) applies to the preceding Search command even if other commands appear in the command string between the Search command and the semicolon.

Flow Control Commands

Flow Control commands enable you to perform unconditional branches to predefined labels, or conditional blocks based on the evaluation of certain conditions.

Unconditional Branching

You use the command string labels to name locations within a command string. A label is a character string delimited by a pair of exclamation marks (!*label*!), and it can appear anywhere in a command string except in text string arguments. Since Speed ignores labels unless they are specifically referenced by an Unconditional Branch command, you can also use them as comments in a command string.

The O command (see Table 3.25) is the Unconditional Branch command.

Command	Action
O*label*$	Transfers control to the command following !*label*! in the command string.

Table 3.25 Flow Control: Unconditional Branch command

You can use an Unconditional Branch command to exit from a command loop.

Conditional Skip

Table 3.26 lists the four forms of the conditional statement.

Command	Action
n''G*string*'	True if *n* is greater than zero.
n''L*string*'	True if *n* is less than zero.
n''E*string*'	True if *n* is equal to zero.
n''N*string*'	True if *n* is not equal to zero.

Table 3.26 Flow Control: Conditional Skip commands

If the condition is true, Speed executes the commands in the command string preceding the single quote (plus, of course, the commands following the single quote). If the condition is false, Speed scans but does not execute (in other words, *skips*) the commands preceding the single quote, and resumes execution at the command following the single quote.

You can nest conditional statements: that is, they can contain, or be contained within, a command loop. If you include a conditional statement within a nest of command loops, you must ensure that the whole of the conditional statement (from '' to ') is within one level of the nested loops.

For example:

```
!<SLDA$0"L1T>'
```

is unacceptable.

Program Execution Commands

The Program Execution commands listed in Table 3.27 allow you to execute CLI commands or your own programs from a Speed command line.

Command	Action
X*string*$	Executes *string* as a CLI command.*
:X*string*$	Executes the program specified by *string.**
:X$	Executes the CLI.

Table 3.27 Program Execution commands

**String must specify the complete program name, including the .PR extension, if relevant.*

Command Line Recall Command

The effect of the Command Line Recall command is summarized in Table 3.28.

Command	Action
_*x*	When used as the first command after the prompt, places the previous command line in buffer *x*. If there was no previous command string longer than 10 characters (the command line terminator, CTRL-D, being counted as two characters, $$), or if the last such command string had previously been saved with an _*x* command, then the last command string issued is placed in buffer *x*.

Table 3.28 Command Line Recall command

Command Modifiers

Command modifiers alter the normal interpretation of Speed commands. Table 3.29 summarizes their effects. A more detailed discussion appears with each relevant command in Chapter 4.

Command Modifier	Command Modified	Action
:	S, C, N, Q, A, Y, R	Returns a 1 if the command was successful, or a 0 if the command failed, and suppresses error messages.
:	P, PW	Deletes from the current buffer those characters that were copied to the output file.
:	T	Prints the output at the line printer instead of at the console.
:	;	Reverses the action of the command.
@	S, C, N, Q, I	Changes the command delimiter from $ (Escape) to any other character.
@	T	Types the character string from the command line.
@	=	Suppresses the New-line character from the display.
&	Any commands or numeric arguments	Switches from decimal to the alternate radix for the single argument or command it precedes.

Table 3.29 Command modifiers

Command Dictionary

4

The Command Dictionary gives a complete definition of all Speed commands. Each entry in the dictionary starts with the command mnemonic and the command name, followed by the formats for the command. The command name generally relates directly to the command mnemonic so that new users will be able to remember the mnemonics easily; for example, the N command is called the Nonstop Search command.

The entries are arranged in alphabetical order, with control and special characters appearing at the end of the dictionary.

A Append

A

This command reads one page or window from the input file and appends it to the end of the current buffer. The CP position remains unchanged. Speed prints an error message if it cannot execute the command.

Modifiers

The A command accepts the colon modifier, which inhibits error messages and returns a 1 if the command was successful and returns a 0 if the command failed. You can use the :A command as a numeric argument to the next command.

Activate Buffer *x*

BA*x*

BA*x*

This command activates buffer *x*, where buffer *x* is an existing buffer left from a previous editing session.

BC*x* Buffer Copy

BC*x*
0BC*x*
*n*BC*x*
-*n*BC*x*
m,*n*BC*x*

The BC*x* command copies the entire contents of the current buffer to buffer *x*, after clearing buffer *x*. The contents and CP position of the current buffer remain unchanged. The CP for buffer *x* is positioned before the first character in the buffer.

0BC*x* copies all the characters from the beginning of the current line to the current CP position into buffer *x*, after clearing buffer *x*.

*n*BC*x* copies the next *n* lines, starting at the current CP position, from the current buffer to buffer *x*, after clearing buffer *x*.

-*n*BC*x* copies the *n* lines preceding the current line, plus the characters on the current line up to the CP position, from the current buffer to buffer *x*, after clearing buffer *x*.

m,*n*BC*x* copies the (*m*+1)th through the *n*th characters from the current buffer to buffer *x*, after clearing buffer *x*.

Example

```
!5BCA$$
```

Copies the 5 lines following the CP from the current buffer into buffer A, after clearing buffer A.

File Backup (Local)

BFB

BFB

The BFB command copies the contents of the current buffer and the remainder of the local input file to the local output file, closes both files, and clears the current buffer. If local Update Mode is on, the input file, FILEIN, is renamed to FILEIN.BU, and the output file is renamed to FILEIN. This command clears the current buffer.

Example

If your input file was TEST, and your output file was TEST.TM, with local Update Mode on, a BFB command would have the following effect:

1. Copy the contents of the current buffer and the remainder of TEST into TEST.TM.
2. Close TEST and TEST.TM.
3. Rename TEST to TEST.BU.
4. Rename TEST.TM to TEST.

This leaves you with a closed input file, TEST.BU, and a closed output file, TEST.

Note

If a local input or output file has not been opened, the corresponding global file is used instead.

BFC File Close (Local)

BFC

This command closes any local input and output files. It does not write from the input file or the buffer before closing the files, and it does not clear the current buffer.

File New Read (Local)

BFNR

BFNR*pathname*$
BFNR$

The BFNR*pathname*$ command closes the current local input file and opens an existing file (specified by *pathname*) as the local input file. After you have issued this command, you must issue a file input command (Y or A) to read a page or window into the current buffer.

The BFNR$ command closes the current local input file without opening a new one.

Note

You cannot issue the BFNR*pathname*$ or the BFNR$ command if local Update Mode is on, that is, if the current local input file was opened with a BFO command.

BFNW File New Write (Local)

BFNW*pathname*$
BFNW$

The BFNW*pathname*$ command closes the current local output file, and creates and opens a new local output file (specified by *pathname*). It does not clear the current buffer.

The BFNW$ command closes the current local output file, but it does not open a new file for output.

Note

You cannot use the BFNW*pathname*$ or the BFNW$ commands if local Update Mode is on, that is, if the current local output file was opened by a BFO command.

The file specified by the BFNW*pathname*$ command cannot already exist.

File Open (Local)

BFO

BFO*pathname*$

This command opens an existing file (specified by *pathname*) as the local input file, moves (yanks) a page or window into the current buffer, and creates a new local output file. The local output file is given the same name as the input file, but with the extension .TM. The BFO command turns on Update Mode for the file.

