SC24-5220-0
File No. S370-39

Program Product

IBM Virtual Machine/ System Product: System Product Editor User's Guide

Program Number 5664-167

IBM

Acknowledgement

We gratefully acknowledge the permission to reprint excerpts from the following:

The People's Almanac, by David Wallechinsky and Irving Wallace. Copyright © 1975 by David Wallace and Irving Wallace. Reprinted by permission of Doubleday & Company, Inc.

I Wouldn't Have Missed It, by Ogden Nash, reprinted by permission of Curtis Brown, Ltd.

Copyright 1930, 1931, 1932, 1933, 1934, 1935, 1936, 1937, 1938, 1939, 1940, 1942, 1943, 1947, 1948, 1949, 1950, 1951, 1952, 1953, 1954. © 1955, 1956, 1957, 1959, 1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971 by Ogden Nash. Copyright 1933, 1934, 1935, 1936, 1939, 1940, 1941, 1942, 1943, 1944, 1945, 1947, 1948 by the Curtis Publishing Company. Copyright 1952 by Cowles Magazines, Inc. Copyright © 1969, 1970, 1971, 1972, 1975 by Isabel Eberstadt and Linell Smith.

Copyrights Renewed © 1957, 1958, 1960, 1961, 1962, 1963, 1964, 1965, 1966, 1968, 1970 by Ogden Nash. Renewed © 1963, 1964 by the Curtis Publishing Company. Renewed © by the Saturday Evening Post Company.

First Edition (July 1980)

This edition, SC24-5220-0, applies to the initial release of IBM Virtual Machine/System Product, Program Number 5664-167, and to all subsequent versions and releases until otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the information herein; before using this publication in connection with the operation of IBM systems, consult the *IBM System/370 and 4300 Processors Bibliography,* GC20-0001, for the editions that are applicable and current.

Preface

This book was written primarily for the individual who has limited data processing experience. It is designed to give you a working knowledge of the System Product editor (also referred to as XEDIT).

The System Product editor provides a wide range of functions for text processing and program development. Both a full screen and a line mode editor, it can be used on display and typewriter terminals.

Some highlights of the editor discussed in this book are:

- Extended string search facilities for improved text processing
- Automatic "wrapping" of lines that are longer than a screen line
- The ability to enter selected subcommands directly on a displayed line
- The ability to tailor the full screen layout
- The ability to divide the screen in order to display multiple views of the same or of different files
- A variety of macros for improved text processing, such as macros to join and split lines
- A HELP facility that provides an on-line full screen display of any XEDIT subcommand or macro (or any command in the CMS HELP facility) during an editing session.

How To Use This Book

This book relies on "before-and-after" examples that illustrate the text. You can also try out these examples for practice.

The first three chapters are intended for data processing novices:

Chapter 1: An XEDIT Subset: Full Screen Text Processing is written for the inexperienced user who has a display terminal used in full screen mode. It defines a subset of XEDIT subcommands that perform commonly-used editing functions.

Chapter 2: A Practice Exercise is designed to give you practice in using the subcommands presented in Chapter 1. It is an interactive text, that is, it walks you through an editing session, step by step.

Chapter 3: An XEDIT Subset: Text Processing on a Typewriter Terminal is similar to Chapter 1, but is written for a new user who has a typewriter terminal.

The last four chapters are intended both for new users who have mastered the fundamentals and for data processing professionals. These chapters introduce more sophisticated editing functions:

Chapter 4: Using Targets explains how to use the editor's extended string search facilities. Targets are used to move the line pointer and to define the scope of many XEDIT subcommands.

Chapter 5: Editing Multiple Files explains how to edit multiple files and how to divide the screen into multiple logical screens for multiple views of the same or of different files.

Chapter 6: Tailoring the Screen explains how you can alter the screen layout to suit yourself.

Chapter 7: The Macro Language explains how to write XEDIT macros and also explains how to write a profile macro.

The Appendix is a summary of all XEDIT subcommands and their functions. These subcommands are described in detail in the publication *VM/SP: System Product Editor Command and Macro Reference.*

Related Publications

IBM Virtual Machine/System Product: System Product Editor Command and Macro Reference, SC24-5221

IBM Virtual Machine/System Product: EXEC 2 Reference, SC24-5219

Contents

Figures

Chapter 1: An XEDIT Subset: Full Screen Text Processing

This chapter is written primarily for the person who has limited data processing experience; however, some VM/SP CMS experience is assumed. For example, you must know how to log on to VM/SP and enter the CMS environment. You should also be familiar with the concept of a CMS file.

When you finish this chapter, you should have a working knowledge of the editor. The subcommands presented here comprise a subset of XEDIT subcommands, with which you can create a file, enter data, manipulate the screen, make changes to the file, and transfer data between files.

The editor has many additional capabilities, which are described in the rest of this book and in the publication *VM/SP: System Product Editor Command and Macro Reference*.

This subset has been selected for text processing on a display terminal used in full screen mode. (If you have a typewriter terminal, refer to Chapter 3.)

Editing a File

To edit a file means to make changes, additions, or deletions to a CMS file that is on a disk, and to make these changes interactively: you instruct the editor to make a change, the editor makes it, and then you request another change.

You can edit a file that does not exist; when you do so, you are creating a file.

XEDIT Command

After you log on to VM/SP and enter the CMS environment, you are ready to enter the edit environment and begin creating a file. The editor is invoked with the CMS command XEDIT, whose format is as follows:

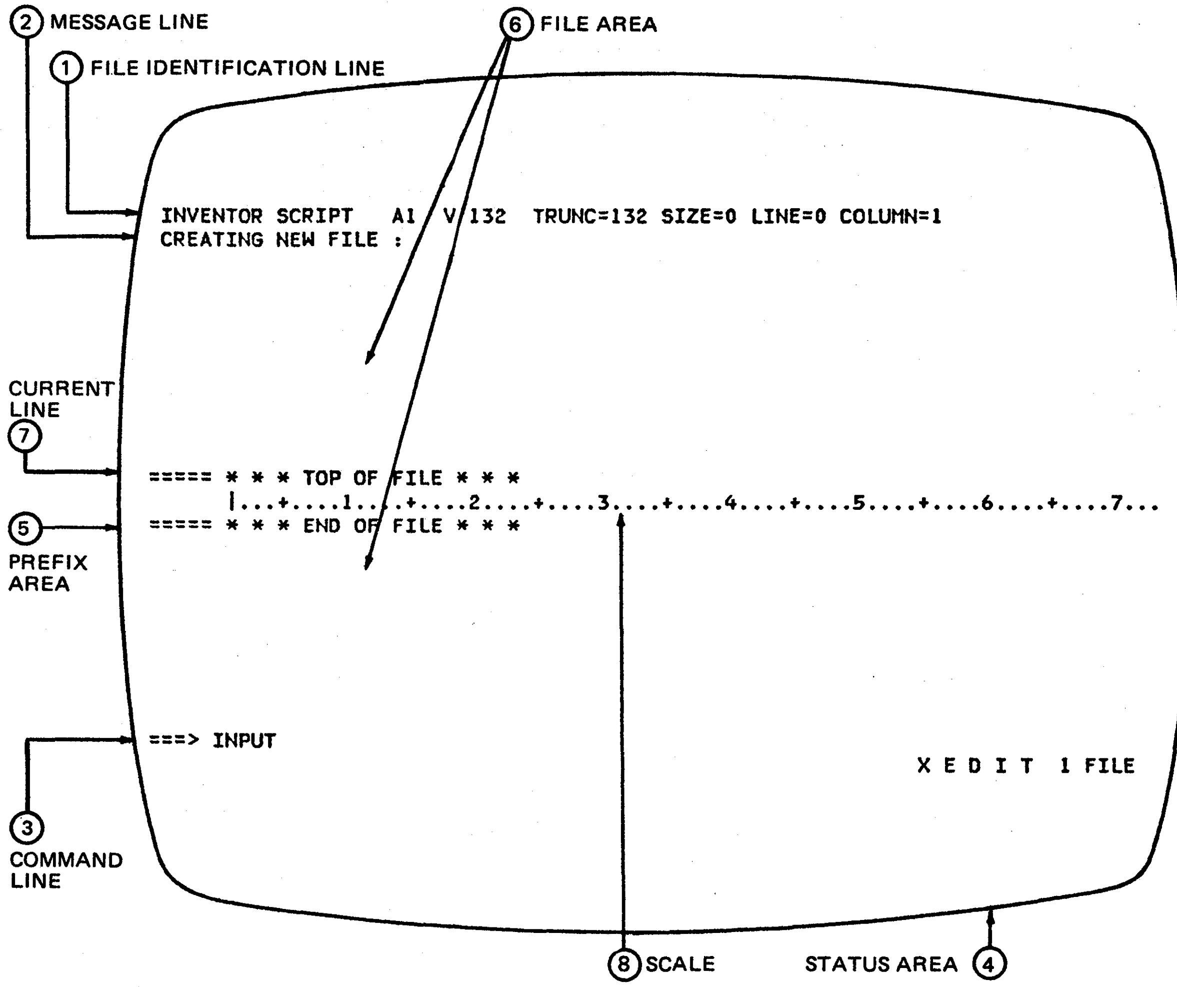
```
XEDIT filename filetype
```

In Figure 1-1, the editor was invoked with the following command:

```
XEDIT INVENTOR SCRIPT
```

Before we see how to enter data in the file, let's look at the screen layout illustrated in Figure 1-1.

Screen Layout



```
INVENTOR SCRIPT   A1  V 132  TRUNC=132 SIZE=0 LINE=0 COLUMN=1
CREATING NEW FILE :



===== * * * TOP OF FILE * * *
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== * * * END OF FILE * * *




===> INPUT
                                                         X E D I T  1 FILE
```

Figure 1-1. The Screen Layout

(1.) File Identification Line
The first line on the screen identifies the file being edited. The following information is displayed:

a. filename, filetype, filemode
If you do not specify a filemode, the editor assigns a filemode of "A1", which means that the file is to become part of a file collection called your "A-disk".

b. record format and record length
The record format and record length (V 132) shown in the example mean that in this file, the length of a line can vary and the file will hold lines up to 132 characters long. Therefore, a file line can be longer than a screen line.

c. truncation column (TRUNC=)
Notice that the truncation column is the same as the record length (132). Since a file line can be only 132 characters long, any data that is entered beyond 132 characters (in total) is truncated.

d. current number of lines in the file (SIZE=)
(Since we have not yet entered data in the file, the number of lines is zero.)

e. file line number of the current line (LINE=)
   (See number 7, below.)

f. position of the column pointer (COLUMN=)
   (See number 8, below.)

2. Message Line
The editor communicates with you by displaying messages on the second line of the screen. These messages tell you if you have made an error, or they provide information. In Figure 1-1, the message line shows that you are creating a new file.

3. Command Line
The large arrow (===>) at the bottom of the screen points to the command input area. One of the ways you communicate with the editor is by entering XEDIT subcommands on this line. Subcommands can be typed in either uppercase or lowercase, or a combination of both, and many can be abbreviated. For example, "INPUT", "Input", and "i" are all valid ways to type the INPUT subcommand.

After typing a subcommand on the command line, you must press the ENTER key to execute the subcommand. Figure 1-1 shows the subcommand "INPUT" typed in the command line. (To move the cursor from any place on the screen to the command line, just press the ENTER key.)

4. Status Area
The lower right corner displays the current status of your editing session, for example, edit mode or input mode, and the number of files you are editing. The status area in Figure 1-1 shows that one file is being edited.

5. Prefix Area
The prefix area is the five left-most columns on the screen and displays five equal signs (=====). Each line in the file has a prefix area associated with it.

You can perform various editing tasks, like deleting a line, by entering one-character commands, called "prefix subcommands", in the prefix area of any line.

6. File Area
The rest of the screen is available to display the file.

You can make changes to the file by moving the cursor under any line and typing over the characters, or by using special keys to insert or delete characters. You can make as many changes as you want on the displayed lines before pressing the ENTER key. When you press the ENTER key, the corresponding changes are made to the copy of the file that is kept in virtual storage. At the end of the editing session, a FILE subcommand will permanently record those changes on the copy of the file that resides on disk.

Since a file may be too long to fit on one screen, various subcommands are used to scroll the screen so that you can move forward and backward in a file.

7. The Current Line
The current line is the file line in the middle of the screen (above the scale). It appears brighter than the other file lines or is "highlighted".

In Figure 1-1, the current line is the "TOP OF FILE" line; at this point, the file contains no data.

The current line is an important concept, because most subcommands perform their functions starting with the current line. Naturally, the line that is current changes during an editing session as you scroll the screen, move up and down, and so forth. When the current line changes, we say that the line pointer (not visible on the screen) has moved. Many XEDIT subcommands perform their functions starting with the current line, and move the line pointer when they are finished.

8. Scale
The scale appears under the current line to assist you in editing. It's like the margin scale on a typewriter.

The vertical bar (|) that appears in column one on the scale is the *column pointer.* Various subcommands perform their functions within a line starting at the column pointer, which you can move to different positions on the scale by using XEDIT subcommands that will be discussed later. The column under which the column pointer is positioned is called the current column.

Entering Data

After you enter the XEDIT command, you are in *edit mode.* You must be in edit mode to enter XEDIT subcommands.

You can enter data into the file using *input mode* or *power typing mode,* which are discussed in the following sections.

INPUT Subcommand

To enter input mode, type the following subcommand in the command line and press the ENTER key:

```
===> INPUT
```

You can then type in your data in the input zone, which is the bottom half of the screen (between the scale and the command line).

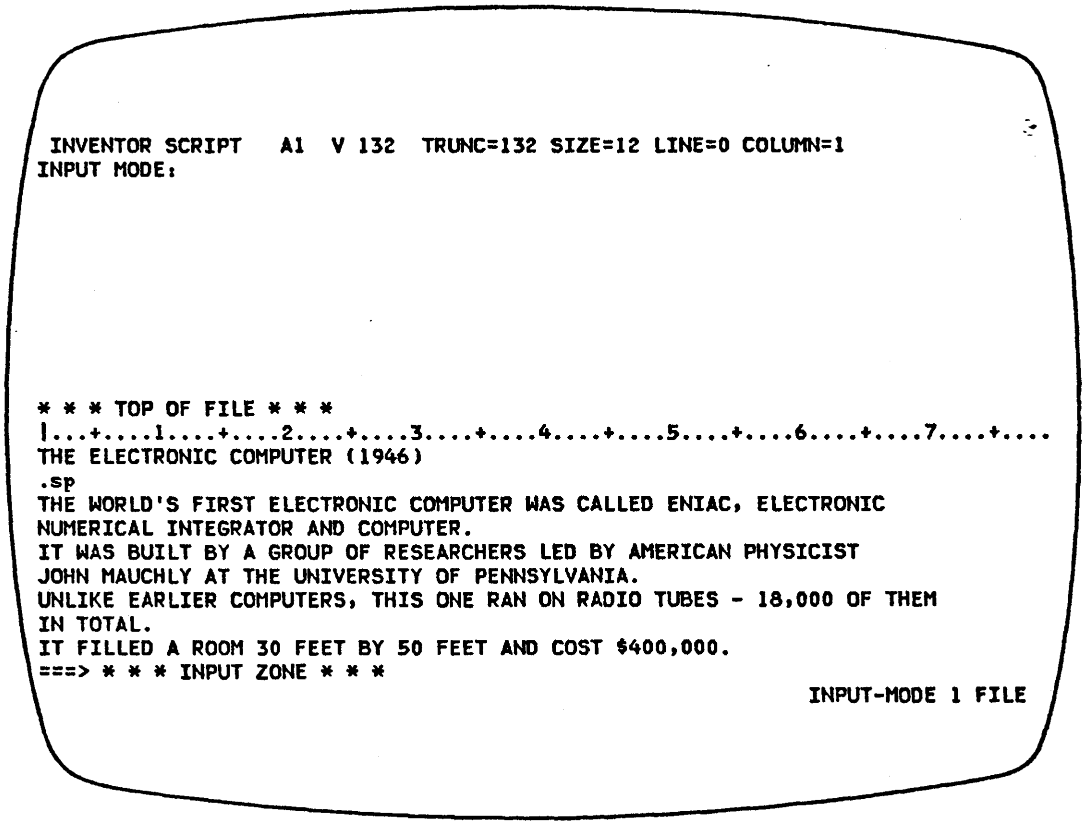
Figure 1-2 is the same file, INVENTOR SCRIPT, that is shown in Figure 1-1. However, the INPUT subcommand has been entered and the lines of data have been typed on the screen. Notice how the screen changes in input mode: the prefix areas (=====) disappear; the message line and status area tell you that you are in input mode; the command line contains the phrase "INPUT ZONE", which marks the end of the input zone and reminds you that you cannot enter subcommands in input mode.

In Figure 1-2 (Part 1), the entire input zone has been filled. To stay in input mode and type more data, press the ENTER key once. The lines that you typed move to the top half of the screen, with the last line you typed becoming the new current line. The input zone is available to type more data, as shown in Figure 1-2 (Part 2).

If you have no more data to type, pressing the ENTER key again takes you out of input mode and back into edit mode.

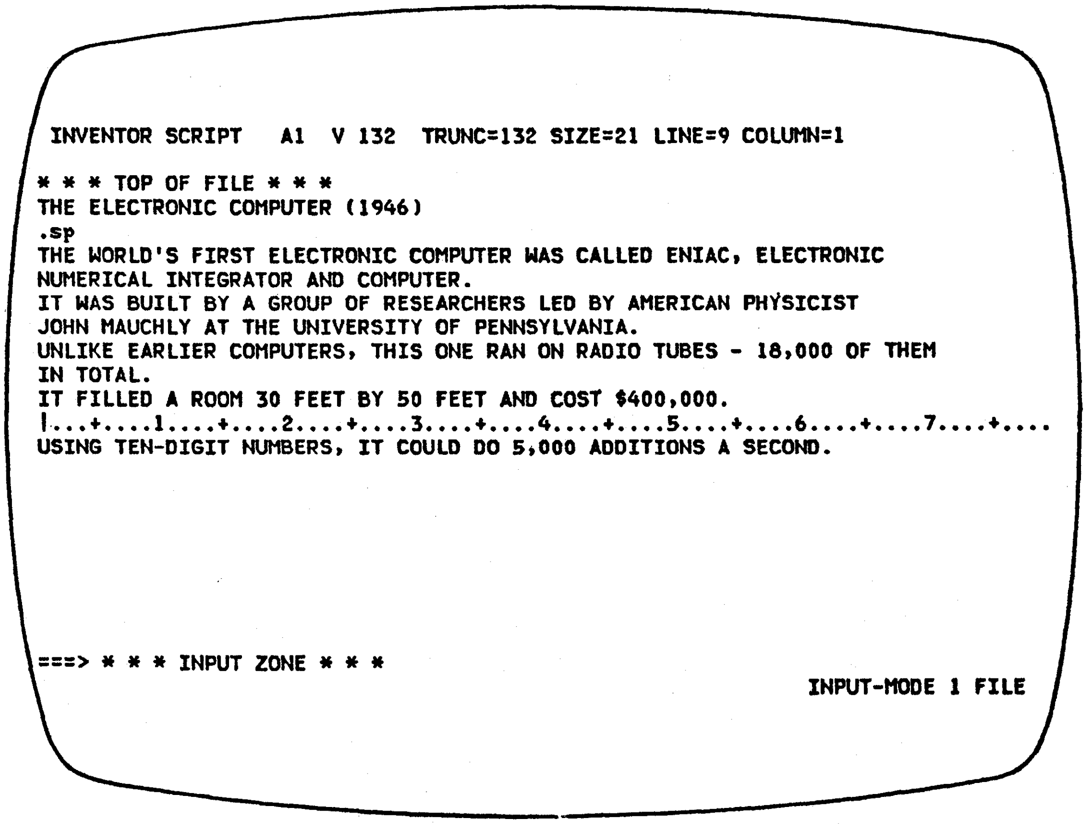
Figure 1-2 (Part 3) shows how the data looks in the file, after the ENTER key has been pressed twice. The display is restored to the edit mode screen layout described in Figure 1-1, and the file contains the data.