Note

Since the BFO command turns local Update Mode on, you cannot issue BFNW or BFNR commands as long as the file is open.

Example

The following example illustrates the action taken by Speed when it executes a BFO command.

```
!BFOEXAMPLE$F?$$

Global:
   Input File   - None
   Output File     - None
Local:
   Input File   - PATHNAME:EXAMPLE
   Output File     - PATHNAME:EXAMPLE.TM
   Update Mode On
```

BFR File Read (Local)

BFR*pathname*$

This command opens an existing file (specified by *pathname*) as the local input file. You must issue a file input command (Y or A) to read a page or window from the file into the current buffer.

File Update (Local)

BFU

This command copies the contents of the current buffer and the remainder of the local input file into the local output file, closes both files, and clears the current buffer.

If Update Mode for the files is on, the command also deletes the input file and renames the output file to the name of the original input file.

If the local input file is TEST, the local output file is TEST.TM, and Update Mode is on, the BFU command will have the following effect:

1. Copy current buffer and the remainder of TEST into TEST.TM.
2. Close TEST and TEST.TM.
3. Delete TEST.
4. Rename TEST.TM to TEST.

This leaves a closed local output file called TEST, and leaves the current buffer cleared.

Example

BFW File Write (Local)

BFW*pathname*$

This command creates and opens a new file (specified by *pathname*) as the local output file. You must close this file before you can issue another BFW command.

Note

The file specified in this command cannot already exist.

Get Characters and Store Them in Buffer *x*

BG*x*

BG*x*
0BG*x*
*n*BG*x*

The BG*x* and 0BG*x* commands get a line of characters typed at the console and store it in buffer *x*, destroying the previous contents of buffer *x*. When this command is executed, Speed displays a question mark followed by a space as a prompt before reading the line.

*n*BG*x* (where $n>0$) gets *n* characters typed at the console and stores them in buffer *x*, destroying the previous contents of buffer *x*. A New-line or other data-sensitive character will terminate the input. When the command is executed, Speed displays a question mark followed by a space as a prompt before reading the characters.

Modifiers

The BG*x* commands accept the colon modifier. This inhibits Speed from displaying the prompt (?), and allows users to define their own prompt.

Note

When you use the *n*BG*x* command, if *n* is greater than the default record length (136), then 136 is used instead; 136 is also used when $n<0$.

BK*x* Kill Buffer *x*

BK*x*.

This command deactivates and deletes buffer *x*. Buffer *x* cannot be the current buffer.

Note

This command does not close any opened local files associated with the deleted buffer.

Swap to Buffer *x*

BS*x*

BS*x*

This command makes buffer *x* the current buffer.

Note

Speed saves the status of the CP position and of any open local files when changing the current edit buffer, and restores them for the new edit buffer if the new buffer was already active.

BT*x* Buffer Transfer

BT*x*
OBT*x*
*n*BT*x*
-*n*BT*x*
m,*n*BT*x*

The BT*x* command copies the entire contents of the current buffer to buffer *x*, after clearing buffer *x*. The characters copied are deleted from the current buffer, but the CP position for the current buffer remains unchanged. The CP for buffer *x* is positioned before the first character in the buffer.

OBT*x* copies all the characters from the beginning of the current line to the current CP position into buffer *x*, after clearing buffer *x*. This command deletes from the current buffer those characters that were copied.

*n*BT*x* copies the next *n* lines, starting at the current CP position, from the current buffer to buffer *x*, after clearing buffer *x*. This command deletes from the current buffer those characters that were copied.

-*n*BT*x* copies the *n* lines preceding the current line, plus the characters on the current line up to the CP position, from the current buffer to buffer *x*, after clearing buffer *x*. This command deletes from the current buffer those characters that were copied.

m,*n*BT*x* copies the (*m*+1)th through the *n*th characters from the current buffer to buffer *x*, after clearing buffer *x*. This command deletes from the current buffer those characters that were copied.

Example

```
!5BTA$$
```

Copies the 5 lines following the CP from the current buffer into buffer A, after clearing buffer A, and deletes those 5 lines from the current buffer.

Buffer Status

B?

B?$
B?*x*

The B?$ command lists all the active buffers and their length (in characters), indicating the current buffer with the symbol =>.

The B?*x* command displays the status of buffer *x*.

Example

B?$

might display

```
=>   BUFFER 0 - 225
     BUFFER 1 - 32
     BUFFER A - 1024
```

C Change

C*string1*$*string2*$
*n*C*string1*$*string2*$
-*n*C*string1*$*string2*$
0C*string1*$*string2*$
m,*n*C*string1*$*string2*$
C$*string2*$

The C*string1*$*string2*$ command searches from the current CP position to the end of the buffer for *string1*. If found, it deletes *string1* and inserts *string2*, positioning the CP after the last character in *string2*. If the search is unsuccessful, Speed prints an error message and positions the CP at the beginning of the buffer.

The *n*C*string1*$*string2*$ command is the same as C*string1*$*string2*$, except that the search is from the current CP position through the next *n* New-line characters. If the search is unsuccessful, Speed prints an error message and positions the CP at the beginning of the line following the last line searched (that is, at its original location + *n* lines).

The -*n*C*string1*$*string2*$ command is the same as C*string1*$*string2*$, except that the search starts at the beginning of the *n*th line preceding the current line and ends at the current CP position. If the search is unsuccessful, Speed prints an error message and it leaves the CP at its original location.

The 0C*string1*$*string2*$ command is the same as C*string1*$*string2*$, except that the search is from the beginning of the current line to the CP position. If the search is unsuccessful, Speed prints an error message and it leaves the CP at its original location.

The *m*,*n*C*string1*$*string2*$ command is the same as C*string1*$*string2*$, except that the search is from the (*m*+*1*)th through the *n*th characters in the buffer. If the search is unsuccessful, Speed prints an error message and positions the CP immediately after the *n*th character in the buffer.

If you omit *string1* from any of the previous commands (for example, use *n*C$*string2*$ instead of *n*C$*string1*$*string2*$), the command will search for the string that was searched for by the previous search command (C, S, N, or Q). If found, the string is deleted and replaced by *string2*. If there was no previous search, or if the previous string was longer than 31 characters, Speed will display the following warning message:

```
Error: Incomplete string in search buffer
```

and the search will continue, using the incomplete string (the first 31 characters of the previous string). Note that C$$ will delete the next occurrence of the previous string.

Modifiers

Each of the versions of the C command will accept either the @ modifier, or the : modifier, or a combination of the two (@: or :@).

The @ modifier has the effect of changing the command delimiter from **$** to the character immediately following the command mnemonic. For example, the command @C%*string1*%*string2*% is the same as the command C*string1*$*string2*$, except that the command delimiter is now %. The new delimiter is effective only for the current command.

The colon modifier has the effect of inhibiting error messages when a search is unsuccessful. The command will return a value of 1 if the search succeeds, and a value of 0 if the search fails. The command can be used as a numeric argument to the next command.

A combination of the two modifiers has the effects of both.

Notes

A detailed discussion of the use of templates with Search commands appears in Chapter 3.