During an editing session, you can enter input mode at any time to insert new lines of data in the file. As you have seen, after the INPUT subcommand is entered, the editor makes room for you to type new lines of data after the current line. In this example, since the file was new and the INPUT subcommand was the first subcommand entered, the TOP OF FILE line was the current line. Later, you will see how to make any line current, so that you can insert lines in input mode between any two existing lines in the file.



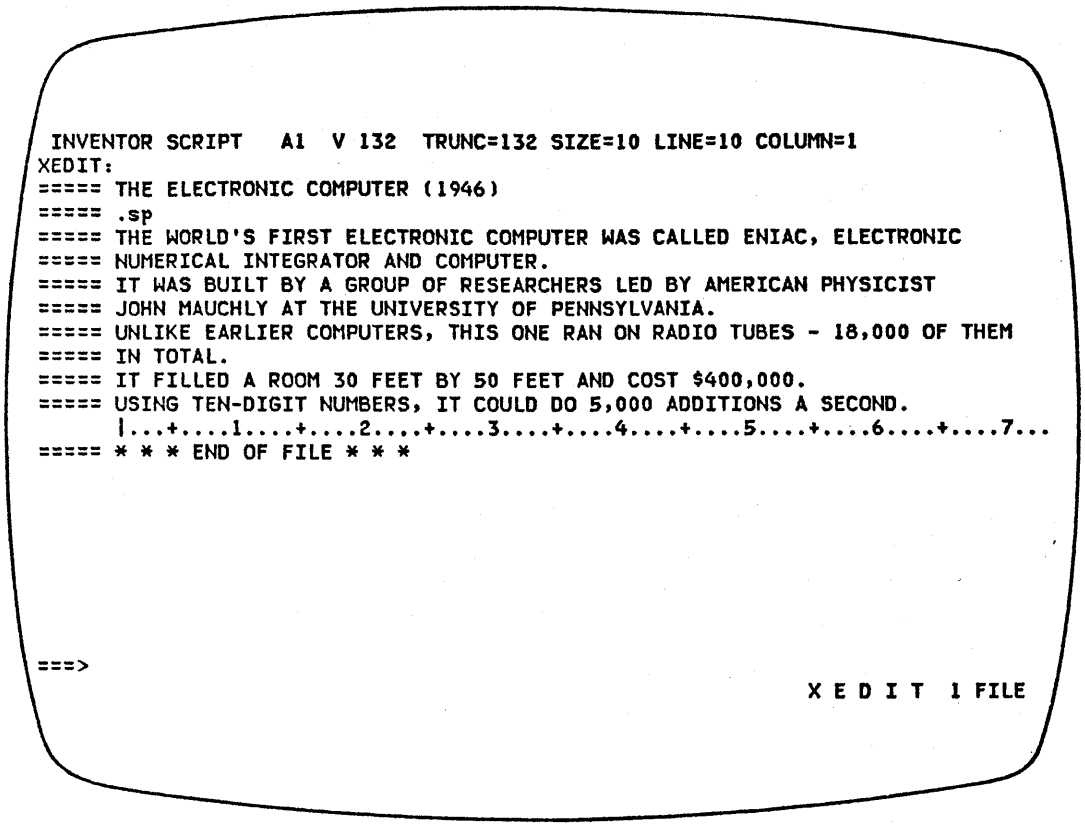
```
 INVENTOR SCRIPT   A1  V 132  TRUNC=132 SIZE=12 LINE=0 COLUMN=1
INPUT MODE:







* * * TOP OF FILE * * *
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....
THE ELECTRONIC COMPUTER (1946)
.sp
THE WORLD'S FIRST ELECTRONIC COMPUTER WAS CALLED ENIAC, ELECTRONIC
NUMERICAL INTEGRATOR AND COMPUTER.
IT WAS BUILT BY A GROUP OF RESEARCHERS LED BY AMERICAN PHYSICIST
JOHN MAUCHLY AT THE UNIVERSITY OF PENNSYLVANIA.
UNLIKE EARLIER COMPUTERS, THIS ONE RAN ON RADIO TUBES - 18,000 OF THEM
IN TOTAL.
IT FILLED A ROOM 30 FEET BY 50 FEET AND COST $400,000.
===> * * * INPUT ZONE * * *
                                                       INPUT-MODE 1 FILE
```

Figure 1-2. Input Mode - Typing the Data (Part 1 of 3)



```
 INVENTOR SCRIPT   A1  V 132  TRUNC=132 SIZE=21 LINE=9 COLUMN=1

* * * TOP OF FILE * * *
THE ELECTRONIC COMPUTER (1946)
.sp
THE WORLD'S FIRST ELECTRONIC COMPUTER WAS CALLED ENIAC, ELECTRONIC
NUMERICAL INTEGRATOR AND COMPUTER.
IT WAS BUILT BY A GROUP OF RESEARCHERS LED BY AMERICAN PHYSICIST
JOHN MAUCHLY AT THE UNIVERSITY OF PENNSYLVANIA.
UNLIKE EARLIER COMPUTERS, THIS ONE RAN ON RADIO TUBES - 18,000 OF THEM
IN TOTAL.
IT FILLED A ROOM 30 FEET BY 50 FEET AND COST $400,000.
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....
USING TEN-DIGIT NUMBERS, IT COULD DO 5,000 ADDITIONS A SECOND.









===> * * * INPUT ZONE * * *
                                                       INPUT-MODE 1 FILE
```

Figure 1-2. Input Mode - Continue Typing (Part 2 of 3)



```
 INVENTOR SCRIPT   A1  V 132  TRUNC=132 SIZE=10 LINE=10 COLUMN=1
XEDIT:
===== THE ELECTRONIC COMPUTER (1946)
===== .sp
===== THE WORLD'S FIRST ELECTRONIC COMPUTER WAS CALLED ENIAC, ELECTRONIC
===== NUMERICAL INTEGRATOR AND COMPUTER.
===== IT WAS BUILT BY A GROUP OF RESEARCHERS LED BY AMERICAN PHYSICIST
===== JOHN MAUCHLY AT THE UNIVERSITY OF PENNSYLVANIA.
===== UNLIKE EARLIER COMPUTERS, THIS ONE RAN ON RADIO TUBES - 18,000 OF THEM
===== IN TOTAL.
===== IT FILLED A ROOM 30 FEET BY 50 FEET AND COST $400,000.
===== USING TEN-DIGIT NUMBERS, IT COULD DO 5,000 ADDITIONS A SECOND.
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== * * * END OF FILE * * *




===>
                                                           X E D I T  1 FILE
```

Figure 1-2. Input Mode - Data Entered in the File (Part 3 of 3)

POWER Subcommand

The easiest way to enter a large amount of text, like one long paragraph, is by using "power typing". To use power typing, enter the following subcommand:

```
===> POWER
```

The advantage of using power typing is that you can enter data as if the screen were one long line. You do not have to be concerned with line length or word length - you can start typing a word on one line of the screen and finish it on the next. In fact, if you're a skilled typist, you don't even have to look at the screen. When you reach the end of a line, the editor automatically "wraps around" to the beginning of the next line. You can type continuously until the screen is filled.

If you fill up a screen and want to continue typing in power typing mode, press the ENTER key once. The last line you typed is displayed at the top of the screen; the rest of the screen is blank and you can continue typing.

When you are finished typing, press the ENTER key twice to exit from power typing and re-enter edit mode. The editor automatically divides the data into appropriate screen lines and reconstructs any split words.

During an editing session, you can use power typing at any time by entering the POWER subcommand. The data entered using power typing is inserted *after the current line,* as it is when you use the INPUT subcommand.

Causing A Break in the Data

If you want to cause a break in the data that you type in power typing mode, that is, you want data to start on a new line (for example, a new paragraph or SCRIPT/VS control words, which must start in column one), you can type a line end character before the data that you want to start on a new line. The default line end character is a pound sign (#).

For example, if the following data is typed in power typing mode:

```
.sp#A pound sign causes the data to start on a new line.#.sp
```

The data will be entered in the file as:

```
===== .sp
===== A pound sign causes the data to start on a new line.
===== .sp
```

Inserting Characters

If you want to insert characters or spaces in a line while you are in power typing mode, you can use the insert mode key. When characters are inserted, the entire stream of data shifts to the right; it's like inserting a box car in a train. Remember to press the RESET key when you are finished inserting characters.

An Example of Power Typing

Figure 1-3 (Part 1) illustrates the same file, INVENTOR SCRIPT, but the data was typed in power typing mode, after the POWER subcommand was entered. The screen changes in several ways in power typing mode: the prefix and status areas disappear; the line that was current when the POWER subcommand was entered moves to the top of the screen, and the rest of the screen is available for typing data. Notice how a word can start at the end of a line and finish on the next. The entire screen can be filled with data, but it doesn't have to be.

Notice the pound signs (#) in the eighth line (from the top of the screen). A pound sign causes the data that follows it to begin on a new line when it is entered into the file. The pound sign itself is not entered in the file.

Figure 1-3 (Part 2) shows how the screen looks after the ENTER key was pressed twice. The screen layout is restored, and the words and lines are reconstructed. Any data that was preceded by a pound sign begins on a new line.

Using Program Function (PF) Keys

Each PF key is set to an XEDIT subcommand, which is executed when the key is pressed. Using the PF key saves you the time it takes to type that subcommand on the command line and press the ENTER key.

You can use the following subcommand to display the PF key settings:

```
===> QUERY PF
```

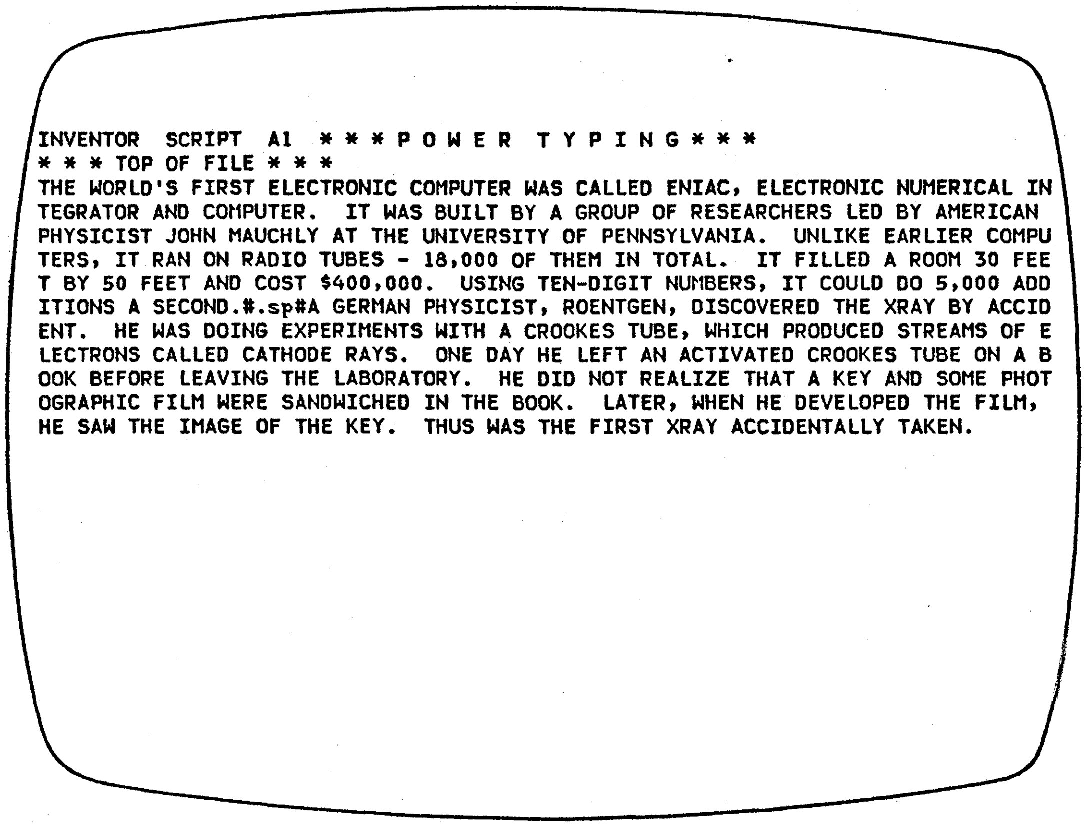
The initial settings are as follows:

13	PF1	HELP MENU
14	PF2	SOS LINEADD
15	PF3	QUIT
16	PF4	TABKEY
17	PF5	SCHANGE 6
18	PF6	?
19	PF7	BACKWARD
20	PF8	FORWARD
21	PF9	=
22	PF10	SPLIT CURSOR
23	PF11	JOIN CURSOR
24	PF12	CURSOR COLUMN

These are the subcommands that the editor assigns to the PF keys. If you would rather have a different subcommand assigned to one (or more) of the PF keys, you can use the SET PF subcommand, whose format is as follows:

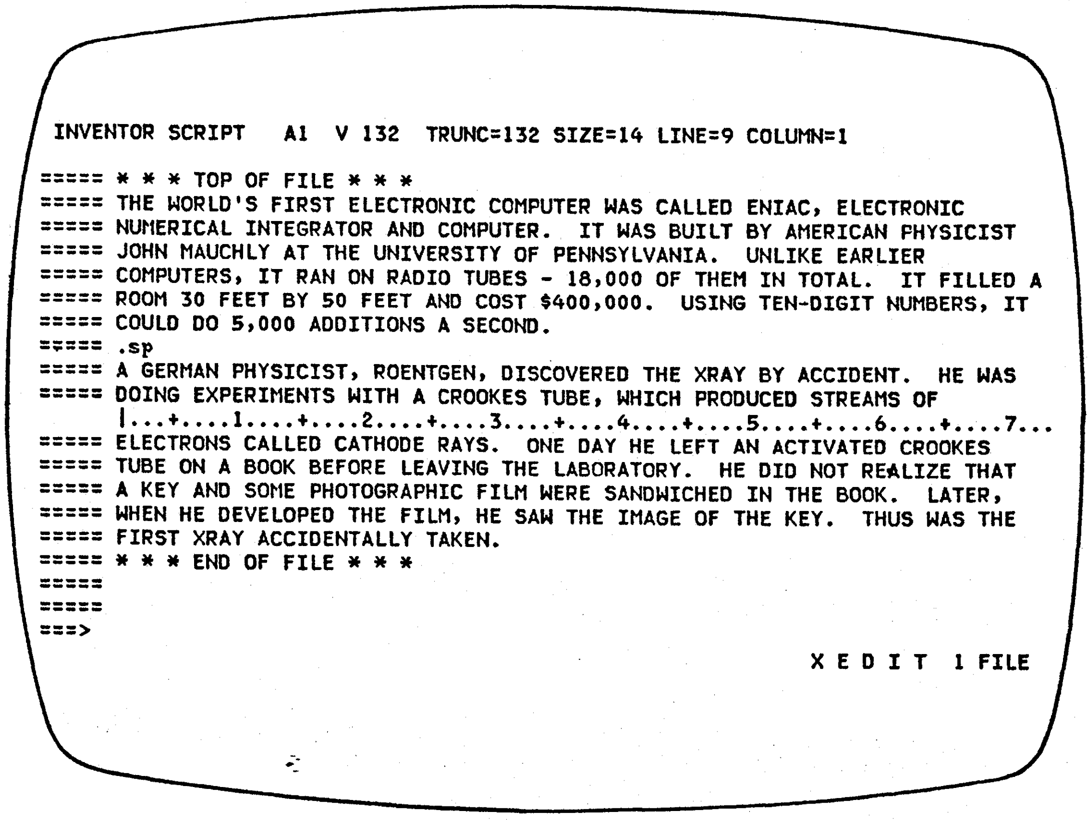
```
===> SET PFn subcommand
```

where "n" is a PF key number, and "subcommand" is any XEDIT subcommand.



```
INVENTOR  SCRIPT  A1  * * * P O W E R   T Y P I N G * * *
* * * TOP OF FILE * * *
THE WORLD'S FIRST ELECTRONIC COMPUTER WAS CALLED ENIAC, ELECTRONIC NUMERICAL IN
TEGRATOR AND COMPUTER.  IT WAS BUILT BY A GROUP OF RESEARCHERS LED BY AMERICAN
PHYSICIST JOHN MAUCHLY AT THE UNIVERSITY OF PENNSYLVANIA.  UNLIKE EARLIER COMPU
TERS, IT RAN ON RADIO TUBES - 18,000 OF THEM IN TOTAL.  IT FILLED A ROOM 30 FEE
T BY 50 FEET AND COST $400,000.  USING TEN-DIGIT NUMBERS, IT COULD DO 5,000 ADD
ITIONS A SECOND.#.sp#A GERMAN PHYSICIST, ROENTGEN, DISCOVERED THE XRAY BY ACCID
ENT.  HE WAS DOING EXPERIMENTS WITH A CROOKES TUBE, WHICH PRODUCED STREAMS OF E
LECTRONS CALLED CATHODE RAYS.  ONE DAY HE LEFT AN ACTIVATED CROOKES TUBE ON A B
OOK BEFORE LEAVING THE LABORATORY.  HE DID NOT REALIZE THAT A KEY AND SOME PHOT
OGRAPHIC FILM WERE SANDWICHED IN THE BOOK.  LATER, WHEN HE DEVELOPED THE FILM,
HE SAW THE IMAGE OF THE KEY.  THUS WAS THE FIRST XRAY ACCIDENTALLY TAKEN.
```

Figure 1-3. Power Typing (Part 1 of 2)



```
 INVENTOR SCRIPT    A1   V 132  TRUNC=132 SIZE=14 LINE=9 COLUMN=1

===== * * * TOP OF FILE * * *
===== THE WORLD'S FIRST ELECTRONIC COMPUTER WAS CALLED ENIAC, ELECTRONIC
===== NUMERICAL INTEGRATOR AND COMPUTER.  IT WAS BUILT BY AMERICAN PHYSICIST
===== JOHN MAUCHLY AT THE UNIVERSITY OF PENNSYLVANIA.  UNLIKE EARLIER
===== COMPUTERS, IT RAN ON RADIO TUBES - 18,000 OF THEM IN TOTAL.  IT FILLED A
===== ROOM 30 FEET BY 50 FEET AND COST $400,000.  USING TEN-DIGIT NUMBERS, IT
===== COULD DO 5,000 ADDITIONS A SECOND.
===== .sp
===== A GERMAN PHYSICIST, ROENTGEN, DISCOVERED THE XRAY BY ACCIDENT.  HE WAS
===== DOING EXPERIMENTS WITH A CROOKES TUBE, WHICH PRODUCED STREAMS OF
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== ELECTRONS CALLED CATHODE RAYS.  ONE DAY HE LEFT AN ACTIVATED CROOKES
===== TUBE ON A BOOK BEFORE LEAVING THE LABORATORY.  HE DID NOT REALIZE THAT
===== A KEY AND SOME PHOTOGRAPHIC FILM WERE SANDWICHED IN THE BOOK.  LATER,
===== WHEN HE DEVELOPED THE FILM, HE SAW THE IMAGE OF THE KEY.  THUS WAS THE
===== FIRST XRAY ACCIDENTALLY TAKEN.
===== * * * END OF FILE * * *
=====
=====
===>
                                                          X E D I T  1 FILE
```

Figure 1-3. Power Typing - Data Entered in the File (Part 2 of 2)

For example:

```
===> SET PF1 INPUT
```

assigns the INPUT subcommand to the PF1 key. Pressing the PF1 key would immediately place you in input mode.