The value of Case Match Mode determines whether or not the C commands will match characters regardless of case. See WS. The default value of Case Match Mode is such that matches are case independent (a will match A).

The Position Mode Value affects the positioning of the CP after an unsuccessful search. See WP. The default value of Position Mode is such that the CP position will be as described above, following an unsuccessful search.

D Delete

*n*D
-*n*D

The *n*D command deletes the *n* characters following the CP from the current buffer.

The -*n*D command deletes the *n* characters preceding the CP from the current buffer.

Note

When the default argument value is 0 (WA=0), the D command has no effect; but if the default argument value has been set to 1 (WA=1), the D command deletes one character from the current buffer. (See WA.)

End

E

This command copies the contents of the current buffer and the remainder of the input file (including any Form Feeds) to the output file. This command leaves the current buffer cleared.

FB File Backup

FB

The FB command copies the contents of the current buffer and the remainder of the global input file to the global output file, closes both files, and clears the current buffer. If global Update Mode is on, the input file, FILEIN, is renamed to FILEIN.BU, and the output file is renamed to FILEIN. This command clears the current buffer.

Example

If your input file was TEST, and your output file was TEST.TM, with global Update Mode on, an FB command would have the following effect:

1. Copy the contents of the current buffer and the remainder of TEST into TEST.TM.
2. Close TEST and TEST.TM.
3. Rename TEST to TEST.BU.
4. Rename TEST.TM to TEST.

This leaves you with a closed input file, TEST.BU, and a closed output file, TEST.

File New Read

FNR

FNR*pathname*$
FNR$

The FNR*pathname*$ command closes the current global input file and opens an existing file (specified by *pathname*) as the global input file. After you have issued this command, you must issue a file input command (Y or A) to read a page or window into the current buffer.

The FNR$ command closes the current global input file without opening a new one.

Note

You cannot issue the FNR*pathname*$ or the FNR$ command if global Update Mode is on, that is, if the current global input file was opened with an FO command.

FNW File New Write

FNW*pathname*$
FNW$

The FNW*pathname*$ command closes the current global output file, and creates and opens a new global output file (specified by *pathname*). It does not clear the current buffer.

The FNW$ command closes the current global output file, but it does not open a new file for output.

Notes

You cannot use the FNW*pathname*$ or the FNW$ commands if global Update Mode is on, that is, if the current global output file was opened by an FO command.

The file specified by the FNW*pathname*$ command cannot already exist.

File Open

FO

FO*pathname*$

This command opens an existing file (specified by *pathname*) as the global input file, moves (yanks) a page or window into the current buffer, and creates a new global output file. The global output file is given the same name as the input file, but with the extension .TM. The FO command turns Update Mode on for the file.

Note

Since the FO command turns global Update Mode on, you cannot issue FNW or FNR commands as long as the file is open.

Example

The following example illustrates the action taken by Speed when it executes an FO command.

```
!FOEXAMPLE$F?$$

GLOBAL:
   Input File   - PATHNAME:EXAMPLE
   Output File     - PATHNAME:EXAMPLE.TM
   Update Mode On
LOCAL:
   Input File   - None
   Output File     - None
```

FR File Read

FR*pathname*$

This command opens an existing file (specified by *pathname*) as the global input file. You must issue a file input command (Y or A) to read a page or window from the file into the current buffer.

File Update

FU

This command copies the contents of the current buffer and the remainder of the global input file into the global output file, closes both files and clears the current buffer.

If Update Mode for the files is on, the command also deletes the input file and renames the output file to the name of the original input file.

If the global input file is TEST, the global output file is TEST.TM, and Update Mode is on, the FU command will have the following effect:

1. Copy current buffer and the remainder of TEST into TEST.TM.
2. Close TEST and TEST.TM.
3. Delete TEST.
4. Rename TEST.TM to TEST.

This leaves a closed global output file called TEST, and leaves the current buffer cleared.

Example

FW File Write

FW*pathname*$

This command creates and opens a new file (specified by *pathname*) as the global output file. You must close this file before you can issue another FW command.

Note

The file specified in this command cannot already exist.

Filestatus

F?

The F? command lists the open global and local input and output files, and indicates whether Update Mode is on.

F?

Example

```
!F?$$

GLOBAL:
   Input File   - PATHNAME:EXAMPLE
   Output File     - PATHNAME:EXAMPLE.TM
   Update Mode On
LOCAL:
   Input File   - None
   Output File     - None
```

H Halt

H

With this command you can exit from Speed and return to the CLI. If any files are open, or if the current buffer is not empty, Speed prints the message:

Confirm?

Type Y if you wish to exit. Speed will then close all files before returning to the CLI.

Insert

I

I*string*$
n I

The I*string*$ command inserts the character string into the current buffer at the current CP location, and repositions the CP to follow the last character in the string.

The *n*I command inserts a single ASCII character into the current buffer at the current CP location; *n* is the decimal equivalent of the ASCII character inserted. This command is particularly useful for inserting characters that may not be available on the input terminal.

Modifiers

The I*string*$ command will accept the @ modifier which has the effect of changing the command delimiter from $ to the character immediately following the command mnemonic. For example, the command @I%*string*% is the same as the I*string*$ command, except that the command delimiter is now %. The new delimiter is effective only for the current command.

Examples

```
!10I$$
```

This command inserts a New-line character at the current CP location. (10 is the decimal equivalent of the ASCII New-line character.)

J CP Jump

J
*n*J

The J, 0J, and 1J commands cause the CP to jump to the beginning of the first line in the current buffer.

The *n*J command causes the CP to jump to the beginning of the *n*th line in the current buffer (*n* is always relative to the beginning of the current buffer).

Note

The J command always causes the CP to jump to the beginning of the first line in the current buffer, regardless of whether the default argument value (WA) is 0 or 1, since 0J and 1J are equivalent. (See WA.)

Kill

K

K

K
OK
*n*K
-*n*K
m,*n*K
#K

The K and OK commands delete all the characters between the beginning of the current line and the CP location.

The *n*K command deletes lines in the current buffer from the CP position up to and including the next *n* New-line characters.

The -*n*K command deletes the *n* lines preceding the current line, plus the characters from the beginning of the current line up to the CP location.

The *m*,*n*K command deletes from the (*m*+1)th through the *n*th character in the current buffer, and positions the CP immediately after the *m*th character.

The #K command deletes the entire contents of the current buffer.

Note

When the default argument value is 0 (WA=0), the K command deletes all the characters between the beginning of the current line and the CP location. If the default argument value has been set to 1 (WA=1), the K command will delete all the characters from the CP to the end of the current line. (See WA.)

L CP Line Move

L
0L
*n*L
-*n*L

The L and 0L commands move the CP to the beginning of the current line.

The *n*L command moves the CP forward across *n* New-line characters and places it at the beginning of the line following the *n*th New-line character.

The -*n*L command moves the CP backward to the beginning of the *n*th line preceding the current line.

Note

When the default argument value is 0 (WA=0), the L command moves the CP to the beginning of the current line. If the default argument value has been set to 1 (WA=1), the L command moves the CP to the beginning of the next line. (See WA.)

CP Move

M

*n*M
-*n*M

The *n*M command moves the CP from its current location *n* characters to the right (forward).

The -*n*M command moves the CP from its current location *n* characters to the left (backward).