When you assign a subcommand to a PF key, the setting remains in effect only for the current editing session. In the next editing session, the initial settings shown above are in effect.

The following sections show how to use some of the PF keys (initial settings). Others will be discussed where appropriate.

Splitting and Joining Lines

Two PF keys that are useful in text processing are PF10 and PF11, which allow you to split and join lines, respectively, *at the cursor position.*

Splitting a Line (PF10)

To split a line in two, simply move the cursor under the character where you want the line to be split, and press the PF10 key.

In the following line, note the position of the cursor, under the "F" in "FOOD".

```
===== GILA MONSTERS HOLD RESERVE FOOD SUPPLIES IN THEIR TAILS.
```

Pressing the PF10 key produces the following lines:

```
===== GILA MONSTERS HOLD RESERVE _
===== FOOD SUPPLIES IN THEIR TAILS.
```

The PF10 key is particularly useful if you want to add information to a line. In the following line, the cursor is placed under the "I" in "IN":

```
===== BIRD SPECIES HAVE DWINDLED IN THE LAST 70 MILLION YEARS.
```

When the PF10 key is pressed, the line is split in two:

```
===== BIRD SPECIES HAVE DWINDLED _
===== IN THE LAST 70 MILLION YEARS.
```

Now there's room to add information on the line:

```
===== BIRD SPECIES HAVE DWINDLED FROM 1.5 MILLION TO 10,000
===== IN THE LAST 70 MILLION YEARS.
```

Joining Two Lines (PF11)

Pressing the PF11 key joins two lines at the cursor position.

For example:

```
===== These lines are _
===== too short.
```

Note the cursor position above. Pressing the PF11 key produces the following line:

```
===== These lines are too short.
```

Keep in mind that the line that is appended, or joined, overlays any data that follows the cursor.

For example:

```
===== The phrase "Things get worse under pressure"
===== is to be deleted.
```

Pressing the PF11 key overlays the data that follows the cursor in the first line and results in the following:

```
===== The phrase is to be deleted.
```

Scrolling Backward and Forward

When a file is too long to fit on one screen, you can use the PF7 and PF8 keys to scroll back and forth through the file.

Pressing the PF7 key, which is set to the BACKWARD subcommand, scrolls the screen backward, toward the top of the file, for one screen display.

Conversely, pressing the PF8 key, which is set to the FORWARD subcommand, scrolls the screen forward, toward the end of the file, for one screen display.

You can press either key repeatedly to scroll back or forth for as many screens as you wish.

Redisplaying a Subcommand

After a subcommand that has been typed in the command line is executed, the command line is cleared. If you use a PF key, the subcommand doesn't appear in the command line at all. Sometimes, you'd like to be able to see the last subcommand that was executed. Perhaps you pressed the wrong PF key, or you didn't enter a subcommand the way you intended to.

Pressing the PF6 key (which is set to the ? subcommand) displays, in the command line, the last subcommand that was executed.

You can then re-execute the subcommand simply by pressing the ENTER key. If the subcommand was entered incorrectly, you can correct the error by typing over the subcommand displayed in the command line and then pressing the ENTER key.

Re-executing a Subcommand

Use the PF9 key, which is set to the = subcommand, to re-execute the last subcommand entered. The subcommand does not appear in the command line, as it does when the PF6 key (which is set to the ? subcommand) is used.

Each time the PF9 key is pressed, the subcommand is executed, thereby saving you the time it takes to re-type the subcommand.

Inserting Words in a Line

Using the PA2 Key

One way to insert letters, spaces, or words in a line is by pressing the PA2 key (or its equivalent) and then by using the insert mode key.

The PA2 key replaces blank spaces at the end of a line with null characters; it "makes room" for the characters in the line to be shifted over so that new ones can be inserted.

The PA2 key operates on only one line at a time; if you move the cursor to another line and want to use insert mode, you must press the PA2 key again.

Remember to press the RESET key when you are finished using insert mode.

This method may be used in both input mode and edit mode, but not in power typing mode. You cannot set the PA2 key to any other function.

Using the SET NULLS Subcommand

If you have insertions to make on many lines, you can issue the following subcommand:

```
===> SET NULLS ON
```

Then, you can use the insert mode key without pressing the PA2 key for each line. When you are finished inserting words, issue the following subcommand:

```
===> SET NULLS OFF
```

(In power typing mode, you can use the insert mode key without issuing a SET NULLS ON subcommand.)

Using Prefix Subcommands

Prefix subcommands are one-character commands used to perform basic editing tasks on a particular line.

The following prefix subcommands are described in this section:

A (add)
D (delete)
" (duplicate)
M (move)
C (copy)
F (following)
P (preceding)
/ (set current line)

Prefix subcommands are entered by typing over any position of the five-character prefix area on one or more lines. When the ENTER key is pressed, all of the prefix subcommands that have been typed on the screen are executed.

Adding and Deleting Lines

A Prefix Subcommand

To add a line, type the single character "A" in the prefix area. When the ENTER key is pressed, a blank line is inserted immediately following the line containing the "A". A number may precede or follow the "A" to indicate that more than one line is to be added. For example, "A5" causes five blank lines to be added.

The following are valid ways to type the A prefix subcommand:

```
====A   Adds one blank line after this line.
a====   Adds one blank line after this line.
10a==   Adds ten blank lines after this line.
===A5   Adds five blank lines after this line.
```

Information may then be typed in the added lines. If no information is typed, the blank lines remain in the file throughout the editing session and after the file is written to disk.

D Prefix Subcommand

To delete a line, enter the single character "D" in the prefix area of a line.

A number may precede or follow the "D" to indicate that more than one line is to be deleted.

To delete a group of consecutive lines, that is, a block of lines, you can enter the double character "DD" in the prefix area of both the first and last lines to be deleted. This method makes it unnecessary for you to count the number of lines to be deleted.

For example:

```
==dd= This is the first line I want to remove.
===== This is the second.
===== This is the third.
===== This is the fourth.
===dd This is the fifth.
```

When the ENTER key is pressed, the above lines are deleted.

The first and last lines of the block need not be on the same screen; you may scroll the screen before entering the second "DD". When one "DD" has been typed and the ENTER key pressed, the status area of the screen displays "BLOCK INCOMPLETE". You can use the PF7 or PF8 keys to scroll the screen until you find the last line of the block, and then type "DD" in its prefix area. When the ENTER key is pressed, the entire block of lines is deleted.

Figure 1-4 is a before-and-after example of the A and D prefix subcommands.

Lost and Found Department

If you delete one or more lines, you can recover them at any time during an editing session by using the RECOVER subcommand.

The following subcommand returns lines deleted in an editing session:

```
===> RECOVER n
```

where "n" represents the number of lines you wish to recover.

The recovered line(s) is inserted immediately before the current line. If the lines were deleted from different places in the file, you'll have to put them back where they belong (by using the M prefix subcommand, discussed below.)

If you want to recover *all* lines that were deleted during an editing session, use the form:

```
===> RECOVER *
```

Duplicating Lines

To duplicate a line, enter the character " (double quote) in the prefix area of a line.

A number may precede or follow the " to duplicate the line more than one time.

For example:

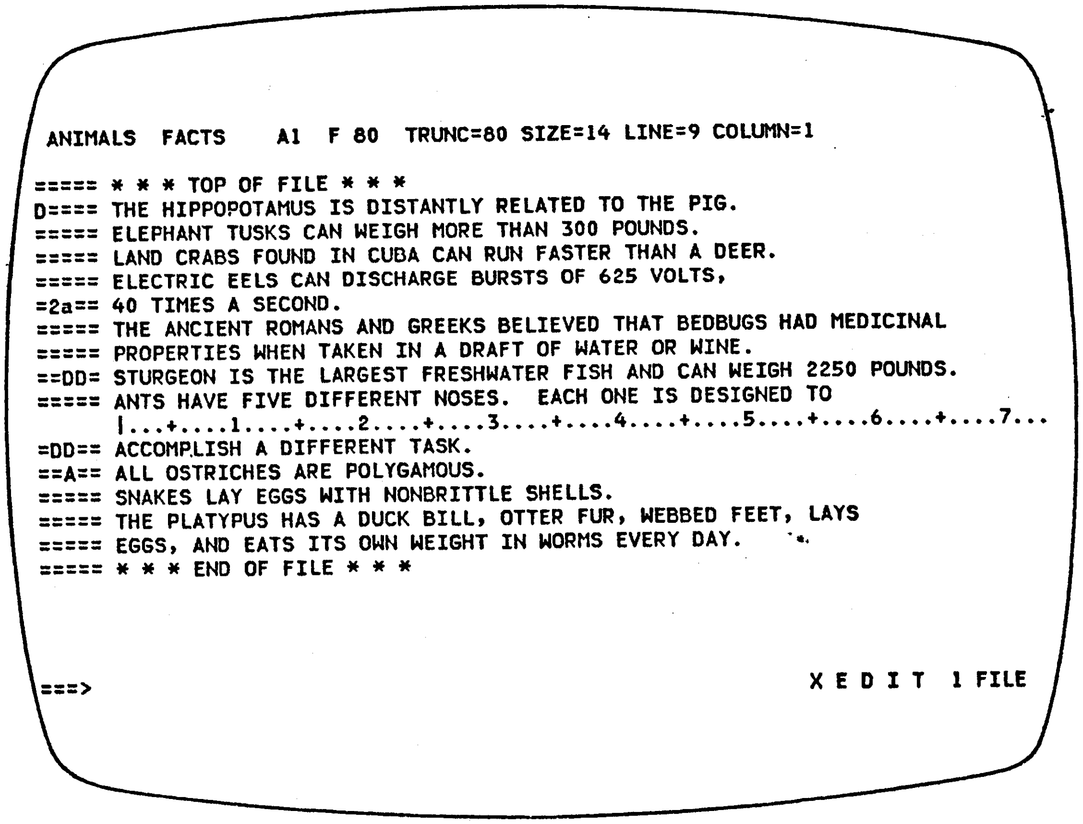
```
=3"== I want three more copies of this line.
===== Oh, yeah?
```

When the ENTER key is pressed, the file looks like this:

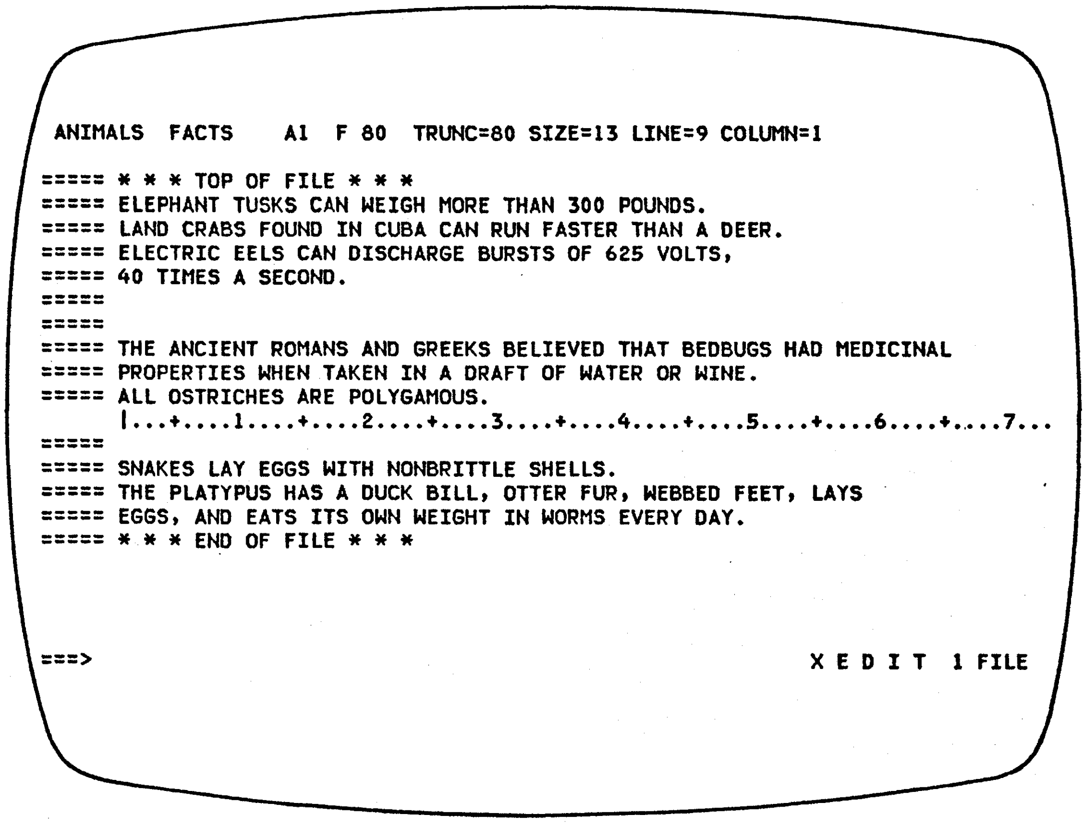
```
===== I want three more copies of this line.
===== I want three more copies of this line.
===== I want three more copies of this line.
===== I want three more copies of this line.
===== Oh, yeah?
```

To duplicate a block of lines either one time or a specified number of times, you can type "" (two double quotes) in the first and last lines of the block. A number can precede or follow the first "" (for example, 5"") to duplicate the block more than one time.

When one "" has been typed and the ENTER key pressed, the status area of the screen displays "BLOCK INCOMPLETE". This allows you to scroll the screen before completing the block and pressing the ENTER key.



```
ANIMALS  FACTS    A1  F 80  TRUNC=80 SIZE=14 LINE=9 COLUMN=1

===== * * * TOP OF FILE * * *
D==== THE HIPPOPOTAMUS IS DISTANTLY RELATED TO THE PIG.
===== ELEPHANT TUSKS CAN WEIGH MORE THAN 300 POUNDS.
===== LAND CRABS FOUND IN CUBA CAN RUN FASTER THAN A DEER.
===== ELECTRIC EELS CAN DISCHARGE BURSTS OF 625 VOLTS,
=2a== 40 TIMES A SECOND.
===== THE ANCIENT ROMANS AND GREEKS BELIEVED THAT BEDBUGS HAD MEDICINAL
===== PROPERTIES WHEN TAKEN IN A DRAFT OF WATER OR WINE.
==DD= STURGEON IS THE LARGEST FRESHWATER FISH AND CAN WEIGH 2250 POUNDS.
===== ANTS HAVE FIVE DIFFERENT NOSES.  EACH ONE IS DESIGNED TO
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
=DD== ACCOMPLISH A DIFFERENT TASK.
==A== ALL OSTRICHES ARE POLYGAMOUS.
===== SNAKES LAY EGGS WITH NONBRITTLE SHELLS.
===== THE PLATYPUS HAS A DUCK BILL, OTTER FUR, WEBBED FEET, LAYS
===== EGGS, AND EATS ITS OWN WEIGHT IN WORMS EVERY DAY.
===== * * * END OF FILE * * *



===>                                                      X E D I T  1 FILE
```



```
ANIMALS  FACTS    A1  F 80  TRUNC=80 SIZE=13 LINE=9 COLUMN=1

===== * * * TOP OF FILE * * *
===== ELEPHANT TUSKS CAN WEIGH MORE THAN 300 POUNDS.
===== LAND CRABS FOUND IN CUBA CAN RUN FASTER THAN A DEER.
===== ELECTRIC EELS CAN DISCHARGE BURSTS OF 625 VOLTS,
===== 40 TIMES A SECOND.
=====
=====
===== THE ANCIENT ROMANS AND GREEKS BELIEVED THAT BEDBUGS HAD MEDICINAL
===== PROPERTIES WHEN TAKEN IN A DRAFT OF WATER OR WINE.
===== ALL OSTRICHES ARE POLYGAMOUS.
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
=====
===== SNAKES LAY EGGS WITH NONBRITTLE SHELLS.
===== THE PLATYPUS HAS A DUCK BILL, OTTER FUR, WEBBED FEET, LAYS
===== EGGS, AND EATS ITS OWN WEIGHT IN WORMS EVERY DAY.
===== * * * END OF FILE * * *



===>                                                      X E D I T  1 FILE
```

Figure 1-4. Prefix Subcommands A and D - "Before" and "After".

Moving and Copying Lines

To move one line, enter the single character "M" in the prefix area of the line to be moved. You must indicate its destination by entering either the character "F" (following) or "P" (preceding) in the prefix area of another line.

When the ENTER key is pressed, the line containing the "M" is removed from its original location and is inserted in one of the following:

- immediately following the line containing the "F"
- immediately preceding the line containing the "P"

A number may precede or follow the "M" to indicate that more than one line is to be moved, for example, "5M" or "M5".

The line to be moved and the destination line can be on different screens. When either an "M" or "F" (or "P") has been entered, the status area of the screen displays "COPY/MOVE PENDING". This pending status allows you to scroll the screen before entering the other prefix subcommand.

To move a block of lines, enter the double character "MM" in the prefix area of both the first and last lines to be moved. The first and last lines to be moved, and the destination line may all be on different screens. You can use PF keys to scroll the screen before pressing the ENTER key.

The procedure for copying lines is the same as for moving lines, except that a "C" or "CC" prefix subcommand is used instead of "M" or "MM". The copy operation leaves the original line(s) in place, and makes a copy at the destination line, which must be indicated by "F" or "P".

Figure 1-5 is a before-and-after example of the M prefix subcommand.

Setting the Current Line (/)

Many subcommands begin their operations starting with the current line. For example, the INPUT subcommand makes room for you to enter data after the current line. You have already seen the INPUT subcommand used to insert lines after the TOP OF FILE line.

The / (diagonal) prefix subcommand can be typed in the prefix area of any line on the screen. When the ENTER key is pressed, that line becomes the current line. Then, if you enter an INPUT subcommand, the new lines entered in input mode will be inserted between the current line and the line that followed it.