Note

When the default argument value is 0 (WA=0), the M command has no effect; but if the default argument value has been set to 1 (WA=1), the M command moves the CP one character forward. (See WA.)

N Nonstop Search

N*string*$
N$

The N*string*$ command searches from the current CP location to the end of the buffer for the character string. If Speed does not find the character string in the current buffer, it executes an R command (moves the contents of the buffer to the output file, clears the buffer, and yanks another page or window from the input file into the buffer) and continues the search. Speed will continue the search in this manner until the character string is found or until it reaches the end of the input file. If the search is successful, Speed positions the CP after the last character in the string. If the search is unsuccessful, Speed displays an error message.

The N$ form searches for the character string that was searched for by the previous search command (C, S, N, or Q). If there was no previous search, or if the previous string was longer than 31 characters, Speed will display the following warning message:

```
Error: Incomplete string in search buffer
```

and the search will continue, using the incomplete string (the first 31 characters of the previous string).

Modifiers

The N commands will accept either the @ modifier, or the : modifier, or a combination of the two (@: or :@).

The @ modifier has the effect of changing the command delimiter from $ to the character immediately following the command mnemonic. For example, the command @N%*string*% is the same as the command N*string*$, except that the command delimiter is now %. The new delimiter is effective only for the current command.

The colon modifier has the effect of inhibiting error messages when a command is unsuccessful. The command will return a value of 1 if the search succeeds, and a value of 0 if the search fails. The command can be used as a numeric argument to the next command.

A combination of the two modifiers has the effects of both.

Notes

A detailed discussion of the use of templates with Search commands appears in Chapter 3.

The value of Case Match Mode determines whether or not the N commands will match characters regardless of case. See WS. The default value of Case Match Mode is such that matches are case independent (a will match A).

Over (Unconditional Branch)

O

O*label*$

This command unconditionally transfers control to !*label*! in the command string, and continues processing the command string from the first command immediately following !*label*!.

The command string label (!*label*!) may appear anywhere in a command string, except in text string arguments or in a command loop. *Label* must contain fewer than 32 characters, and any ASCII characters can be used except Escape (the string delimiter).

P Put Buffer into Output File

P
OP
*n*P
-*n*P
m,*n*P

The P command copies the entire contents of the current buffer to the output file, with an appended Form Feed character.

The OP command copies characters on the current line, from the beginning of the line to the CP location, from the buffer to the output file, with an appended Form Feed character.

The *n*P command copies lines from the current buffer, from the CP location up to and including the next *n* New-line characters, to the output file, with an appended Form Feed character.

The -*n*P command copies the *n* lines preceding the current line, plus the characters from the beginning of the current line up to the CP position, from the current buffer to the output file, with an appended Form Feed character.

The *m*,*n*P command copies the (*m*+1)th to the *n*th characters inclusive from the current buffer to the output file, with an appended Form Feed character.

Modifiers

All of the P commands accept the colon modifier. The modified commands delete from the current buffer those characters that have been copied to the output file.

Put Buffer into Output File Without Form Feed

PW

PW
0PW
*n*PW
-*n*PW
m,*n*PW

The PW command copies the entire contents of the current buffer to the output file, without an appended Form Feed character.

The 0PW command copies characters on the current line, from the beginning of the line to the CP location, from the buffer to the output file, without an appended Form Feed character.

The *n*PW command copies lines from the current buffer, from the CP location up to and including the next *n* New-line characters, to the output file, without an appended Form Feed character.

The -*n*PW command copies the *n* lines preceding the current line, plus the characters from the beginning of the current line up to the CP position, from the current buffer to the output file, without an appended Form Feed character.

The *m*,*n*PW command copies the $(m+1)$th to the *n*th characters inclusive from the current buffer to the output file, without an appended Form Feed character.

Modifiers

All the PW commands accept the colon modifier. The modified commands delete from the current buffer those characters that have been copied to the output file.

Q Quest Search

Q*string*$
Q$

The Q*string*$ command searches from the current CP location to the end of the buffer for the character string. If Speed does not find the character string in the current buffer, it executes a Y command (yanks another page or window from the input file into the buffer, overwriting the previous contents of the buffer) and continues the search. Speed will continue the search in this manner until the character string is found or until it reaches the end of the input file. This command does not transfer any data to the output file. If the search is successful, Speed positions the CP after the last character in the string. If the search is unsuccessful, Speed displays an error message.

The Q$ command searches for the character string that was searched for by the previous search command (C, S, N, or Q). If there was no previous search, or if the previous string was longer than 31 characters, Speed will display the following warning message:

```
Error: Incomplete string in search buffer
```

and the search will continue, using the incomplete string (the first 31 characters of the previous string).

Modifiers

The Q commands will accept either the @ modifier, or the : modifier, or a combination of the two (@: or :@).

The @ modifier has the effect of changing the command delimiter from $ to the character immediately following the command mnemonic. For example, the command @Q%*string*% is the same as the command Q*string*$, except that the command delimiter is now %. The new delimiter is effective only for the current command.

The colon modifier has the effect of inhibiting error messages when a command is unsuccessful. The command will return a value of 1 if the search succeeds, and a value of 0 if the search fails. The command can be used as a numeric argument to the next command.

A combination of the two modifiers has the effects of both.

Notes

If you issue a Q command when global and local Update Modes are off and Q is not in a command loop, Speed displays the following message:

```
Confirm (Q-command) ?
```

If you really meant to use the Q command, you should type Y followed by New-line. Speed will ignore the command if you type any other character.

A detailed discussion of the use of templates with Search commands appears in Chapter 3.

The value of Case Match Mode determines whether or not the Q commands will match characters regardless of case. See WS. The default value of Case Match Mode is such that matches are case independent (a will match A).

R Read Out and Read In a Page

R
*n*R

The R command copies the contents of the current buffer to the output file, clears the buffer, and reads a page or window from the input file into the current buffer. This command is equivalent to a combination of the P (or PW) and Y commands.

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

Modifiers

The R commands accept the colon modifier. Modified commands return a value of 1 if input is successful and a value of 0 if it fails. You can use these values as numeric arguments to the next command. No error message appears. Command execution continues whether or not the command succeeds.

Search

S

*S**string**$*
*n**S**string**$*
*-n**S**string**$*
*w**S**string**$*
*m,n**S**string**$*
S$

The *S**string**$* command searches from the current CP position to the end of the buffer for *string*. If found, it positions the CP after the last character in *string*. If the search is unsuccessful, Speed prints an error message and positions the CP at the beginning of the buffer.

The *nS**string**$* command is the same as *S**string**$*, except that the search is from the current CP position through the next *n* New-line characters. If the search is unsuccessful, Speed prints an error message and positions the CP at the beginning of the line following the last line searched (that is, at its original location + *n* lines).

The *-nS**string**$* command is the same as *S**string**$*, except that the search starts at the beginning of the *n*th line preceding the current line and ends at the current CP position. If the search is unsuccessful, Speed prints an error message and it leaves the CP at its original location.

The 0*S**string**$* command is the same as *S**string**$*, except that the search is from the beginning of the current line to the CP position. If the search is unsuccessful, Speed prints an error message and it leaves the CP at its original location.

The *m,nS**string**$* command is the same as *S**string**$*, except that the search is from the (*m*+1)th through the *n*th characters in the buffer. If the search is unsuccessful, Speed prints an error message and positions the CP immediately after the *n*th character in the buffer.

If you do not specify a search string with any of the above commands (for example, you use *n*S$ instead of *nS**string**$*), the command will search for the character string that was searched for by the previous search command (C, S, N, or Q). If there was no previous search, or if the previous string was longer than 31 characters, Speed will display the following warning message:

```
Error: Incomplete string in search buffer
```

and the search will continue, using the incomplete string (the first 31 characters of the previous string).

Modifiers

Each of the versions of the S command will accept either the @ modifier, or the : modifier, or a combination of the two (@: or :@). The @ modifier has the effect of changing the command delimiter from $ to the character immediately following the command mnemonic. For example, the command @S%*string*% is the same as the command S*string*$, except that the command delimiter is now %. The new delimiter is effective only for the current command.

The colon modifier has the effect of inhibiting error messages when a search is unsuccessful. The command will return a value of 1 if the search succeeds, and a value of 0 if the search fails. The command can be used as a numeric argument to the next command.

A combination of the two modifiers has the effects of both.

Notes

A detailed discussion of the use of templates with Search commands appears in Chapter 3.

The value of Case Match Mode determines whether or not the S commands will match characters regardless of case or not. See WS. The default value of Case Match Mode is such that matches are case independent (a will match A).

The Position Mode value affects the positioning of the CP after an unsuccessful search. See WP. The default value of Position Mode is such that the CP position will be as described above following an unsuccessful search.

Type Out

T

T
0T
*n*T
-*n*T
m,*n*T
#T
@T%*string*%

The T command types the current line, displaying the CP location with a flashing asterisk. (A circumflex (^) shows the CP location on consoles other than models 605*x*.)

The 0T command types the current line from the beginning to the location of the CP.

The *n*T command types the contents of the current buffer from the current CP location through the next *n* New-line characters.

The -*n*T command types the contents of the *n* lines preceding the current line, plus the contents of the current line up to the CP location.

The *m*,*n*T command types the contents of the current buffer from the (*m*+1)th character up to and including the *n*th character.

The #T command types the entire contents of the current buffer.

The @T%*string*% command types out the character string. The first character after the T (arbitrarily shown here as %) delimits *string*.

Modifiers

The Type Out commands all accept the colon modifier which causes the output to be typed at the line printer instead of at the console.

Note

The T commands do not move the CP.

VD*x* Variable Decrement

VD*x*

This command decrements the value of variable *x* by 1, and returns its new value.

Note

Variable *x* is one of the ten number integer variables named V0 through V9.

Variable Increment

VIx

This command increments the value of variable *x* by 1, and returns its new value.

Variable *x* is one of the ten numeric integer variables named VO through V9.

VIx

Note

VS*x* Variable Set

*n*VS*x*

This command sets the value of variable *x* to *n*, and returns its new value.

Notes

Variable *x* is one of the ten numeric integer variables named VO through V9.

You cannot use this command with arithmetic operators within a numeric expression. For example,

```
9VS0 + 3/4
```

is a legal expression, whereas

```
9 + VS0 + 3/4
```

is not.

Variable Value

Vx

This command returns the value of variable *x*. (The numeric variables are named V0 through V9, so *x* may be any digit, 0 through 9.)

WA Default Argument Mode

WA
*n*WA

The Default Argument Mode command sets the Default Argument value for the Delete (D), Jump (J), Kill (K), CP Line Move (L), and CP Move (M) commands. Initially, WA is set to zero.

The *n*WA command sets WA to 1 if *n* is not equal to zero, and sets WA to 0 if *n* is equal to zero.

When WA is used as an argument to the = command, Speed returns the current default argument value (1 or 0).

If you omit the numeric argument *n* from the *n*D, *n*J, *n*K, *n*L, or *n*M commands, the value of the default argument is used instead. You can also use WA as the numeric argument to any of these commands. For example, if the default argument value has been set to 1, the WAD command is equivalent to 1D.

Case Control Mode

WC

WC

0WC
*n*WC*x*$
*n*WC*xy*

Case Control commands enable you to create and edit uppercase and lowercase files from an uppercase terminal.

The 0WC command turns case control off. Characters are read from the terminal exactly as typed, with no translation from uppercase to lowercase.

When *n* is positive, the *n*WC*x* command designates *x* to be the shift-up character. Any character that is preceded by an *x* will be interpreted as uppercase. All other characters are interpreted as lowercase.

When *n* is negative, the *n*WC*x* command designates *x* to be the shift-down character. Any character that is preceded by an *x* will be interpreted as lowercase. All other characters are interpreted as uppercase.

The *n*WC*xy* command has the same effect as the *n*WC*x* command, except that *y* is designated as the shift-lock character that shifts all characters it precedes to uppercase (if *n* is positive) or lowercase (if *n* is negative), until the next appearance of *y* or the command terminator (CTRL-D).

When you use WC as an argument to the = command, Speed returns the current status of Case Control Mode, as shown in the table.

WC	Means
0	Case control off.
1	Up-shifting (shift characters precede uppercase letters).
-1 (65535)	Down-shifting (shift characters precede lowercase letters).

Modifiers

Normally, case control affects only those characters in the command string which were typed from the keyboard. The colon modifier extends case control to the entire command string. This means that characters included through the use of ↑B*x* and ↑F*pathname*$ are also subject to case control. Speed converts these character sequences to uppercase, and then interprets the letters as uppercase or lowercase in the same way as with the nonmodified *n*WC*x* and *n*WC*xy* commands.

The :WC command returns a value of 1 if case control has been extended to characters included through the use of ↑B*x* and ↑F*pathname*$. Otherwise, it returns a value of 0.

Notes

You can use WC as a numeric argument to another command in a command string. Speed replaces WC with 0, 1, or -1, according to whether case mode is deactivated, up-shifting, or down-shifting.

Case Control Mode can be set and used in the same command string. Case changes will take effect for the command string containing the Case Control command once the command has been executed.

Once case control is active, it affects everything you type in, including WC commands. (However, case control only applies to alphabetic characters; that is, if you type a shift character preceding a nonalphabetic character, shifting has no effect except as noted below.)

If you want to enter the characters you have chosen to be the shift character (such as #) or the shift lock character (such as "), you must precede them by the shift character (that is, you would type ## or #'').

Display Mode

WD
*n*WD
0WD

The Display Mode commands set the current Display Mode value. If the Display Mode value is set to zero, Display Mode is off. If the Display Mode value is set to any other value, Display Mode is on and Speed tries to display that number of lines of text on either side of the CP before each prompt. For example, if the Display Mode value is set to 7, Speed tries to display the 7 lines preceding and the 7 lines following the CP, showing the CP location with a flashing asterisk. The Display Mode value may be no larger than 10.

The *n*WD command turns Display Mode on, and sets WD to the value specified by *n*. If *n*>10, the value of WD is set to 10.

The 0WD command turns Display Mode off.

If you use WD as an argument to the = command, Speed returns the current Display Mode value.

WD

Notes

Even if Display Mode is on, Speed will not display text if the previous command string generated output or if the previous command was an Execute (X) command.