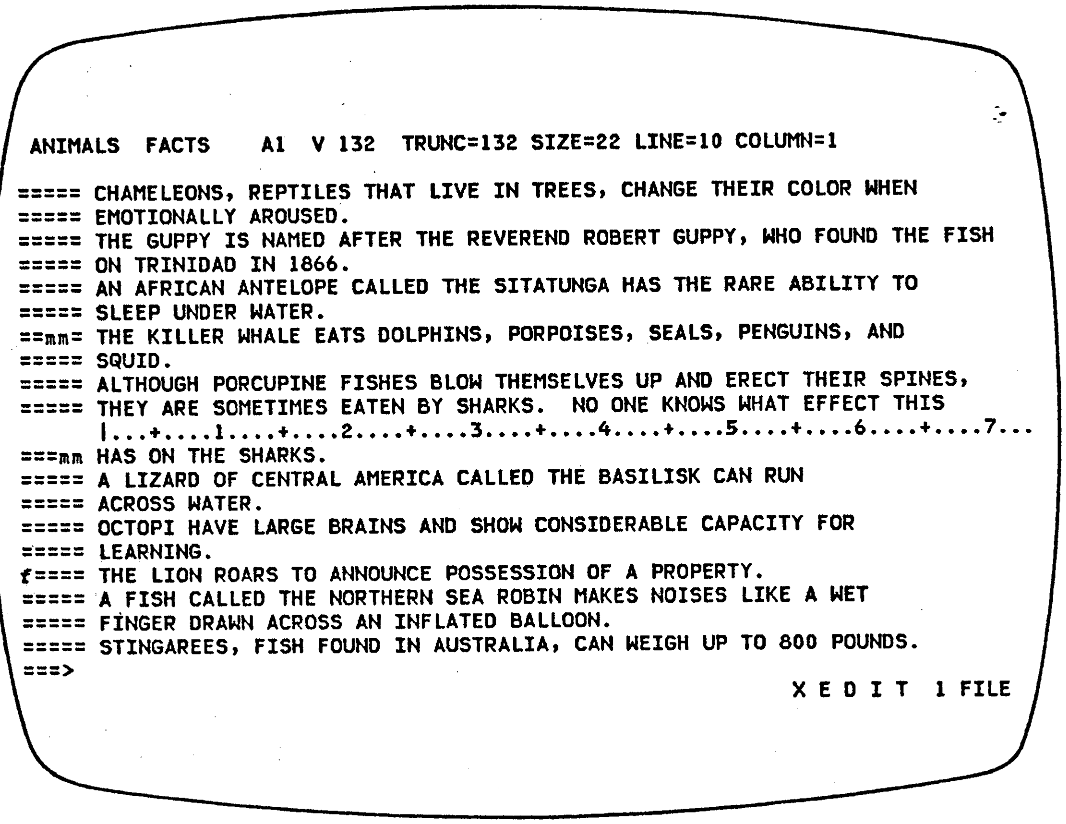
Canceling Prefix Subcommands

If you have entered one or more prefix subcommands that create a "BLOCK INCOMPLETE" or a "COPY/MOVE PENDING" status, you can cancel all these prefix subcommands by entering the following subcommand *in the command line:*

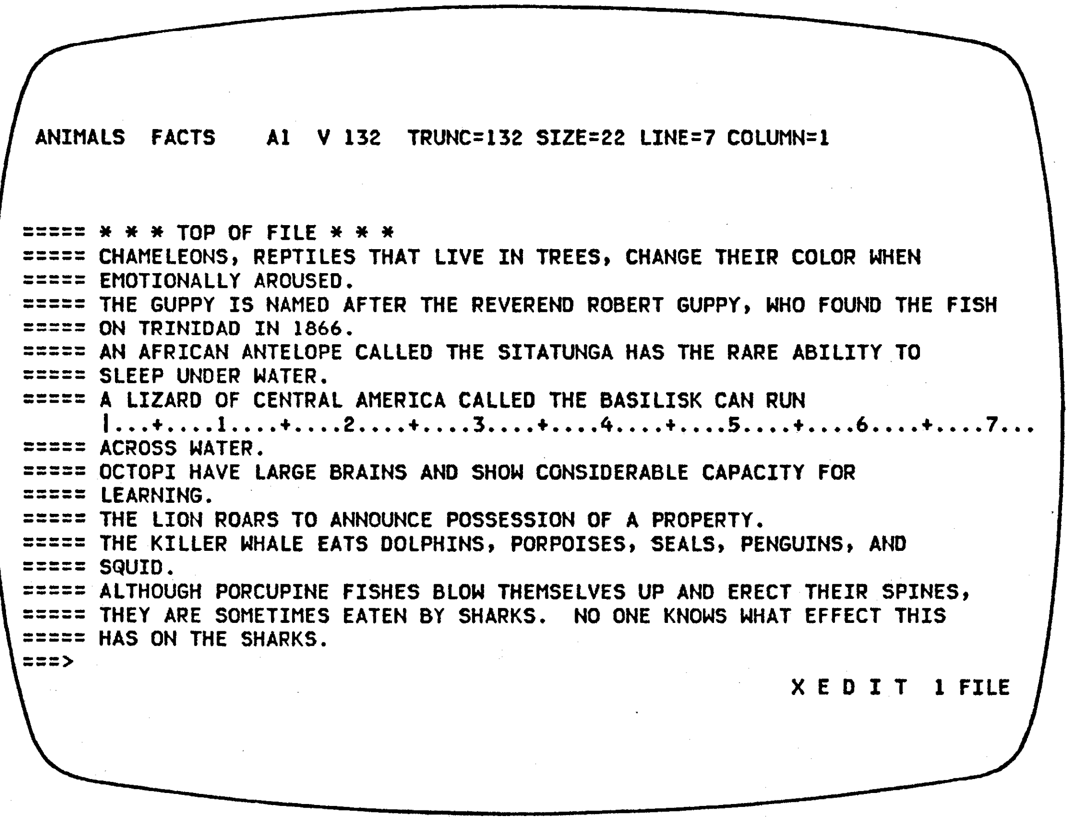
```
===> RESET
```

When the ENTER key is pressed, all prefix subcommands disappear from the display and the prefix areas are restored with equals signs (=====).

If you have typed any prefix subcommands (even those that do not cause an incomplete or pending status) but have not yet pressed the ENTER key, you can press the CLEAR key to remove them.



```
ANIMALS  FACTS    A1  V 132  TRUNC=132 SIZE=22 LINE=10 COLUMN=1

===== CHAMELEONS, REPTILES THAT LIVE IN TREES, CHANGE THEIR COLOR WHEN
===== EMOTIONALLY AROUSED.
===== THE GUPPY IS NAMED AFTER THE REVEREND ROBERT GUPPY, WHO FOUND THE FISH
===== ON TRINIDAD IN 1866.
===== AN AFRICAN ANTELOPE CALLED THE SITATUNGA HAS THE RARE ABILITY TO
===== SLEEP UNDER WATER.
==mm= THE KILLER WHALE EATS DOLPHINS, PORPOISES, SEALS, PENGUINS, AND
===== SQUID.
===== ALTHOUGH PORCUPINE FISHES BLOW THEMSELVES UP AND ERECT THEIR SPINES,
===== THEY ARE SOMETIMES EATEN BY SHARKS.  NO ONE KNOWS WHAT EFFECT THIS
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===mm HAS ON THE SHARKS.
===== A LIZARD OF CENTRAL AMERICA CALLED THE BASILISK CAN RUN
===== ACROSS WATER.
===== OCTOPI HAVE LARGE BRAINS AND SHOW CONSIDERABLE CAPACITY FOR
===== LEARNING.
f==== THE LION ROARS TO ANNOUNCE POSSESSION OF A PROPERTY.
===== A FISH CALLED THE NORTHERN SEA ROBIN MAKES NOISES LIKE A WET
===== FINGER DRAWN ACROSS AN INFLATED BALLOON.
===== STINGAREES, FISH FOUND IN AUSTRALIA, CAN WEIGH UP TO 800 POUNDS.
===>
                                                        X E D I T  1 FILE
```



```
ANIMALS  FACTS    A1  V 132  TRUNC=132 SIZE=22 LINE=7 COLUMN=1


===== * * * TOP OF FILE * * *
===== CHAMELEONS, REPTILES THAT LIVE IN TREES, CHANGE THEIR COLOR WHEN
===== EMOTIONALLY AROUSED.
===== THE GUPPY IS NAMED AFTER THE REVEREND ROBERT GUPPY, WHO FOUND THE FISH
===== ON TRINIDAD IN 1866.
===== AN AFRICAN ANTELOPE CALLED THE SITATUNGA HAS THE RARE ABILITY TO
===== SLEEP UNDER WATER.
===== A LIZARD OF CENTRAL AMERICA CALLED THE BASILISK CAN RUN
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== ACROSS WATER.
===== OCTOPI HAVE LARGE BRAINS AND SHOW CONSIDERABLE CAPACITY FOR
===== LEARNING.
===== THE LION ROARS TO ANNOUNCE POSSESSION OF A PROPERTY.
===== THE KILLER WHALE EATS DOLPHINS, PORPOISES, SEALS, PENGUINS, AND
===== SQUID.
===== ALTHOUGH PORCUPINE FISHES BLOW THEMSELVES UP AND ERECT THEIR SPINES,
===== THEY ARE SOMETIMES EATEN BY SHARKS.  NO ONE KNOWS WHAT EFFECT THIS
===== HAS ON THE SHARKS.
===>
                                                        X E D I T  1 FILE
```

Figure 1-5. Prefix Subcommands M and F - "Before" and "After"

Moving Through a File

The following subcommands are discussed in this section:

BACKWARD
FORWARD
TOP
BOTTOM
UP
DOWN

BACKWARD and FORWARD Subcommands

Scrolling the screen is like turning the pages of a book. You have already seen that the PF7 and PF8 keys are set to the BACKWARD and FORWARD subcommands, which scroll one full screen backward or forward. The BACKWARD and FORWARD subcommands can also be entered in the command line.

The format of these subcommands is:

```
===> BACKWARD n
===> FORWARD  n
```

where "n" is the number of screen displays you want to scroll backward or forward. (This is like pressing the PF7 or PF8 key "n" times.) If you omit "n", the editor scrolls one screen backward or forward.

If you enter a BACKWARD subcommand when the current line is the "TOP OF FILE" line, the editor "wraps around" the file, making the last line of the file the new current line. Similarly, if you enter a FORWARD subcommand when the current line is the "END OF FILE" line, the editor makes the first line of the file the new current line.

TOP and BOTTOM Subcommands

Suppose the file is many screens long, and the current screen display is somewhere in the middle of the file. To go back to the beginning of the file, you could enter multiple BACKWARD subcommands - or - you could enter the TOP subcommand. The TOP subcommand makes the "TOP OF FILE" line the new current line. Its format is:

```
===> TOP
```

The BOTTOM subcommand makes the last line of the file the new current line. Its format is:

```
===> BOTTOM
```

These subcommands are useful when you want to insert new lines either at the beginning or end of a file. The TOP subcommand followed by an INPUT or POWER subcommand makes room for you to add lines at the beginning of a file; use the BOTTOM subcommand followed by INPUT or POWER to add lines to the end of a file.

DOWN and UP Subcommands

Suppose that you want to move the file up or down a few *lines* instead of a whole screen. The DOWN subcommand advances the line pointer one or more lines toward the *end* of a file. The line pointed to becomes the new current line. For example:

```
===> DOWN 5
```

makes the fifth line down from the current line the new current line. If the number is omitted, "1" is assumed.

The UP subcommand moves the line pointer toward the *beginning* of the file. The line pointed to becomes the new current line. For example:

```
===> UP 5
```

makes the fifth line up from the current line the new current line. If a number is omitted, "1" is assumed.

Figure 1-6 is a before-and-after example of the DOWN subcommand.

Making Changes in a File

When you're looking at a screen of data that you have just entered and decide to make some changes, it's easy to type over the information to be changed.

However, it's not always that simple. Typically, you have numerous files stored on direct access devices and need to make changes even though you don't know exactly where the data is located in a file.

The challenge is two-fold: find the data; then change it.

The following subcommands are discussed in this section:

CLOCATE
CHANGE
CINSERT
CFIRST

CLOCATE Subcommand

The CLOCATE subcommand searches a file, beginning with the current column in the current line, for a character string that you specify.

If the string is located, two things happen:

- The line containing the string becomes the new current line; however, if the string is in the current line, the line pointer does not move.
- The column pointer, represented in the scale as a vertical bar (|), moves under the first character of the string.

These changes are reflected in the file identification area at the top of the screen (LINE=nnn and COLUMN=nn).

One format of the CLOCATE subcommand is as follows:

```
===> CLOCATE/string/
```

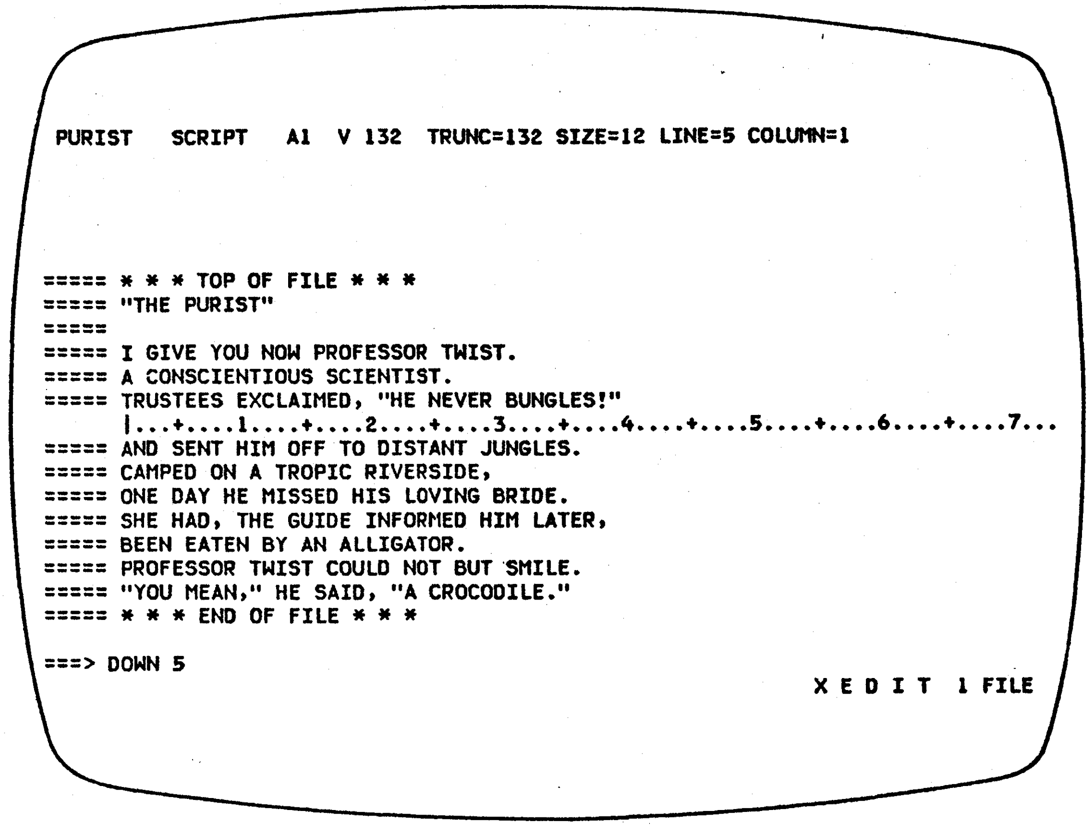
The string must be enclosed in delimiters. In the examples used in this book, the delimiter is a diagonal (/); however, you can use any character that does not appear in the string itself (for example, CLOCATE.VM/CMS.).

In the following example, the string to be located is in the current line. Therefore, the line pointer does not move, but look what happens to the column pointer:

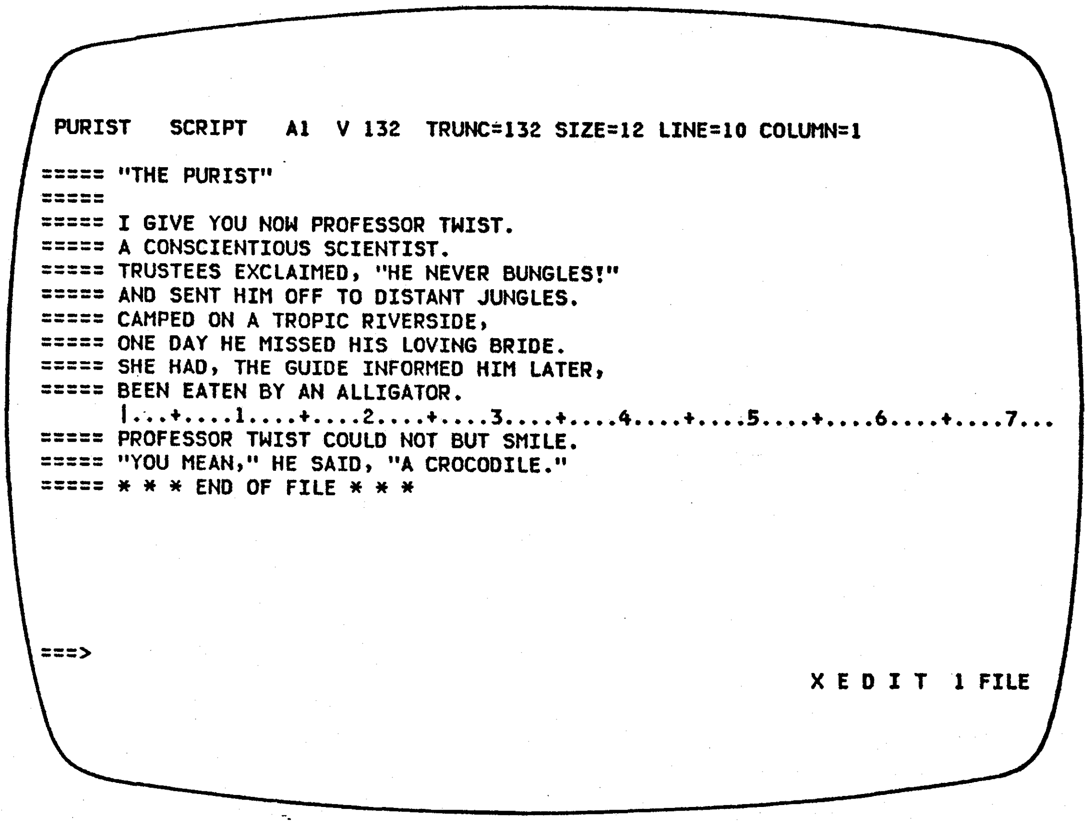
```
===== To be or not to be - that is the question.
      |...+....1....+....2....+....3....+....4....+....5....+....6....

===> CLOCATE/be/

===== To be or not to be - that is the question.
      <..|+....1....+....2....+....3....+....4....+....5....+....6....
```

Notice that the column pointer in the scale has moved under the first character (b) in the string (be).

If you wanted to find all occurrences of "be" throughout the file, you could enter the CLOCATE/be/ subcommand repeatedly (or use the PF9 key, which is set to the = subcommand, for repeated execution). If a string appears more than once in a



```
PURIST   SCRIPT   A1  V 132  TRUNC=132 SIZE=12 LINE=5 COLUMN=1




===== * * * TOP OF FILE * * *
===== "THE PURIST"
=====
===== I GIVE YOU NOW PROFESSOR TWIST.
===== A CONSCIENTIOUS SCIENTIST.
===== TRUSTEES EXCLAIMED, "HE NEVER BUNGLES!"
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== AND SENT HIM OFF TO DISTANT JUNGLES.
===== CAMPED ON A TROPIC RIVERSIDE,
===== ONE DAY HE MISSED HIS LOVING BRIDE.
===== SHE HAD, THE GUIDE INFORMED HIM LATER,
===== BEEN EATEN BY AN ALLIGATOR.
===== PROFESSOR TWIST COULD NOT BUT SMILE.
===== "YOU MEAN," HE SAID, "A CROCODILE."
===== * * * END OF FILE * * *

===> DOWN 5
                                                          X E D I T  1 FILE
```



```
PURIST   SCRIPT   A1  V 132  TRUNC=132 SIZE=12 LINE=10 COLUMN=1
===== "THE PURIST"
=====
===== I GIVE YOU NOW PROFESSOR TWIST.
===== A CONSCIENTIOUS SCIENTIST.
===== TRUSTEES EXCLAIMED, "HE NEVER BUNGLES!"
===== AND SENT HIM OFF TO DISTANT JUNGLES.
===== CAMPED ON A TROPIC RIVERSIDE,
===== ONE DAY HE MISSED HIS LOVING BRIDE.
===== SHE HAD, THE GUIDE INFORMED HIM LATER,
===== BEEN EATEN BY AN ALLIGATOR.
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== PROFESSOR TWIST COULD NOT BUT SMILE.
===== "YOU MEAN," HE SAID, "A CROCODILE."
===== * * * END OF FILE * * *





===>
                                                          X E D I T  1 FILE
```

Figure 1-6. The DOWN Subcommand - "Before" and "After"

line, as in the example above, the line pointer remains the same, but the column pointer moves under the next occurrence of the string.