When Speed is executed without the global /D switch, WD=0 (that is, Display Mode is off). The /D switch turns Display Mode on, setting WD=10.

WM Window Mode

WM
*n*WM
0WM

The Window Mode commands set the current Data Input Mode value. If the Data Input Mode value equals zero, Speed is in Page Mode; that is, text is moved to and from the current buffer in pages (from Form Feed to Form Feed*). If the Data Input Mode value is equal to any other value (say *n*), Speed is in Window Mode; that is, text is moved to and from the current buffer in windows (where *n* is the number of lines per window).

The *n*WM command changes the input mode to Window Mode, with *n* specifying the number of lines per window.

The 0WM command changes the input mode to Page Mode.

If you use WM as an argument to the = command, Speed returns the current Data Input Mode value.

**The text between Form Feeds is read into the buffer, but the end Form Feed is not. Its place is remembered by Speed, so that when the page is written out, the Form Feed is put back. The beginning Form Feed is part of the previous page.*

Position Mode

WP
*n*WP
0WP

Position Mode affects the positioning of the CP after an unsuccessful search that has been confined to a single page of the file (C or S). The Position Mode commands set the Position Mode value. If WP=0, the CP is positioned in its normal location after an unsuccessful search (see the table below). If WP=*n* (where *n* is not zero), the CP location after an unsuccessful search is as shown in the table.

Search Command Format	CP Position if WP=0	CP Position if WP<>0
S	Beginning of buffer.	Position before search.
*n*S	Position before search + *n* lines.	Position before search.
-*n*S	Position before search.	Position before search -*n* lines.
m,*n*S	After the *n*th character in the buffer.	After the *m*th character in the buffer

The *n*WP command sets the Position Mode value to *n* (WP=*n*).

The 0WP command sets the Position Mode value to zero (WP=0).

If you use WP as an argument to the = command, Speed returns the current Position Mode value.

WP=0 is the normal (default) mode.

When WP<>0, the CP is always positioned before the character where the search actually started.

WP

Notes

WR Alternate Radix

WR
*n*WR

The *n*WR command sets the value of the Alternate Radix to *n*. For example, if *n*=2 the Alternate Radix is binary; *n* may be any number between 2 and 36.

If you use WR as an argument to the = command, Speed returns the current Alternate Radix value.

Note

The Standard Radix is always 10 (decimal), and the Alternate Radix is initially set to 8 (octal).

The ampersand modifier switches from decimal to the Alternate Radix for the duration of a single Speed argument or command. See the section about modifiers in Chapter 3.

Search Case Match Mode

WS

WS

*n*WS

0WS

Case Match Mode affects whether Search commands (C, N, Q, and S) will match letters regardless of their case (uppercase or lowercase). The Case Match Mode commands set the value of Case Match Mode. If WS=0, matches will be case independent (that is, a will match A). If WS<>0, matches will be case dependent (that is, a will only match a).

The *n*WS command sets WS=*n*; that is, matches will be case dependent.

The 0WS command sets WS=0; that is, matches will be case independent.

If you use WS as an argument to the = command, Speed will return the current value of Case Match Mode.

The initial (default) mode is WS=0 (case independent).

Note

X Execute

X*string*$
:X$
:X*program*.PR$

The X*string*$ command executes *string* as a CLI command.

The :X$ command executes the CLI.

The :X*program*.PR$ command is a quick way of executing a program from Speed when the program does not require switches or arguments from a command line. Control returns to Speed once the command or program has completed execution.

Example

```
!XTIME$$
23:31:12
!
```

This command calls the CLI to execute the TIME command.

Yank

Y

The Y command clears the current buffer, reads a page or window from the input file into the current buffer, and places the CP at the beginning of the buffer.

The Y command accepts the colon modifier which inhibits error messages and returns a value of 1 if the command was successful and a value of 0 if the command was unsuccessful. You can use the :Y command as a numeric argument to the next command.

If you issue a Y command when global or local Update Mode is on, the buffer is not empty, and the Y command is not in a command loop, Speed returns the following message:

```
Confirm (Y-command) ?
```

Type Y followed by New-line if you wish to use the command. If you type any other character, Speed ignores the command.

Y

Modifiers

Note

CTRL-B*x* Insert Buffer *x*

↑B*x*

This command inserts the contents of buffer *x* into the command string in place of the ↑B*x*. You may nest the command up to nine levels, but remember that Insert File (↑F) commands also count as part of the nesting limit.

Notes

If you wish to insert a CTRL-B into a command line, type CTRL-B CTRL-B or CTRL-F CTRL-B. Speed will evaluate this as a single CTRL-B and will not treat it as part of a reference to a buffer.

Buffer *x* may contain text to be inserted, commands to be executed, or both, but it cannot be the current buffer.

Insert File

↑F*pathname*$

This command inserts the file specified by *pathname* into the command string in place of the command. You may nest the command up to nine levels, but remember that Insert Buffer (↑B*x*) commands also count as part of the nesting limit.

If you wish to insert a CTRL-F into a command line, type CTRL-B CTRL-F or CTRL-F CTRL-F. Speed will evaluate this as a single CTRL-F and will not treat it as a reference to a file.

The file you wish to insert may contain text, commands to be executed, or both, but it must not be greater than 65,535 bytes in length.

CTRL-F

Notes

CTRL-I Insert String with Tab

↑I*string*$

This command inserts a tab character followed by *string* into the current buffer at the CP location, and repositions the CP after the last character of the inserted *string*.

Modifiers

The CTRL-I command accepts the @ modifier. This has the effect of changing the command delimiter from $ to a tab. For example, the command @↑I*string*↑I is the same as the command ↑I*string*, except that the command delimiter is now Tab.

Insert String of ASCII Digits

\

n\

The *n*\ command inserts a string of ASCII digits having the value of the decimal number *n* at the current CP location, suppressing any leading zeros.

Modifier

This command accepts the & modifier which causes *n* to be interpreted in the Alternate Radix.

&*n*\ affects the interpretation of *n*. *n*&\ affects the way *n* is inserted. &*n*&\ affects both the interpretation of *n* and the way it is inserted.

Examples

-1\

This command will insert 65535 into the buffer at the current CP location. Note that 65535 is the way that Speed displays negative one.

n''x Conditional Execution

n''*x*string'

In this format *n* is a numeric argument, *x* is one of the conditions specified below, and *string* is a command string.

If the condition is true, the commands in the command *string* preceding the single quote are executed (plus, of course, the commands following the single quote). If the condition is false, the commands preceding the single quote are skipped, and control passes to the command following the single quote.

Specific Formats

n''G*string*'
n''L*string*'
n''E*string*'
n''N*string*'

The action taken depends on the condition specifier as shown in the table.

Condition Specifier	Action
G	True if *n* is greater than 0.
L	True if *n* is less than 0.
E	True if *n* equals 0.
N	True if *n* is not equal to 0.

Save Command Line in Buffer *x*

_ x

This command places the previous command line (if it was larger than 10 characters) in buffer *x*, freeing the space occupied by the string for use by subsequent command strings. If there was no previous command string longer than 10 characters (the command string terminator, CTRL-D, being counted as two characters, $$), or if the last such command string had previously been saved with a _*x* command, then the last command string issued is placed in buffer *x*.

Notes

The _*x* command may not be executed from a buffer or from a file.