For example, if the CLOCATE/be/ subcommand is entered again, the line looks like this:

```
===== To be or not to be - that is the question.
      <...+....1....+.|..2....+....3....+....4....+....5....+....6....
```

Note the position of the column pointer, under the second "be".

Each time the CLOCATE/be/ subcommand is entered, the column pointer moves under the next occurrence of "be"; in addition, the line pointer advances, until all occurrences of "be" have been found.

If the string that you're searching for is in a *backward* direction from the current line, toward the top of the file, you can tell the editor to search backward by typing a minus sign (-) in front of the string. For example:

```
===> CLOCATE -/glance/
```

is a backward search for "glance".

CHANGE Subcommand

Replacing one word with another is the simplest type of change. If the string you want to change is not in the current line, you can use the CLOCATE subcommand to move the line pointer to the line that contains the string. Then, you can use the following form of the CHANGE subcommand, which changes the first occurrence of a word in the current line:

```
===> CHANGE/oldword/newword/
```

For example:

```
===== A rose is a rose is a rose.
      |...+....1....+....2....+....3....+...
===> CHANGE/rose/daisy/
```

(with apologies to Gertrude Stein)

```
===== A daisy is a rose is a rose.
      |...+....1....+....2....+....3....+...
```

Note that the editor automatically made room in the line for "daisy" even though it is longer than "rose". Conversely, a word can be replaced by a shorter word; the editor removes extra blanks.

You can use the CLOCATE and CHANGE subcommands to locate and change any string in a file. If the line containing the string is the current line, you don't have to use a CLOCATE subcommand; the CHANGE subcommand both locates and changes it.

Making a Selective Change

Suppose you want to change one word to another only *some* of the time, that is, you want to make a selective, or "safe" change. You can do this by locating (repeatedly) the string you want to change, and by entering a CHANGE subcommand only when you want to change the string. However, there's an easier way.

All you have to do is type a CHANGE subcommand (in the form CHANGE/oldword/newword) in the command line. Then, use the PF5 key to locate each occurrence of the old word, examine it, and then either change it (by pressing the PF6 key), or go on to the next occurrence (by pressing the PF5 key).

Here's how to make a selective change:

1. Move the line pointer to the line where you want the search to begin. (You can use TOP, /, DOWN, or UP.)
2. Type a CHANGE subcommand (CHANGE/oldword/newword) in the command line, but *don't press the ENTER key.*
3. Press the PF5 key. The cursor moves under the first occurrence of the old word, and the line that contains it is highlighted.
4. If you want to change the word, press the PF6 key. If not, press the PF5 key again, and step number 3 (above) will be repeated.

Using this sequence, you can locate all the occurrences of the old word, and press the PF6 key to change it only when desired. When all occurrences of the old word on one screen have been located, the editor scrolls the screen forward automatically.

Figure 1-7 is an example of using the PF5 and PF6 keys to locate and change selectively a character string throughout a file. The following subcommand was typed in the command line but the ENTER key was not pressed:

```
===> CHANGE/rose/daisy/
```

This subcommand is executed when the PF6 key is pressed.

In the top screen, pressing the PF5 key has placed the cursor (and the column pointer) under the first occurrence of "rose".

In the bottom screen, the PF5 key was pressed successively until the last occurrence of "rose". Then the PF6 key was pressed to execute the change specified in the command line.

If you want to locate all occurrences of a string, but you don't want to make any changes, you can type a CLOCATE/string/ subcommand instead of a CHANGE subcommand. Then, each time you press the PF5 key, the cursor moves under the next occurrence of the string and the line is highlighted. Pressing the PF6 key has no effect.

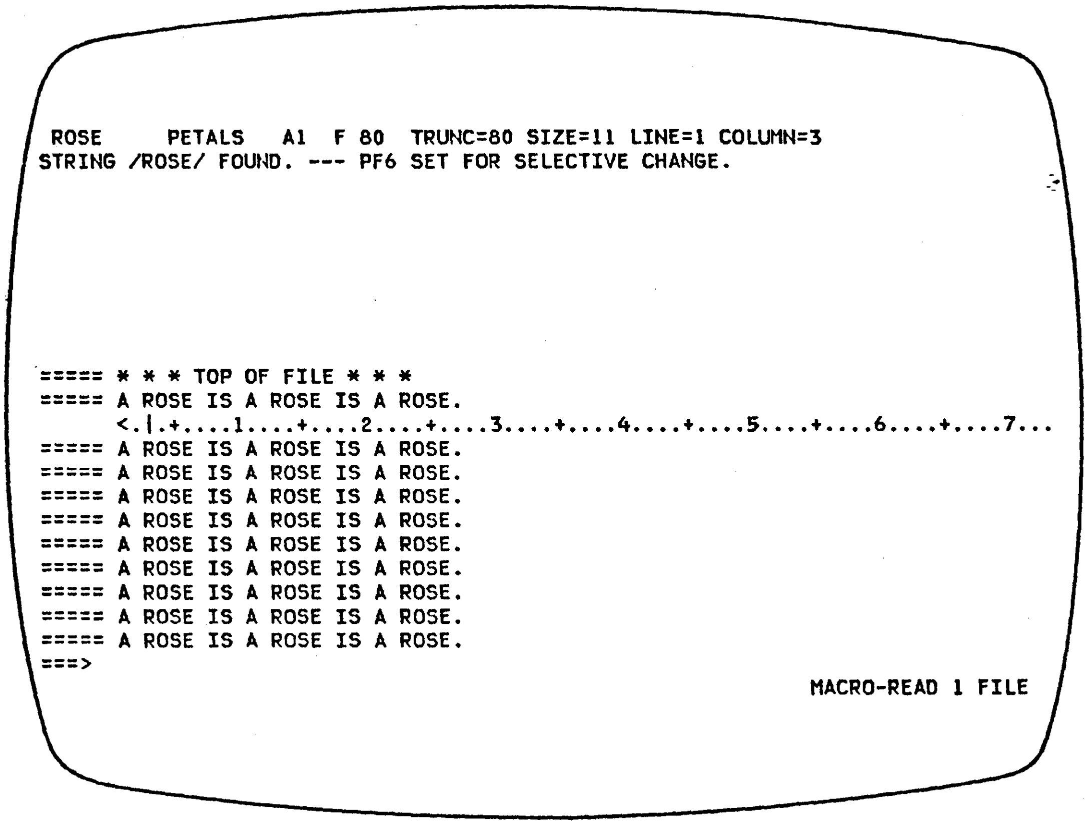
Making a Global Change

If you want to make a global change, that is, change *every occurrence* of a word *throughout the file,* first make sure that the first line of the file is the current line (via the TOP subcommand) and use the following form of the CHANGE subcommand:

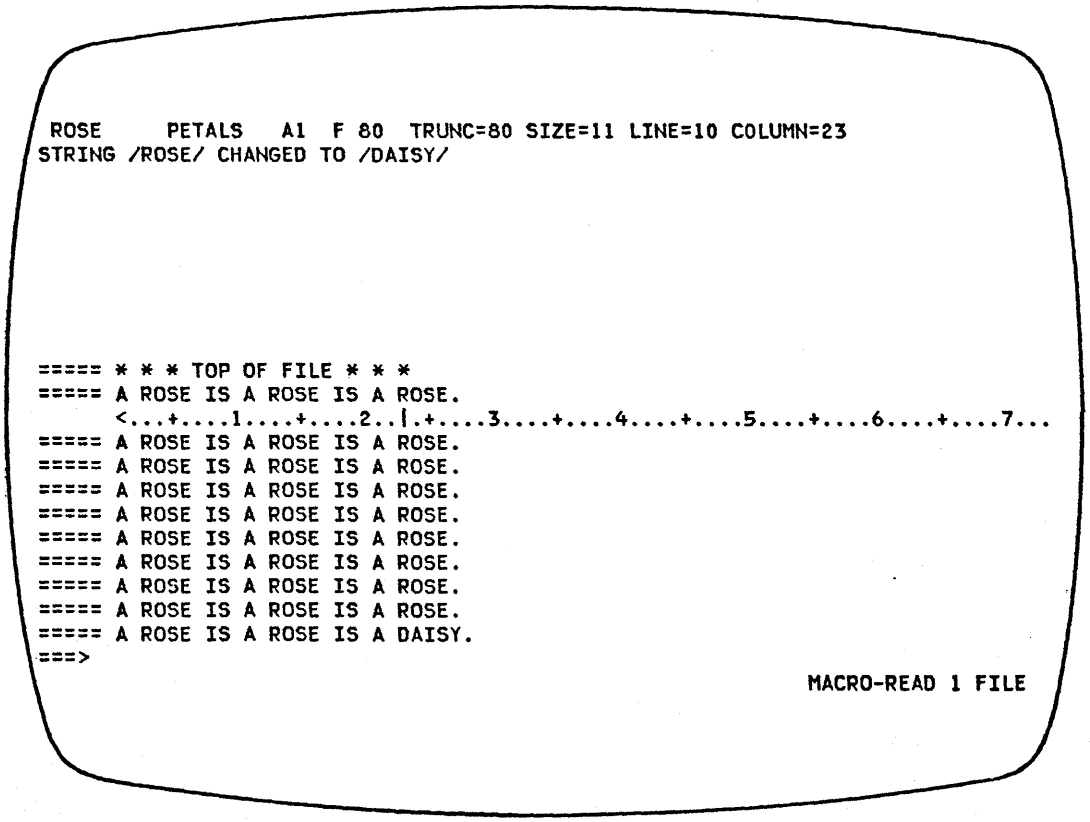
```
===> CHANGE/oldword/newword/ * *
```

For example:

```
===== * * * TOP OF FILE * * *
===== A rose is a rose is a rose.
===== A rose is a rose is a rose.
===== A rose is a rose is a rose.
===== A rose is a rose is a rose.
===== * * * END OF FILE * * *

===> CHANGE/rose/daisy/ * *

===== * * * TOP OF FILE * * *
===== A daisy is a daisy is a daisy.
===== A daisy is a daisy is a daisy.
===== A daisy is a daisy is a daisy.
===== A daisy is a daisy is a daisy.
===== * * * END OF FILE * * *
```



```
 ROSE     PETALS   A1  F 80  TRUNC=80 SIZE=11 LINE=1 COLUMN=3
STRING /ROSE/ FOUND. --- PF6 SET FOR SELECTIVE CHANGE.







===== * * * TOP OF FILE * * *
===== A ROSE IS A ROSE IS A ROSE.
      <.|.+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== A ROSE IS A ROSE IS A ROSE.
===== A ROSE IS A ROSE IS A ROSE.
===== A ROSE IS A ROSE IS A ROSE.
===== A ROSE IS A ROSE IS A ROSE.
===== A ROSE IS A ROSE IS A ROSE.
===== A ROSE IS A ROSE IS A ROSE.
===== A ROSE IS A ROSE IS A ROSE.
===== A ROSE IS A ROSE IS A ROSE.
===== A ROSE IS A ROSE IS A ROSE.
===>
                                                           MACRO-READ 1 FILE
```



```
 ROSE     PETALS   A1  F 80  TRUNC=80 SIZE=11 LINE=10 COLUMN=23
STRING /ROSE/ CHANGED TO /DAISY/







===== * * * TOP OF FILE * * *
===== A ROSE IS A ROSE IS A ROSE.
      <...+....1....+....2..|.+....3....+....4....+....5....+....6....+....7...
===== A ROSE IS A ROSE IS A ROSE.
===== A ROSE IS A ROSE IS A ROSE.
===== A ROSE IS A ROSE IS A ROSE.
===== A ROSE IS A ROSE IS A ROSE.
===== A ROSE IS A ROSE IS A ROSE.
===== A ROSE IS A ROSE IS A ROSE.
===== A ROSE IS A ROSE IS A ROSE.
===== A ROSE IS A ROSE IS A ROSE.
===== A ROSE IS A ROSE IS A DAISY.
===>
                                                           MACRO-READ 1 FILE
```

Figure 1-7. Using PF5 and PF6 to Make a Selective Change

This form of the CHANGE subcommand can also be used to make a global change starting in the middle of a file. Since the change starts with the current line, you would just make current (via /) that line where you want the change to begin.

Another variation of the CHANGE subcommand can be used if you want to change a word throughout the file, but you want to change only the first occurrence in each line:

```
===> CHANGE/oldword/newword/ *
```

CINSERT Subcommand

Often, you need to insert words in a line. You have already seen how to use the PA2 and insert mode keys and the SET NULLS subcommand. Another way to insert words is by using the CINSERT subcommand, which allows you to insert characters in the current line *immediately before the column pointer.*

You can use a CLOCATE/string/ subcommand to move the column pointer to the desired position. You can also use another form of the CLOCATE subcommand to move the column pointer:

```
===> CLOCATE :n
```

where ":n" represents an absolute column number, easily determined by looking at the scale.

For example:

```
===== To be or not to be - that is the question.
      |...+....1....+....2....+....3....+....4....+....5....+....6....
===> CLOCATE :4
===== To be or not to be - that is the question.
      <..|+....1....+....2....+....3....+....4....+....5....+....6....
```

The column pointer has moved to column four.

In the following example, the CLOCATE subcommand is used to move the column pointer; then the CINSERT subcommand is used to insert characters immediately before the column pointer position.

```
===== If anything can go, it will.
      |...+....1....+....2....+....3....+....4....+....5....+....6....
===> CLOCATE/,/     or     ===> CLOCATE :19
```

(move the column pointer)

```
===== If anything can go, it will.
      <...+....1....+...|2....+....3....+....4....+....5....+....6....
===> CINSERT  wrong
```

(insert "wrong" before the column pointer)

```
===== If anything can go wrong, it will.
      <...+....1....+...|2....+....3....+....4....+....5....+....6....
```

(In the CINSERT subcommand above, note that there are *two spaces* between "CINSERT" and "wrong": one is the required space between the subcommand name and the operand; one is the blank space needed between "go" and "wrong".)

If only one blank space were used, the result would be the following:

```
===== If anything can gowrong, it will.
```

The editor allows you to insert blanks with the CINSERT subcommand - simply type the required number of blanks (by pressing the space bar) in the operand. For example:

```
===== If anything can go wrong, it will.

===> CLOCATE/can/
===> CINSERT
```

(Press the space bar six times.)

```
===== If anything       can go wrong, it will.
```

If the inserted characters make the line longer than the screen line, the editor automatically "wraps around" to the next line. Characters can be inserted up to the truncation column, as shown in the following example:

```
===== It takes less time to do a thing than to explain why you did it.
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...

===> CLOCATE/than/
```

(move the column pointer)

```
===== It takes less time to do a thing than to explain why you did it.
      <...+....1....+....2....+....3...|+....4....+....5....+....6....+....7...

===> CINSERT right
```

(insert the first word)

```
===== It takes less time to do a thing right than to explain why you did it.
      <...+....1....+....2....+....3...|+....4....+....5....+....6....+....7...

===> CLOCATE/./
```

(move the column pointer again)

```
===== It takes less time to do a thing right than to explain why you did it.
      <...+....1....+....2....+....3....+....4....+....5....+....6....+....|...

===> CINSERT  wrong
```

(insert the second word)

```
===== It takes less time to do a thing right than to explain why you did it wron
g.
```

Even though the resulting line is longer than a screen line, it is considered to be one logical line.

Notice that the line has one prefix area associated with it. Any prefix subcommands entered in the prefix area affect the entire logical line. For example, if a D prefix subcommand is entered, the whole sentence is deleted.

CFIRST Subcommand

After using subcommands that move the column pointer, it's a good idea to reset the column pointer to column one by issuing the CFIRST subcommand.

For example:

```
===== If anything can go wrong, it will.
      <...+....1....+....2...|+....3....+....4....+....5....+....6....

===> CFIRST

===== If anything can go wrong, it will.
      |...+....1....+....2....+....3....+....4....+....5....+....6....
```

Setting Tabs

Sometimes you may want to place information in specific columns. The PF4 key functions like a tab key on a typewriter. Each time the PF4 key is pressed, the cursor is positioned under the next tab column, where you can enter data.

Initial tab settings are defined by the editor according to filetype; they may be displayed by using the following subcommand:

```
===> QUERY TABS
```

You can change these settings one or more times during an editing session with the SET TABS subcommand. For example:

```
===> SET TABS 10 20 30
```

The first time the PF4 key is pressed, the cursor moves to column 10 on the screen. The second time, it moves to column 20, and so forth.

The PF4 key may be used for tabbing in input mode, but not in power typing mode.

You can change the tab settings by issuing another SET TABS subcommand, or, if you'd like to see the current tab settings before changing them, you can use the following subcommand:

```
===> MODIFY TABS
```

The current SET TABS subcommand is then displayed in the command line; you can type over the numbers and press the ENTER key to define new tabs.