The _*x* command must be the first command after the prompt.

If memory space becomes scarce, Speed will save the space occupied by the command string waiting to be stored, in order to avoid the error

Memory Space Exhausted

When this happens, the effect is the same as if there were no previous commands.

= Equals

n=

This command types out the value of the numeric argument *n*, where *n* may be any numeric argument or expression.

When a mode command mnemonic (WA, WC, WD, WM, WP, WR, WS) is used as an argument to the = command, Speed returns the current mode value. For example, WD= returns the current value of Display Mode.

Modifiers

The @ modifier suppresses the New-line character from the display. The colon modifier sends the output to the line printer instead of the console. A combination of the two modifiers has the effects of both.

The & modifier causes *n* to be interpreted or displayed in the alternate radix.

&*n*= affects the interpretation of *n*. *n*&= affects the way *n* is displayed. &*n*&= affects both the interpretation and the display of *n*.

Examples

```
!8*8=$$
64
!
```

This is the normal display of the *n*= command.

```
!8*8@=$$
64!
```

This is the effect of the @ modifier.

Command Loop

<>

<command string>
n<command string>

Speed will execute a command string any number of times when you place the command string between angle brackets and specify the number of iterations. A command string enclosed by angle brackets is known as a command loop.

The command loop *<command string>* repeats *command string* until a command within the loop terminates the loop.

The command loop *n<command string>* repeats *command string n* times when *n* is greater than zero, and it will skip the command string if *n* is less than or equal to zero.

Notes

Speed treats all search commands (S, C, N, and Q) and all file input commands (Y, R, and A) within command loops as if they were modified by a colon. That is, the command returns a value of 1 if the command is successful, returns a value of 0 if it fails, and inhibits error messages.

Even if a search command fails, command execution within the command loop continues. Therefore, you should always write Search commands within command loops so that they provide a numeric argument to the next command, or combine them with the semicolon command, which will terminate the loop.

; Conditionally Terminate Command Loop or Command Line Execution

;
n;

A semicolon (;) may appear anywhere in a command line except as the first character. When the semicolon appears in a command loop (angle brackets), the loop may be terminated and the next lower level of the command line will be executed. When the semicolon appears outside a command loop, the command line may be terminated. Termination depends on the specific form of the semicolon command, as shown below.

The **;** form of the semicolon command terminates the command loop (or command line) if the last Search command (C, S, N, or Q) was unsuccessful.

The *n*; form of the semicolon command terminates the command loop (or command line) if *n* is less than or equal to zero.

Modifiers

The colon modifier reverses the effect of the semicolon command, as shown below.

The **:;** command terminates the command loop (or command line) if the last search command was successful.

The *n*:; command terminates the command loop (or command line) if *n* is greater than zero.

Note

Speed treats an unsuccessful search outside of a command loop as an error condition, regardless of whether the Search command was followed by a semicolon.

Trace Mode Toggle

?

The **?** command turns Trace Mode on if it was off, and turns it off if it was on. When Trace Mode is on, Speed displays the characters in the command line as they are executed. Specifically, each character including New-line, Form Feed, space, arguments to the command, and the first letter of the command is echoed. (For brevity, the rest of the characters in the command string, for example, a long insert, are not echoed.) Once the entire command has been processed, Speed resumes the process of echoing each character.

ASCII Character Set

A

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)

DG-05495

Speed Error Messages

B

The following list gives the error messages which Speed can return. The first column gives the number of the error message; the second column indicates whether the message is an error (E) or a warning (W); and the third column lists the messages.

04401 (E)	*Illegal filename*
04402 (E)	*Syntax error*
04403 (E)	*Illegal variable name*
04404 (E)	*Illegal number of arguments to command*
04405 (E)	*Illegal buffer name*
04406 (E)	*Buffer is inactive*
04407 (E)	*Maximum iteration level exceeded*
04410 (E)	*No open file*
04411 (E)	*File already exists*
04412 (E)	*File does not exist*
04413 (E)	*File already open*
04414 (E)	*No more characters in input file*
04415 (E)	*Unsuccessful search*
04416 (E)	*Maximum insert depth exceeded*
04417 (E)	*Search string <> or broken over two levels*
04420 (E)	*Insert string too long*

04421 (E) *No more channels available*
04422 (W) *Input line too long*
04423 (E) *Attempt to delete current buffer*
04424 (E) *Parity error*
04425 (E) *Stack overflow*
04426 (E) *Memory space exhausted*
04427 (E) *Attempt to execute current buffer*
04430 (E) *Unterminated string*
04431 (E) *< with no corresponding >*
04432 (E) *with no corresponding '*
04433 (E) *Label not found*
04434 (E) *Unable to open $LPT*
04435 (E) *String argument too long*
04436 (E) *First argument greater than second argument*
04437 (E) *↑\ with no corresponding ↑*
04440 (E) *Renaming error*
04441 (E) *Illegal command*
04442 (E) *Illegal argument to command*
04443 (E) *Illegal control character in search string*
04444 (E) *File read protected*
04445 (E) *File write protected*
04446 (E) *Update mode on*
04447 (E) *Filename too long*
04450 (W) *Incomplete string in search buffer*
04453 (E) *Directory specifier unknown*

Index

J

K

L

M

N

O

P

R

S

T

U

V

W

Y

DG OFFICES

NORTH AMERICAN OFFICES

Alabama: Birmingham
Arizona: Phoenix, Tucson
Arkansas: Little Rock
California: Anaheim, El Segundo, Fresno, Los Angeles, Oakland, Palo Alto, Riverside, Sacramento, San Diego, San Francisco, Santa Barbara, Sunnyvale, Van Nuys
Colorado: Colorado Springs, Denver
Connecticut: North Branford, Norwalk
Florida: Ft. Lauderdale, Orlando, Tampa
Georgia: Norcross
Idaho: Boise
Iowa: Bettendorf, Des Moines
Illinois: Arlington Heights, Champaign, Chicago, Peoria, Rockford
Indiana: Indianapolis
Kentucky: Louisville
Louisiana: Baton Rouge, Metairie
Maine: Portland, Westbrook
Maryland: Baltimore
Massachusetts: Cambridge, Framingham, Southboro, Waltham, Wellesley, Westboro, West Springfield, Worcester
Michigan: Grand Rapids, Southfield
Minnesota: Richfield
Missouri: Creve Coeur, Kansas City
Mississippi: Jackson
Montana: Billings
Nebraska: Omaha
Nevada: Reno
New Hampshire: Bedford, Portsmouth
New Jersey: Cherry Hill, Somerset, Wayne
New Mexico: Albuquerque
New York: Buffalo, Lake Success, Latham, Liverpool, Melville, New York City, Rochester, White Plains
North Carolina: Charlotte, Greensboro, Greenville, Raleigh, Research Triangle Park
Ohio: Brooklyn Heights, Cincinnati, Columbus, Dayton
Oklahoma: Oklahoma City, Tulsa
Oregon: Lake Oswego
Pennsylvania: Blue Bell, Lancaster, Philadelphia, Pittsburgh
Rhode Island: Providence
South Carolina: Columbia
Tennessee: Knoxville, Memphis, Nashville
Texas: Austin, Dallas, El Paso, Ft. Worth, Houston, San Antonio
Utah: Salt Lake City
Virginia: McLean, Norfolk, Richmond, Salem
Washington: Bellevue, Richland, Spokane
West Virginia: Charleston
Wisconsin: Brookfield, Grand Chute, Madison