Figure 1-8 is an example of data that was entered using the PF4 key as a tab key. The following subcommand was used to define the tab columns:

```
===> SET TABS  5 35 45
```

Ending an Editing Session

The following subcommands are discussed in this section:

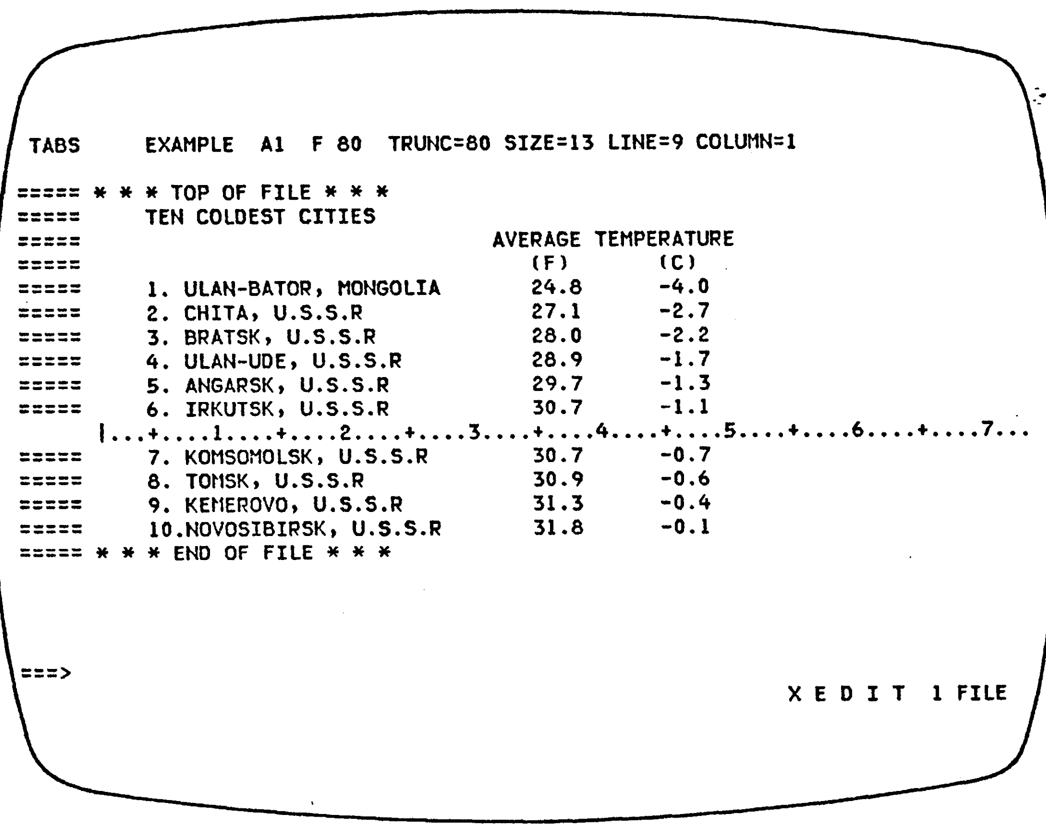
FILE
QUIT
SET AUTOSAVE

FILE Subcommand

When you use the XEDIT command to create a new file, the file is created in virtual storage. When you make changes to an existing file, those changes are made to a copy of the file that is brought into virtual storage (when the XEDIT command is entered). However, virtual storage is *temporary.* To write a new or modified file on disk, which is *permanent* storage, you must enter the following subcommand:

```
===> FILE
```

When the FILE subcommand is executed, the file is written on disk and control is returned to CMS.



```
TABS      EXAMPLE  A1  F 80  TRUNC=80 SIZE=13 LINE=9 COLUMN=1

===== * * * TOP OF FILE * * *
=====      TEN COLDEST CITIES
=====                                    AVERAGE TEMPERATURE
=====                                       (F)        (C)
=====      1. ULAN-BATOR, MONGOLIA          24.8       -4.0
=====      2. CHITA, U.S.S.R                27.1       -2.7
=====      3. BRATSK, U.S.S.R               28.0       -2.2
=====      4. ULAN-UDE, U.S.S.R             28.9       -1.7
=====      5. ANGARSK, U.S.S.R              29.7       -1.3
=====      6. IRKUTSK, U.S.S.R              30.7       -1.1
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
=====      7. KOMSOMOLSK, U.S.S.R           30.7       -0.7
=====      8. TOMSK, U.S.S.R                30.9       -0.6
=====      9. KEMEROVO, U.S.S.R             31.3       -0.4
=====      10.NOVOSIBIRSK, U.S.S.R          31.8       -0.1
===== * * * END OF FILE * * *



===>
                                                    X E D I T  1 FILE
```

Figure 1-8. Using the PF4 Key for Tabbing

QUIT Subcommand

Use the QUIT subcommand to end an editing session and leave the permanent copy of the file intact on the disk. If the file is new, it is not written on disk.

You can execute the QUIT subcommand either by pressing the PF3 key or by entering it on the command line, like this:

```
===> QUIT
```

You would use the QUIT subcommand instead of the FILE subcommand when you edit a file merely to examine, but not to change, its contents, or if you discover you have made errors in changing a file and do not want them to be recorded.

If a file is new or has been changed, the editor gives you a warning message to prevent the inadvertent use of a QUIT instead of a FILE. The message is as follows:

```
FILE HAS BEEN CHANGED.  USE QQUIT TO QUIT ANYWAY.
```

If you really don't want to save the file, enter "QQUIT" (abbreviated as "QQ"). If you wish to save the changes, enter "FILE".

SET AUTOSAVE Subcommand

Files on disk are not affected if the system malfunctions, or "goes down." However, a new file that you're creating or the changes you're making to an existing file might be lost if the system fails. You can minimize this danger by using the SET AUTOSAVE subcommand, whose format is as follows:

```
===> SET AUTOSAVE n
```

The SET AUTOSAVE subcommand causes your file to be written to disk automatically, after you've typed in or changed a certain number of lines. You specify what that number will be with the "n" operand of the SET AUTOSAVE subcommand. If you want the file written to disk, or "saved", every time you've changed ten lines, enter the following subcommand:

```
===> SET AUTOSAVE 10
```

The SET AUTOSAVE subcommand can be issued at any time during an editing session. It's a good idea, however, to issue the subcommand right after you issue an XEDIT command to create a new file or to call an existing file from disk.

If you have issued a SET AUTOSAVE subcommand and the system goes down, your file is written to disk with a new fileid. The filename is a number, and the filetype is AUTOSAVE. You can change the fileid back to its original filename and filetype by issuing the CMS command ERASE to erase the original file and then by issuing the CMS command RENAME.

For example, if your AUTOSAVE file is labeled "1 AUTOSAVE A1" and the original file is "INVENTOR SCRIPT A1", use the following CMS commands to rename it:

```
ERASE INVENTOR SCRIPT
RENAME 1 AUTOSAVE A1 INVENTOR SCRIPT A1
```

Then you'll be back in business and can use the XEDIT command to start editing the file again.

A QUIT subcommand cancels a SET AUTOSAVE subcommand. If you issue a SET AUTOSAVE subcommand while you're creating a new file, and then issue a QUIT subcommand, the file is not saved. However, the AUTOSAVE file is available on disk. If you issue a SET AUTOSAVE subcommand while you're revising an existing file and then you issue a QUIT subcommand, no revisions are saved.

Inserting Data From Another File

To insert all or part of one file into another file, you can use the GET subcommand. The chapters in this book were created as separate files and then combined into one file by using the GET subcommand.

The GET subcommand inserts another file after the current line in the file you are editing. Therefore, you must move the line pointer to the desired line of the file. For example, if you want to insert another file at the *end* of a file, you can use the BOTTOM subcommand. If you want to insert another file in the *middle* of a file, you can use the / prefix subcommand to make the desired line current.

Inserting a Whole File

Suppose you were writing a cookbook, and you created a separate file for each recipe. To combine two of the recipes into one file, you would use the following form of the GET subcommand:

```
===> GET filename filetype
```

Figure 1-9 shows how the GET subcommand is used to insert one whole file at the end of another file:

The top screen shows a file (DESSERT COOKBOOK) that contains a recipe for cream puffs. A recipe for almond cookies is contained in another file, COOKIES COOKBOOK.

The following subcommand was entered:

```
===> GET COOKIES COOKBOOK
```

In the bottom screen, the message "EOF REACHED" indicates that the entire file has been inserted. Notice that the last line inserted becomes the new current line. The file DESSERT COOKBOOK now contains two recipes. The file COOKIES COOKBOOK is left intact.

Inserting Part of Another File

To insert part of another file, you can specify in the GET subcommand the line number of the first line and the number of lines you want to insert. The following GET subcommand inserts the first ten lines of a second file:

```
===> GET FILE2 1 10
```

If you don't know the line numbers, you can: call out a second file without ending your current editing session; put the lines you want to insert into a temporary file; and insert them into your current file.

This might sound complicated, but all you need to learn is one more subcommand - PUT.

First, let's identify and explain the steps you would take to insert part of another file and then illustrate them with an example.

1. While editing the first file, enter an XEDIT subcommand to call out the second file. (You do not have to end your current editing session, because the editor allows you to edit multiple files simultaneously.) The second file will appear on the screen.

2. Use the PUT subcommand to indicate which lines are to be inserted in the first file. The PUT subcommand stores lines in a temporary holding area, *starting with the current line,* up to an ending, or target, line. Its format is as follows:

   ```
   ===> PUT target
   ```

   where "target" identifies the end of the group of lines to be inserted. It is a signal to the editor to stop "putting" lines.

   A target operand may be specified in various ways, which are described in detail in "Chapter 4. Using Targets". A brief description of three ways to specify a target follows. They are all equivalent; you can choose whichever type you prefer.

   One way to specify the target is to count the number of lines you want to insert, starting with the current line. For example, if a file contains:

   ```
   ===== a loaf of bread
   ===== a jug of wine
   ===== thou
   ===== a portable television
   ```

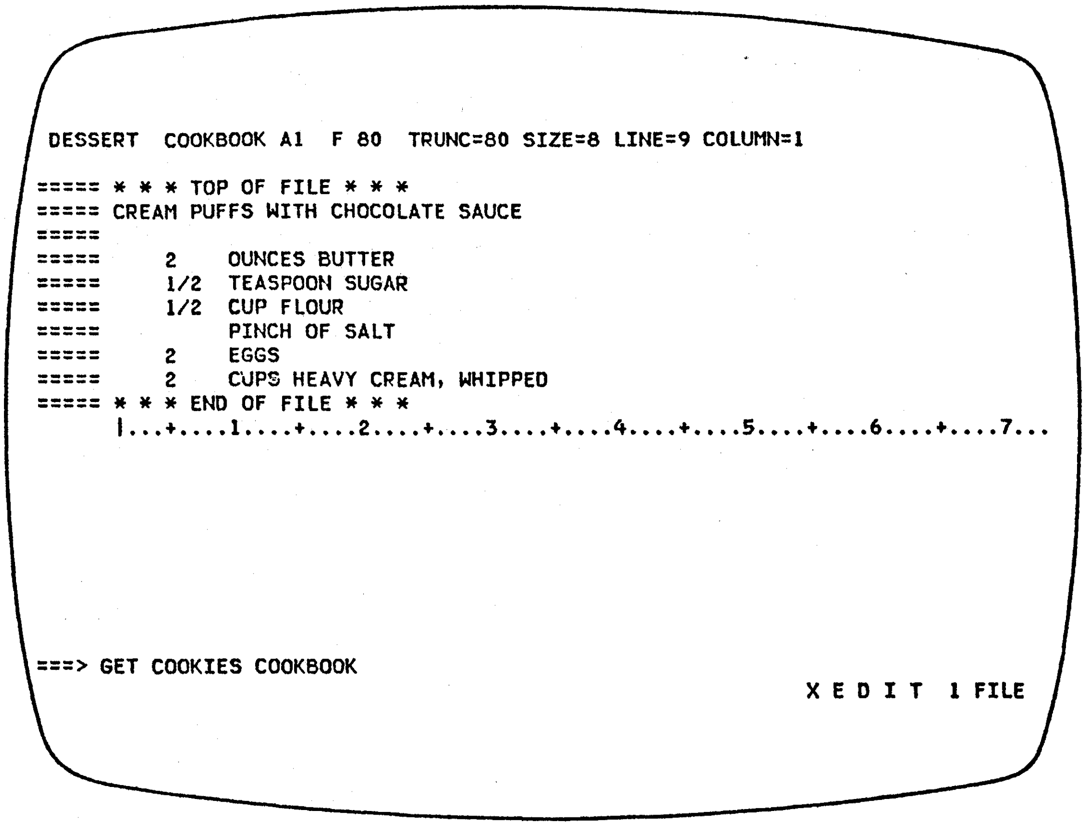
   and the line containing "a loaf of bread" is current, the following subcommand stores all the above lines:

   ```
   ===> PUT 4
   ```

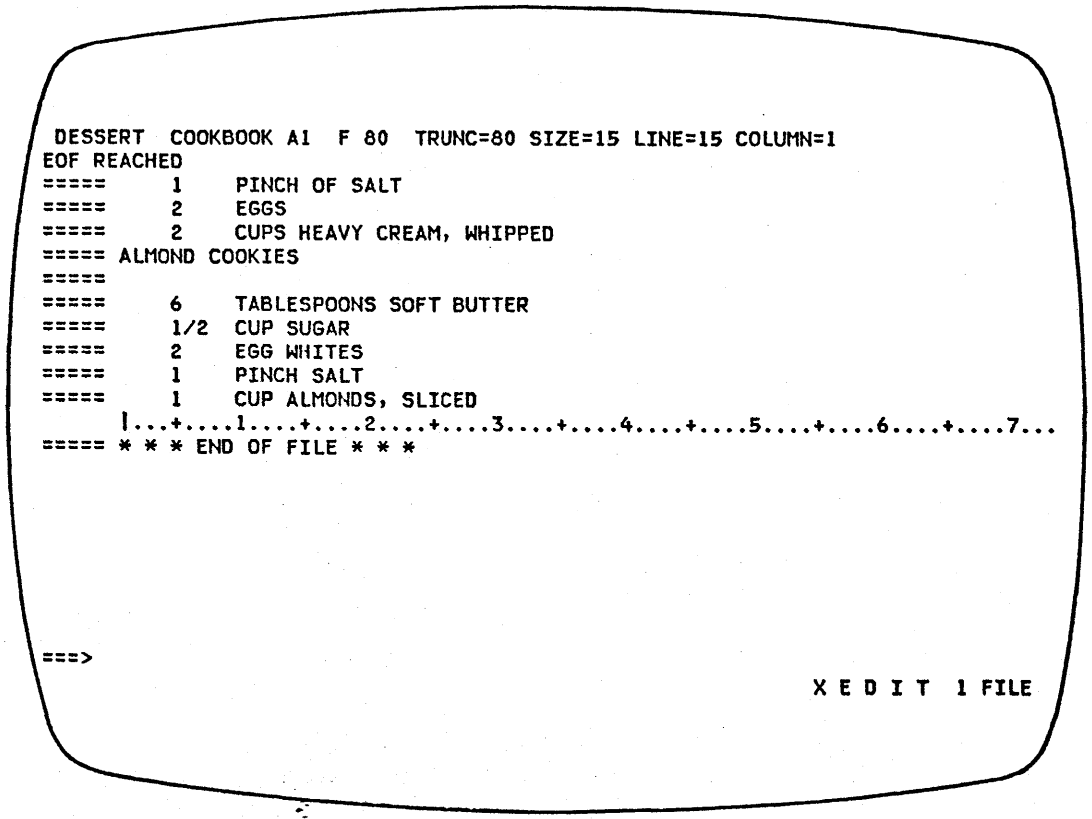
   Another way to specify the target is with a character string; the editor will "put" all the lines, beginning with the current line, up to, but not including, the line containing the string.

   For example, the following subcommand will "put" the first three lines, but it will not "put" the line containing "a portable television".

   ```
   ===> PUT/television/
   ```



```
 DESSERT  COOKBOOK A1  F 80  TRUNC=80 SIZE=8 LINE=9 COLUMN=1

===== * * * TOP OF FILE * * *
===== CREAM PUFFS WITH CHOCOLATE SAUCE
=====
=====     2    OUNCES BUTTER
=====     1/2  TEASPOON SUGAR
=====     1/2  CUP FLOUR
=====          PINCH OF SALT
=====     2    EGGS
=====     2    CUPS HEAVY CREAM, WHIPPED
===== * * * END OF FILE * * *
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...








===> GET COOKIES COOKBOOK
                                                                   X E D I T  1 FILE
```



```
 DESSERT  COOKBOOK A1  F 80  TRUNC=80 SIZE=15 LINE=15 COLUMN=1
EOF REACHED
=====     1    PINCH OF SALT
=====     2    EGGS
=====     2    CUPS HEAVY CREAM, WHIPPED
===== ALMOND COOKIES
=====
=====     6    TABLESPOONS SOFT BUTTER
=====     1/2  CUP SUGAR
=====     2    EGG WHITES
=====     1    PINCH SALT
=====     1    CUP ALMONDS, SLICED
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== * * * END OF FILE * * *








===>
                                                                   X E D I T  1 FILE
```

Figure 1-9. Inserting a Whole File

A third way to specify a target is the file line number. To display the line numbers in the prefix area, you must issue the following subcommand:

```
===> SET NUMBER ON
```

Here's how the above lines might look:

```
00010 a loaf of bread
00011 a jug of wine
00012 thou
00013 a portable television
```

To specify a target as a line number, type a colon (:) followed by the line number.

The following subcommand puts lines up to, but not including, line 13.

```
===> PUT :13
```

3. Enter a QUIT subcommand. The first file reappears on the screen.
4. Make sure that the current line is the line after which you want the lines from the second file to be inserted. Then enter the following subcommand:

   ```
   ===> GET
   ```

   No operands are required. The lines that were stored by the PUT subcommand are inserted; the last line inserted becomes the new current line.

Figure 1-10 shows how the PUT and GET subcommands are used to insert part of a file into another file:

The file DESSERT COOKBOOK promises a recipe for cream puffs with chocolate sauce. The cream puffs recipe is there, but the chocolate sauce is missing. All the sauces are contained in another file called SAUCES COOKBOOK. To insert the recipe for chocolate sauce after the recipe for cream puffs, first make the desired line current (via /) in the file DESSERT COOKBOOK. Since the sauce recipe must follow the cream puffs recipe, the current line is the last line of the cream puffs recipe (Figure 1-10, Part 1). Then enter the following subcommand:

```
===> XEDIT SAUCES COOKBOOK
```

This file appears on the screen. The status area (lower right corner) indicates that two files are being edited. Move the line pointer to the beginning of the lines to be inserted, via UP or /. The beginning line contains "CHOCOLATE SAUCE" (Figure 1-10, Part 2). Now enter the subcommand to store the chocolate sauce recipe:

```
===> PUT/VINAIGRETTE/
```

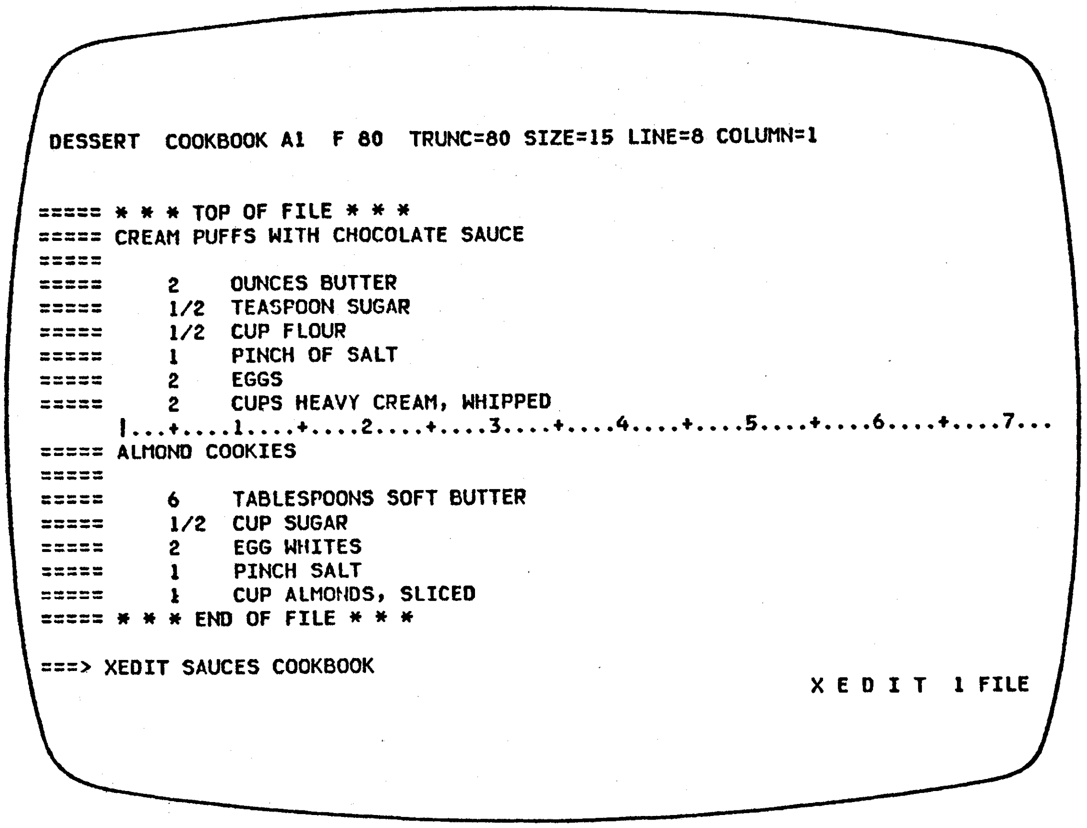
The lines that are stored begin with "CHOCOLATE SAUCE" and end with the line *preceding* "VINAIGRETTE". The PUT subcommand could also have been entered as PUT :15 or PUT 7. In this screen, line numbers are displayed in the prefix area, which means that a SET NUMBER ON subcommand was issued. After the PUT subcommand is executed, you can quit this file by entering:

```
===> QUIT
```

The original file comes back on the screen (Figure 1-10, Part 3). Now enter the following subcommand to insert the lines that were "put":

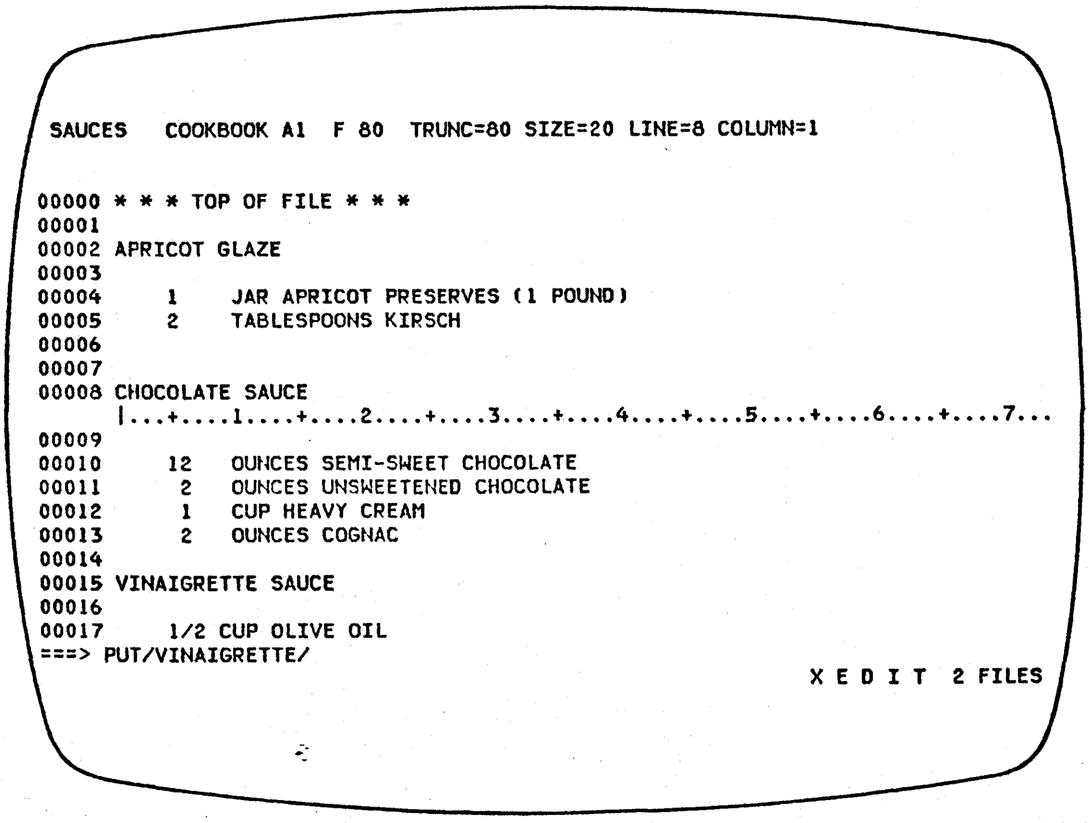
```
===> GET
```

The sauce recipe is inserted, as shown in Figure 1-10, Part 4. The last line inserted is the new current line.



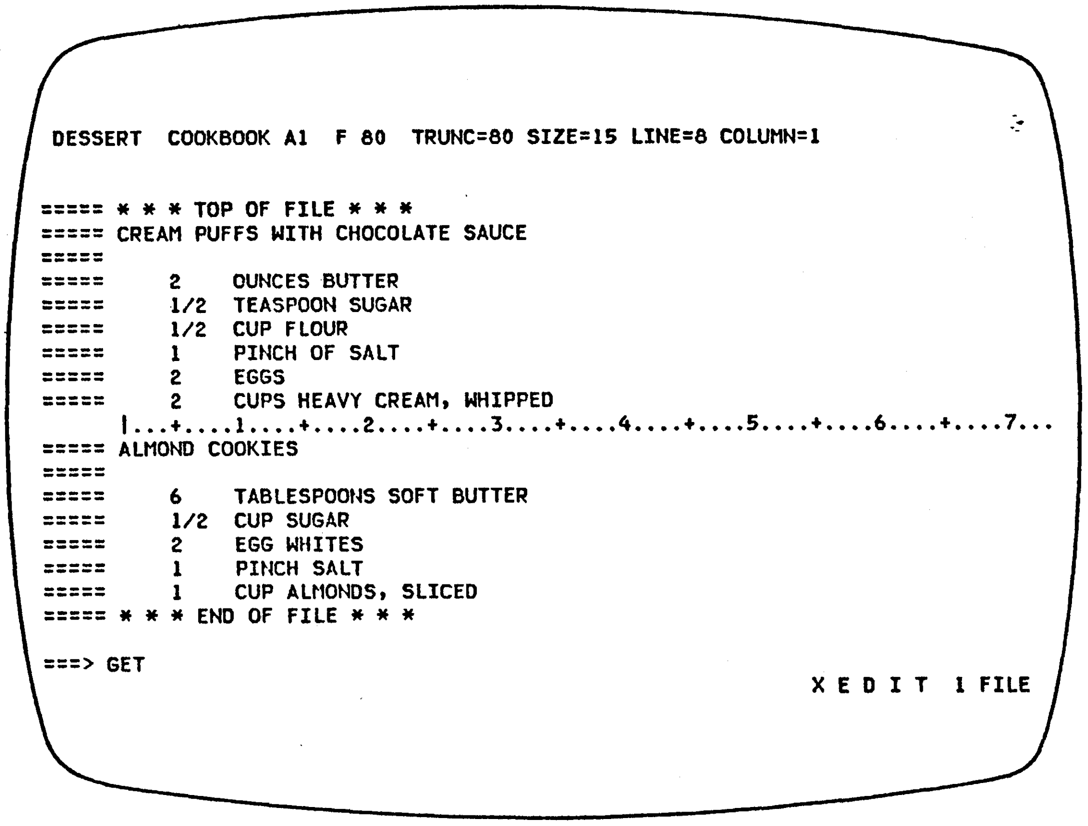
```
DESSERT  COOKBOOK A1  F 80  TRUNC=80 SIZE=15 LINE=8 COLUMN=1


===== * * * TOP OF FILE * * *
===== CREAM PUFFS WITH CHOCOLATE SAUCE
=====
=====      2    OUNCES BUTTER
=====      1/2  TEASPOON SUGAR
=====      1/2  CUP FLOUR
=====      1    PINCH OF SALT
=====      2    EGGS
=====      2    CUPS HEAVY CREAM, WHIPPED
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== ALMOND COOKIES
=====
=====      6    TABLESPOONS SOFT BUTTER
=====      1/2  CUP SUGAR
=====      2    EGG WHITES
=====      1    PINCH SALT
=====      1    CUP ALMONDS, SLICED
===== * * * END OF FILE * * *

===> XEDIT SAUCES COOKBOOK
                                                          X E D I T  1 FILE
```

Figure 1-10. Inserting Part of a File - Call out the Second File (Part 1 of 4)



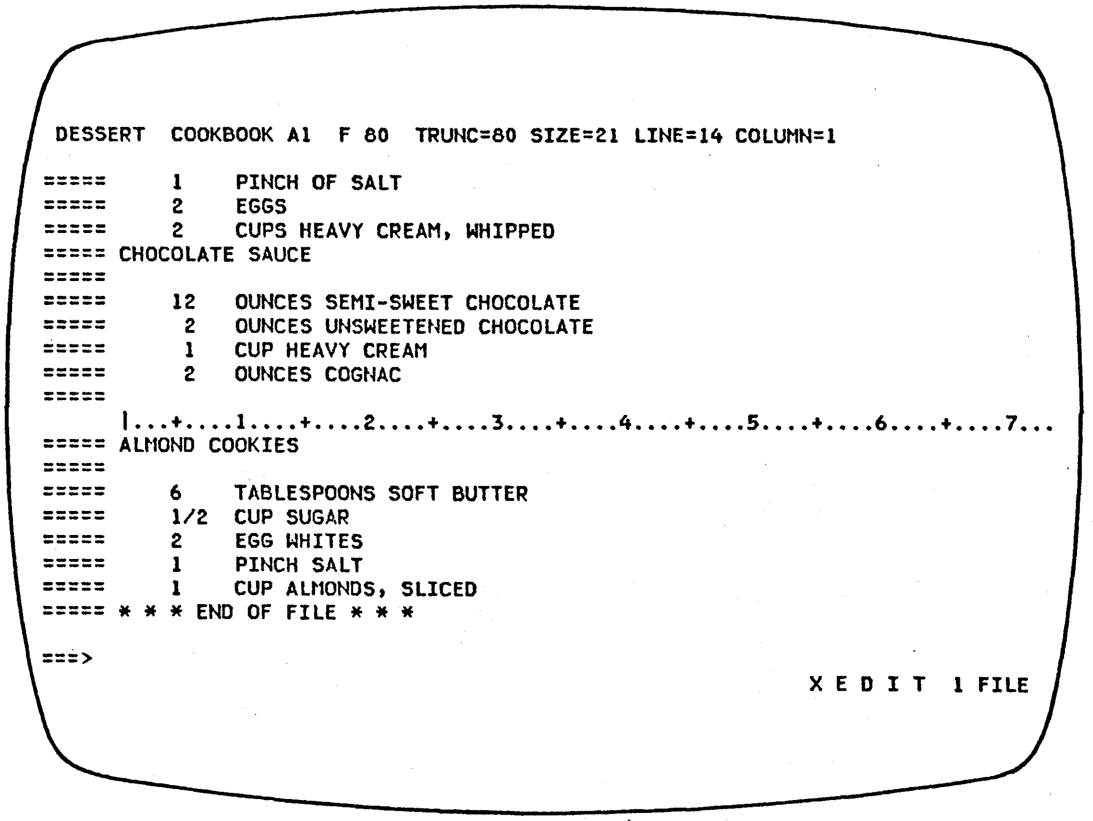
```
SAUCES   COOKBOOK A1  F 80  TRUNC=80 SIZE=20 LINE=8 COLUMN=1


00000 * * * TOP OF FILE * * *
00001
00002 APRICOT GLAZE
00003
00004      1    JAR APRICOT PRESERVES (1 POUND)
00005      2    TABLESPOONS KIRSCH
00006
00007
00008 CHOCOLATE SAUCE
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
00009
00010      12   OUNCES SEMI-SWEET CHOCOLATE
00011       2   OUNCES UNSWEETENED CHOCOLATE
00012       1   CUP HEAVY CREAM
00013       2   OUNCES COGNAC
00014
00015 VINAIGRETTE SAUCE
00016
00017      1/2 CUP OLIVE OIL
===> PUT/VINAIGRETTE/
                                                          X E D I T  2 FILES
```

Figure 1-10. Inserting Part of a File - Put Lines to be Inserted, then QUIT (Part 2 of 4)



```
 DESSERT  COOKBOOK A1  F 80  TRUNC=80 SIZE=15 LINE=8 COLUMN=1


===== * * * TOP OF FILE * * *
===== CREAM PUFFS WITH CHOCOLATE SAUCE
=====
=====     2    OUNCES BUTTER
=====     1/2  TEASPOON SUGAR
=====     1/2  CUP FLOUR
=====     1    PINCH OF SALT
=====     2    EGGS
=====     2    CUPS HEAVY CREAM, WHIPPED
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== ALMOND COOKIES
=====
=====     6    TABLESPOONS SOFT BUTTER
=====     1/2  CUP SUGAR
=====     2    EGG WHITES
=====     1    PINCH SALT
=====     1    CUP ALMONDS, SLICED
===== * * * END OF FILE * * *

===> GET
                                                        X E D I T  1 FILE
```

Figure 1-10. Inserting Part of a File - GET (Part 3 of 4)



```
 DESSERT  COOKBOOK A1  F 80  TRUNC=80 SIZE=21 LINE=14 COLUMN=1

=====     1    PINCH OF SALT
=====     2    EGGS
=====     2    CUPS HEAVY CREAM, WHIPPED
===== CHOCOLATE SAUCE
=====
=====     12   OUNCES SEMI-SWEET CHOCOLATE
=====      2   OUNCES UNSWEETENED CHOCOLATE
=====      1   CUP HEAVY CREAM
=====      2   OUNCES COGNAC
=====
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== ALMOND COOKIES
=====
=====     6    TABLESPOONS SOFT BUTTER
=====     1/2  CUP SUGAR
=====     2    EGG WHITES
=====     1    PINCH SALT
=====     1    CUP ALMONDS, SLICED
===== * * * END OF FILE * * *

===>
                                                        X E D I T  1 FILE
```

Figure 1-10. Inserting Part of a File - The Lines are Inserted (Part 4 of 4)

Getting Help

If you forget how to use a subcommand or would like to see information about subcommands not covered in this subset, you can press the PF1 key, which is set to the HELP MENU subcommand.

When the PF1 key is pressed, a list of all subcommands and macros available with the editor appears on the screen. You then move the cursor to the desired subcommand and press the appropriate PF key (which is defined on the screen). The subcommand description appears on the screen, replacing the file display. Pressing the PF12 key takes you out of the Help display and restores your file on the screen.

Learning More About the Editor

The following is a list of additional XEDIT subcommands that are useful in text processing. You can learn how to use them by using the Help facility (described above) or by referring to the publication *VM/SP: System Product Editor Command and Macro Reference.*

ALTER
: Allows you to change a character to one that is not available on your keyboard, like a backspace character.

COMPRESS, EXPAND
: Allow you to reposition data in new tab columns without retyping it.

LEFT, RIGHT
: Allow you to view columns of data that extend to the left or right of the screen display.

LOWERCAS, UPPERCAS
: Allow you to translate alphabetic characters to all lowercase or all uppercase.

SET ARBCHAR
: Allows you to specify only the beginning and end of a long string that is to be located or changed.

SET CASE
: Allows you to choose whether data that is typed on the terminal is to be entered in the file the same way you type it or translated into uppercase.

SET POINT
: Allows you to assign a name to any line; you can reference the name in XEDIT subcommands.

SET SCREEN
: Allows you to view multiple files or multiple views of the same file on one screen.

SET VERIFY
: Allows you to view only specified columns of data, in character or hexadecimal or both.

SORT
: Allows you to arrange the file lines in alphabetical order.

Summary of XEDIT Subset

The following table summarizes the subcommands that have been presented in this chapter. When a subcommand can be abbreviated, its minimum abbreviation is shown in uppercase letters.

Function	Subcommand/PF Key
To create or edit a file	**XEDIT** (CMS command)
To enter data	**I**nput **POW**erinp
To scroll the screen	**BA**ckward **FO**rward **TOP** **B**ottom
To move the line pointer	**D**own **U**p
To move the column pointer	**CL**ocate **CF**irst
To make changes to the file	**C**hange **CI**nsert
To locate data	**CL**ocate
To recover deleted data	**REC**over
To set tabs	**SET TABS** **MOD**ify **TABS**
To display current tab settings	**Q**uery **TABS**
To display line numbers in the prefix area	**SET NUM**ber **ON**
To end an editing session without saving the changes	**QUIT**
To save automatically after changing a specified number of lines	**SET AU**tosave
To save the changed file when you have finished working on it	**FILE**
To store lines to be inserted in another file by a subsequent GET	**PUT**
To imbed a complete or a partial copy of one file in another	**GET**
To cancel pending prefix subcommands	**RES**et
Prefix subcommands:	
To add lines	**A**
To delete lines	**D**
To duplicate lines	**"**
To move lines	**M** and **F** or **P**
To copy lines	**C** and **F** or **P**
To set the current line	**/**

PF keys, initial settings:	
To get a Help display	**PF1**
To add a line	**PF2**
To end a session without saving	**PF3**
To use a tab key	**PF4**
To locate and change selectively	**PF5, PF6**
To redisplay a subcommand	**PF6**
To scroll one screen backward	**PF7**
To scroll one screen forward	**PF8**
To repeat previous subcommand	**PF9**
To split a line at the cursor	**PF10**
To join two lines at the cursor	**PF11**
To move cursor to current column	**PF12**

Chapter 2: A Practice Exercise

This chapter is designed to give you practice in using some of the XEDIT subcommands discussed in Chapter 1.

The exercise is divided into five parts. You do not have to do all of them at one time, but you should do them in sequence.

Some of the data you will be asked to type contains errors, so that you can use subcommands to correct them.

Remember to press the ENTER key each time you type a subcommand in the command line. However, when you press a PF key, do not press the ENTER key.

Exercise 1. Creating a File

This part of the exercise covers the following subcommands: SET AUTOSAVE, INPUT, QUERY TABS, SET TABS, FILE, and the PF4 key.

Your first file will contain a list of famous inventions. The filename is INVENTOR; the filetype is SCRIPT.

Type the following command in the CMS command line:

```
xedit inventor script
```

Now press the ENTER key. The file identification line appears on the first line of the screen. The message, CREATING NEW FILE:, appears on the second line (the message line). Take a moment to review the screen layout described in Figure 1-1. Notice that the cursor is positioned on the command line, after the large arrow (===>).

To cause your file to be written to disk at periodic intervals, enter the following subcommand:

```
===> set autosave 20
```

You will enter data in the file using the PF4 key for tabbing. To display the editor's initial tab settings for this filetype, enter:

```
===> query tabs
```

The tab settings for a SCRIPT filetype are displayed in the message line. You are going to use different tab settings, so enter:

```
===> set tabs 10 30
```

Now you're ready to begin entering data. Enter:

```
===> input
```

The cursor is positioned on the first line of the input zone. Press the PF4 key, and the cursor moves to the column (10) you specified in the SET TABS subcommand. Type:

```
         Telescope
```

Press the PF4 key again. The cursor moves to column 30. Type:

```
                    1608
```

Press the PF4 key. The cursor moves to column 10 on the next line of the input zone. Type:

```
          Hot air balloon
```

Press the PF4 key and then type:

```
                    1783
```

Using the PF4 key to move the cursor, type the following:

```
          Margarine           1869
          Tranquilizer        1952
```

Now press the ENTER key. The status area (lower right corner) shows that you are still in input mode. The data you entered has moved up on the screen, with the last line you typed becoming the new current line. If you had more data to type, you could start typing at the cursor position. For now, press the ENTER key to return to edit mode.