DG-04976

INTERNATIONAL OFFICES

Argentina: Buenos Aires
Australia: Adelaide, Brisbane, Hobart, Melbourne, Newcastle, Perth, Sydney
Austria: Vienna
Belgium: Brussels
Bolivia: La Paz
Brazil: Sao Paulo
Canada: Calgary, Edmonton, Montreal, Ottawa, Quebec, Toronto, Vancouver, Winnipeg
Chile: Santiago
Columbia: Bogata
Costa Rica: San Jose
Denmark: Copenhagen
Ecuador: Quito
Egypt: Cairo
Finland: Helsinki
France: Le Plessis-Robinson, Lille, Lyon, Nantes, Paris, Saint Denis, Strasbourg
Guatemala: Guatemala City
Hong Kong
India: Bombay
Indonesia: Jakarta, Pusat
Ireland: Dublin
Israel: Tel Aviv
Italy: Bologna, Florence, Milan, Padua, Rome, Tourin
Japan: Fukuoka, Hiroshima, Nagoya, Osaka, Tokyo, Tsukuba
Jordan: Amman
Korea: Seoul
Kuwait: Kuwait
Lebanon: Beirut
Malaysia: Kuala Lumpur
Mexico: Mexico City, Monterrey
Morocco: Casablanca
The Netherlands: Amsterdam, Rijswijk
New Zealand: Auckland, Wellington
Nicaragua: Managua
Nigeria: Ibadan, Lagos
Norway: Oslo
Paraguay: Asuncion
Peru: Lima
Philippine Islands: Manila
Portugal: Lisbon
Puerto Rico: Hato Rey
Saudi Arabia: Jeddah, Riyadh
Singapore
South Africa: Cape Town, Durban, Johannesburg, Pretoria
Spain: Barcelona, Bibao, Madrid
Sweden: Gothenburg, Malmo, Stockholm
Switzerland: Lausanne, Zurich
Taiwan: Taipei
Thailand: Bangkok
Turkey: Ankara
United Kingdom: Birmingham, Bristol, Glasgow, Hounslow, London, Manchester
Uruguay: Montevideo
USSR: Espoo
Venezuela: Maracaibo
West Germany: Dusseldorf, Frankfurt, Hamburg, Hannover, Munich, Nuremburg, Stuttgart

DataGeneral

Ordering Technical Publications

TIPS is the Technical Information and Publications Service—a new support system for DGC customers that makes ordering technical manuals simple and fast. Simple, because TIPS is a central supplier of literature about DGC products. And fast, because TIPS specializes in handling publications.

TIPS was designed by DG's Educational Services people to follow through on your order as soon as it's received. To offer discounts on bulk orders. To let you choose the method of shipment you prefer. And to deliver within a schedule you can live with.

How to Get in Touch with TIPS

Contact your local DGC education center for brochures, prices, and order forms. Or get in touch with a TIPS administrator directly by calling (617) 366-8911, extension 4086, or writing to

Data General Corporation
Attn: Educational Services, TIPS Administrator
MS F019
4400 Computer Drive
Westborough, MA 01580

TIPS. For the technical manuals you need, when you need them.

DGC Education Centers

Boston Education Center
Route 9
Southboro, Massachusetts 01772
(617) 485-7270

Washington, D.C. Education Center
7927 Jones Branch Drive, Suite 200
McLean, Virginia 22102
(703) 827-9666

Atlanta Education Center
6855 Jimmy Carter Boulevard, Suite 1790
Norcross, Georgia 30071
(404) 448-9224

Los Angeles Education Center
5250 West Century Boulevard
Los Angeles, California 90045
(213) 670-4011

Chicago Education Center
703 West Algonquin Road
Arlington Heights, Illinois 60005
(312) 364-3045

Data General

Technical Products Publications

Comment Form

Please help us improve our future publications by answering the questions below. Use the space provided for your comments.

Title: _____

Document No. 069-400202-00

Yes	No		
☐	☐	Is this manual easy to read?	○ You (can, cannot) find things easily. ○ Other: ○ Language (is, is not) appropriate. ○ Technical terms (are, are not) defined as needed.
		In what ways do you find this manual useful?	○ Learning to use the equipment ○ To instruct a class. ○ As a reference ○ Other: ○ As an introduction to the product
☐	☐	Do the illustrations help you?	○ Visuals (are,are not) well designed. ○ Labels and captions (are,are not) clear. ○ Other:
☐	☐	Does the manual tell you all you need to know? What additional information would you like?	
☐	☐	Is the information accurate? (If not please specify with page number and paragraph.)	

Name: _____ Title: _____

Company: _____ Division: _____

Address: _____ City: _____

State: _____ Zip: _____ Telephone: _____ Date: _____

CUT ALONG DOTTED LINE

DG-06895

Data General

Data General Corporation, Westboro, Massachusetts 01580

FOLD FOLD

TAPE TAPE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:

Data General

ATTN: Technical Products Publications (C-138)
4400 Computer Drive
Westboro, MA 01581

DataGeneral

users group

Installation Membership Form

Name _____ Position _____ Date _____

Company, Organization or School _____

Address _____ City _____ State _____ Zip _____

Telephone: Area Code _____ No. _____ Ext. _____

CUT ALONG DOTTED LINE

1. Account Category

- ☐ OEM
- ☐ End User
- ☐ System House
- ☐ Government
- ☐ Educational

2. Hardware

	Qty. Installed	Qty. On Order
M/600		
COMMERCIAL ECLIPSE		
SCIENTIFIC ECLIPSE		
AP/130		
CS Series		
Mapped NOVA		
Unmapped NOVA		
microNOVA		
Other (Specify) _____		
_____		

3. Software

- ☐ AOS
- ☐ DOS
- ☐ MP/OS
- ☐ RDOS
- ☐ Other

Specify _____

4. Languages

- ☐ Algol
- ☐ DG/L
- ☐ Cobol
- ☐ PASCAL
- ☐ Business BASIC
- ☐ BASIC
- ☐ Assembler
- ☐ Fortran
- ☐ RPG II
- ☐ PL/1
- ☐ Other

Specify _____

5. Mode of Operation

- ☐ Batch (Central)
- ☐ Batch (Via RJE)
- ☐ On-Line Interactive

6. Communications

- ☐ HASP
- ☐ RJE80
- ☐ RCX 70
- ☐ CAM
- ☐ XODIAC
- ☐ Other

Specify _____

7. Application Description

○ _____

8. Purchase

From whom was your machine(s) purchased?

- ☐ **Data General Corp.**
- ☐ Other

Specify _____

9. Users Group

Are you interested in joining a special interest or regional Data General Users Group?

○ _____

DataGeneral

Data General Corporation, Westboro, Massachusetts 01580. (617) 366-8911

FOLD

TAPE

FOLD

TAPE

FOLD

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:

Data General

ATTN: Users Group Coordinator (C-228)
4400 Computer Drive
Westboro, MA 01581

Data General

Data General Corporation, Westboro, Massachusetts 01580

069-400202-00