Checkpoint:
If you have done everything correctly, your screen should look like this:

```
===== * * * TOP OF FILE * * *
=====          Telescope          1608
=====          Hot air balloon    1783
=====          Margarine          1869
=====          Tranquilizer       1952
```

Enter:

```
===> file
```

Exercise 2. Using Power Typing

This part of the exercise covers the following subcommands: POWER, TOP, BOTTOM, UP, DOWN, /, the PF10 key, and the PA2 and insert mode keys.

Your second file will contain a description of the invention of the telescope. Enter:

```
xedit telescop script
```

In this file, you will enter the data in power typing mode. Enter:

```
===> power
```

In power typing mode, you type continuously, without regard to the length of the screen line. If you come to the end of a line and you're in the middle of a word, just keep on typing. The cursor will move to the beginning of the next line. Two of the words that you type will start on one line and end on the next - "accidentally" and "mounted". Now type the following data (with errors):

```
One day in 1608 held a lens in each hand and peered through both at once, accide
ntally discovering that two lenses placed in line would magnify an image. #He mo
unted lens at each end of a tube and invented the telescope.
```

Press the ENTER key twice.

Checkpoint:
Your file should look like this:

```
One day in 1608 held a lens in each hand and peered through both at
once, accidentally discovering that two lenses placed in line would
magnify an image.
He mounted lens at each end of a tube and invented the telescope.
```

The two words that began on one line and finished on the next ("accidentally" and "mounted") are put back together. The second sentence starts on a new line, because you typed a pound sign (#) before it. (Remember that a pound sign, the line end character, causes the data that follows it to start on a new line.)

Obviously, the first sentence is missing some words. One way to insert a long phrase in a line is to split the line in two. Move the cursor under the "h" in "held". Press the PF10 key, and the line is split. Now type:

```
a Dutch spectacle maker named Lippershey
```

In the second sentence, the word "a" is missing before the word "lens". Move the cursor under the "l" in "lens". Press the PA2 key, and press the insert mode key. Type the word "a" and press the space bar once. The sentence has moved over to accomodate the added word. Now press the RESET key, to take you out of insert mode.

Checkpoint:
Your file should look like this:

```
One day in 1608 a Dutch spectacle maker named Lippershey
held a lens in each hand and peered through both at
once, accidentally discovering that two lenses placed in line would
magnify an image.
He mounted a lens at each end of a tube and invented the telescope.
```

The rest of this exercise will give you practice in moving the line pointer. Enter:

```
===> top
```

The new current line is the TOP OF FILE line. If you wanted to add data at the beginning of the file in either input mode or power typing mode, you would enter TOP, followed by either INPUT or POWER.

Enter:

```
===> bottom
```

The new current line is the last line of the file. Enter:

```
===> up 2
```

The new current line is two lines up, toward the top of file.

Enter:

```
===> down 2
```

The new current line is two lines down, toward the end of file.

Now type a / (diagonal) in the prefix area of any line, like this:

```
====/    or  this:   ==/==    or  this:   /====
```

When you press the ENTER key, that line becomes the new current line.

When your file is too big to fit on one screen, you can use the PF7 and PF8 keys (the BACKWARD and FORWARD subcommands) to scroll the screen.

Enter the following subcommand to write this file on disk:

```
===> file
```

Exercise 3. Using Prefix Subcommands

This part covers the RECOVER subcommand and the following prefix subcommands: a, d, m, and p.

To create this file, enter:

```
xedit balloon script
```

Enter:

```
===> input
```

Type:

```
The heat inflated the petticoat and caused it to rise.
The Montgolfier brothers were paper manufacturers.
Hot air from a fire lifted the first balloon.
```

Press the ENTER key twice to re-enter edit mode.

Let's rearrange these sentences. Type an "M" in the prefix area of the second sentence, and a "P" in the prefix area of the first sentence, like this:

```
====p The heat inflated the petticoat and caused it to rise.
===m= The Montgolfier brothers were paper manufacturers.
```

Now press the ENTER key. The sentences have been reversed.

Type an "a" in the prefix area of the first sentence in the file and press the ENTER key. Type the following in the blank line you just added:

```
They realized hot air's ability to float a balloon by accident.
```

The cursor is at the end of the line you just typed. Without moving the cursor, press the PF2 key, which adds a new blank line and moves the cursor to the beginning of it.

Now type:

```
Jacques' wife washed a petticoat and hung it over a fire to dry.
```

Type "5a" in the prefix area of the last line, and press the ENTER key. Type in anything you want. Now, type "DD" in both the first and last lines you added, like this:

```
=dd== This is your first line.
        •
        •
=dd== This is your fifth line.
```

Press the ENTER key.

Do you really want to keep those lines? If you do, enter:

```
===> recover *
```

Checkpoint:
Your file should look like this:

```
The Montgolfier brothers were paper manufacturers.
They realized hot air's ability to float a balloon by accident.
Jacques' wife washed a petticoat and hung it over a fire to dry.
The heat inflated the petticoat and caused it to rise.
Hot air from a fire lifted the first balloon.
```

Enter:

```
===> file
```

Exercise 4. Making Changes

This part of the exercise covers the following subcommands: CHANGE, PF5, and PF6 keys for a selective change.

Enter:

```
xedit margarin script
```

Enter:

```
===> input
```

Type these lines:

```
Bitter was expensive and in short supply.
Napoleon sought a substitute for butter that wasn't bitter.
He needed something like bitter that would store well on ships.
He held a contest and offered a prize for the best bitter substitute.
```

Press the ENTER key twice to re-enter edit mode.

Move the line pointer to the first line of the file by entering:

```
===> up 3
```

Enter:

```
===> change/Bitter/Butter
```

Now you're going to practice using the PF5 and PF6 keys to make a selective change. You want to change "bitter" to "butter", but not all of the time.

Type the following subcommand in the command line, but *do not press the ENTER key.*

```
===> c/bitter/butter
```

Now press the PF5 key. The cursor moves under "bitter" in the second sentence, and the line is highlighted. The message line tells you that if you want to make the change, press the PF6 key. This "bitter" is fine, so press the PF5 key again.

In the third sentence, you want to make the change, so press the PF6 key. The message line tells you that the change has been made.

Press the PF5 key.
Press the PF6 key.

Checkpoint:
Your file should look like this:

```
Butter was expensive and in short supply.
Napoleon sought a substitute for butter that wasn't bitter.
He needed something like butter that would store well on ships.
He held a contest and offered a prize for the best butter substitute.
```

Enter:

```
===> file
```

Exercise 5. Getting It All Together

This part covers the following subcommands: GET and PUT.

You now have the following files:

inventor script
telescop script
balloon script
margarin script

The following exercise will give you practice in transferring data between files. Enter:

```
xedit inventor script
```

You are going to insert the entire file named TELESCOP SCRIPT at the end of this file.

To make the last line of this file current, enter:

```
===> bottom
```

Now enter:

```
===> get telescop
```

You do not have to specify a filetype when you GET a file if the filetype of the file you are "getting" is the same as the file you're currently editing.

The message, "EOF REACHED" tells you that the entire file has been inserted. The new current line is the last line that was inserted. The file TELESCOP is still on disk; only a copy of it has been inserted.

Now you're going to insert part of a file into this one.
Enter:

```
===> xedit balloon
```

This file now appears on the screen. Notice that the status area indicates that you are editing two files, that is, two files are in virtual storage.

You're going to insert lines two and three into the INVENTOR file. Enter:

```
===> down 2
```

Enter:

```
===> put 2
```

Enter:

```
===> quit
```

The INVENTOR file now appears on the screen. Enter:

```
===> get
```

Lines two and three from the BALLOON file are inserted; the new current line is the last line that was inserted.

Now you're going to insert the entire MARGARIN file. Enter:

```
===> get margarin
```

The entire file is inserted.

Checkpoint:
Your file should look like this:

```
         Telescope           1608
         Hot air balloon     1783
         Margarine           1869
         Tranquilizer        1952
One day in 1608 a Dutch spectacle maker named Lippershey
held a lens in each hand and peered through both at
once, accidentally discovering that two lenses placed in line would
magnify an image.
He mounted a lens at each end of a tube and invented the telescope.
They realized hot air's ability to float a balloon by accident.
Jacques' wife washed a petticoat and hung it over a fire to dry.
Butter was expensive and in short supply.
Napoleon sought a substitute for butter that wasn't bitter.
He needed something like butter that would store well on ships.
He held a contest and offered a prize for the best butter substitute.
```

You have inserted two whole files and one partial file into another file. This is a good place to practice prefix subcommands. Using the "A" prefix subcommand, add lines between the different inventions, and then type headings in those lines. You can also rearrange the inventions by using the "M" and "P" (or "F") prefix subcommands. When you are finished, enter:

```
===> quit
```

A warning message tells you to issue a FILE subcommand if you want to save the changes you have made during this part of the exercise. If you don't, enter:

```
===> qquit
```

Chapter 3: An XEDIT Subset: Text Processing on a Typewriter Terminal

This chapter is written primarily for the person who has limited data processing experience; however, some VM/SP CMS experience is assumed. For example, you must know how to log on to VM/SP and enter the CMS environment. You should also be familiar with the concept of a CMS file.

When you finish this chapter, you should have a working knowledge of the editor. The subcommands presented here comprise a subset of XEDIT subcommands, with which you can create a file, enter data, make changes to the file, and transfer data between files. The editor has many additional capabilities, which are described in the rest of this book and in the publication *VM/SP: System Product Editor Command and Macro Reference.*

This subset has been selected for text processing on a typewriter terminal.

Editing a File

To edit a file means to make changes, additions, or deletions to a CMS file that is on a disk, and to make these changes interactively: you instruct the editor to make a change, the editor makes it, and then you request another change.

You can edit a file that does not exist; when you do so, you are creating a file.

XEDIT Command

After you log on to VM/SP and enter the CMS environment, you are ready to enter the edit environment.

The editor is invoked with the CMS command XEDIT, whose format is as follows:

XEDIT filename filetype

If the file already exists on your A-disk, a copy of that file is brought into virtual storage; then you can use XEDIT subcommands to make changes or corrections to lines in that file. You enter an XEDIT subcommand by typing the subcommand and then pressing the RETURN key. (XEDIT subcommands, like CMS commands, can be typed in either uppercase or lowercase, or a combination of both.)

If the file is not found on disk, the editor creates it in virtual storage.

When a subcommand changes a line, the editor displays, or "verifies", the changed line. The editor also communicates with you by displaying error or information messages. For purposes of illustration in this chapter, anything displayed by the editor is enclosed in a box. Subcommands or data that you would enter are not.

Now let's create a simple file. Its filename and filetype will be POEM1 SCRIPT. The following command is entered to begin creating the file:

```
XEDIT POEM1 SCRIPT
```

Because the file is new, the editor responds with the following message:

```
CREATING NEW FILE :
```

Entering Data

The following subcommands are discussed in this section:

INPUT
QUERY LRECL
SET CASE

INPUT Subcommand

After you enter the XEDIT command, you are in edit mode. You must be in edit mode to enter XEDIT subcommands.

However, to enter data in the file, you must be in input mode. Type the following subcommand and press the RETURN key to enter input mode:

```
INPUT
```

The editor displays the following message:

```
INPUT MODE :
```

You can then type in the data. Each line that you enter while in input mode is considered to be a data line and will be written in the file. To end a line, press the RETURN key; the line will then be inserted into the copy of the file in virtual storage.

No line may be longer than the logical record length of the file, which varies according to filetype. To find out the logical record length of any file, you can enter the following subcommand (in edit mode):

```
QUERY LRECL
```

In the examples used here, the filetype is SCRIPT, which has a logical record length of 132. If you type more than 132 characters in a line before pressing the RETURN key, the editor truncates the extra characters.

Now let's start typing lines to be entered in the file:

```
"THE OCTOPUS", by Ogden Nash
Tell me, O Octopus, I begs,
Is those things arms, or is they legs?
I marvel at thee, Octopus;
If I were thou, I'd call me Us.
```

When you are finished typing data and want to return to edit mode (either to make changes to the file or to end the editing session), *press the RETURN key on a null line.*

During an editing session, you can enter input mode at any time to insert new lines of data in the file. After the INPUT subcommand is entered, the editor inserts the lines you type after the current line. In this example, since the file is new, the lines are inserted at the beginning of the file. Later, you will see how to make any line the current line, so that you can insert lines between any two existing lines in a file.

This is how the data looks in the file. The following two subcommands, which will be discussed later, are used to display the data that was entered in input mode:

```
TOP
```

```
TOF:
```

```
TYPE *
```

```
TOF:
"THE OCTOPUS", by Ogden Nash
Tell me, O Octopus, I begs,
Is those things arms, or is they legs?
I marvel at thee, Octopus;
If I were thou, I'd call me Us.
EOF:
```

Column Pointer

Notice that the first letter in each line is underscored. This underscore character (_) is not contained in the file, and it will not appear on a printed copy of the file. It represents the *column pointer.*

Various subcommands perform their editing functions within a line starting at the column pointer, which you can move to different column positions by using XEDIT subcommands that will be discussed later. The column under which the column pointer is positioned is called the *current column.* In the example above, the current column is column one.

Moving Through a File

The following subcommands are discussed in this section:

TYPE
UP
DOWN
TOP
BOTTOM

When you use the XEDIT command to create a new file, the file is created in virtual storage. When the XEDIT command is used to call out an existing file, a copy is brought into virtual storage. In either case, you can picture the file as a series of records, or lines; these lines are available for you to change or delete. You can also insert new lines following any line that is already in the file.

Line Pointer

The line that you are currently editing is called the *current line.*

Naturally, the line that is current changes as you move up and down in the file to edit various lines. When the line that is current changes, we say that the line pointer has moved. Many XEDIT subcommands perform their functions starting with the current line and move the line pointer when they are finished.

You can change which line is current, that is, you can move the line pointer, by using the subcommands discussed in this section.

What you do during an editing session is:

- Position the line pointer at the line you want to edit.
- Edit the line (change characters in it, delete it, or insert new lines following it).
- Position the line pointer at the next line you want to edit.

TYPE Subcommand

Many XEDIT subcommands operate either on, or starting with, the current line. For example, the INPUT subcommand inserts new lines of data after the current line. Therefore, you often need to determine which line is current so that you can move the line pointer, if necessary.

To display the current line, enter the TYPE subcommand, whose format is:

```
TYPE
```

To display more than one line, enter the TYPE subcommand with the number of lines you want to see. For example, the following subcommand displays 5 lines, beginning with the current line:

```
TYPE 5
```

To display the entire file, you must first position the line pointer at the top of the file. The following subcommands move the line pointer to the top of the file and then display the entire file:

```
TOP
```

(moves the line pointer to the top of the file and displays "TOF:")

```
TYPE *
```

(displays all the lines in the file)

After the TYPE subcommand is executed, the line pointer is positioned at the last line that was displayed. For example, if you type the entire file, the null "EOF" line will become the new current line. Of course, if you type only one (the current) line, the line pointer will not move.

UP and DOWN Subcommands

You can move the line pointer up or down one or more lines.

The UP subcommand moves the line pointer toward the beginning of the file and displays the new current line. Its format is:

```
UP n
```

where "n" is the number of lines you want to move the line pointer. If the number is omitted, "1" is assumed.

The DOWN subcommand moves the line pointer toward the end of the file and displays the new current line. Its format is:

```
DOWN n
```

where "n" is the number of lines you want to move the line pointer. If the number is omitted, "1" is assumed.

Let's look at the poem file again:

```
TOP
```

(move the line pointer to the top of the file)

```
TOF:
```

```
TYPE *
```

(display the whole file)

```
TOF:
"THE OCTOPUS", by Ogden Nash
Tell me, O Octopus, I begs,
Is those things arms, or is they legs?
I marvel at thee, Octopus;
If I were thou, I'd call me Us.
EOF:
```

The TYPE * subcommand was used to display the entire file; since the last line displayed by a TYPE subcommand is the new current line, the "EOF" line is now the current line.

The following subcommands show how the UP and DOWN subcommands are used to move the line pointer up and down in the file. Each time the line pointer is moved, the editor displays the new current line.

```
UP 2
```

(move the line pointer up two lines from the EOF line)

```
I marvel at thee, Octopus;
```

```
DOWN
```

(move the line pointer down one line)

```
If I were thou, I'd call me Us.
```

To insert new lines of data after any existing file line, you can do the following:

- Issue the UP or DOWN subcommand to move the line pointer to the line after which you want the data to be inserted.
- Then enter the INPUT subcommand.

TOP and BOTTOM Subcommands

You can also move the line pointer to the the beginning or end of the file.

To move the line pointer to the null "TOF" line that precedes the first line of the file, issue the following subcommand:

```
TOP
```

To move the line pointer to the last file line, issue the following subcommand:

```
BOTTOM
```

To begin entering new lines either at the beginning or the end of a file, you can use the following sequence of subcommands:

```
TOP (or BOTTOM)
INPUT
```

Then you enter new data lines.

Making Changes in a File

The following subcommands are discussed in this section:

CLOCATE
CFIRST
CINSERT
CDELETE
CAPPEND
CHANGE
=

Often, you need to insert or delete charact rs in a line or change one word to another. The subcommands discussed in this section enable you to insert, delete, or change characters based on the position of the column pointer, which is represented as an underscore character (_) when a line is displayed.

CLOCATE Subcommand

The CLOCATE subcommand is used to move the column pointer to the column where you want to insert, delete, or change characters.

The CLOCATE subcommand searches a file, *beginning with the current line,* for a character string that you specify. Its format is as follows:

```
CLOCATE/string/
```

The character string must be enclosed in delimiters. The diagonal (/) is the delimiter used in these examples, but it may be any character that does not also appear in the character string (for example, CLOCATE.VM/CMS.)

If the string is found, two things happen: the line that contains the string becomes the new current line (and is displayed); and the column pointer moves under the first character of the string.

For example, in the file shown above, the subcommands:

```
TOP
```

(move the line pointer to the top of the file)

```
CLOCATE/legs/
```

(locate the string)

cause the following line to be displayed:

```
Is those things arms, or is they legs?
```

Notice that the line pointer moved to the line containing the string "legs", and the column pointer moved under the first character of the string.

CFIRST Subcommand

After using subcommands that move the column pointer, it's a good idea to reset the column pointer to the beginning of the line. The following subcommand moves the column pointer to the beginning of the line:

```
CFIRST
```

For example, in the line shown above, where the column pointer is under the "l" in "legs", issuing a CFIRST subcommmand results in:

```
Is those things arms, or is they legs?
```

CINSERT Subcommand

The CINSERT subcommand is used to insert characters immediately before the column pointer.

For example, a file contains the following line:

```
Mt. Everest is high.
```

Note the position of the column pointer, in column one. To insert the phrase "exactly 29,000 feet" before the word "high", first move the column pointer to the first character in "high", by using the following subcommand:

```
CLOCATE/high/
```

The editor moves the column pointer and displays the line:

```
Mt. Everest is high.
```

Now you can insert the phrase:

```
CINSERT exactly 29,000 feet
```

The editor inserts whatever you type in the operand of the CINSERT subcommand. In the subcommand above, the space bar was pressed once after the word "feet" so that a blank would separate "feet" and "high".

The resulting line is displayed:

```
Mt. Everest is exactly 29,000 feet high.
```

Let's look at another example. The CLOCATE subcommand is used to move the column pointer; then the CINSERT subcommand is used to insert characters immediately before the column pointer position.

A file contains the following line:

```
If anything can go, it will.
```

```
CLOCATE/,/
```

(move the column pointer)

```
If anything can go, it will.
```

```
CINSERT  wrong
```

(insert "wrong" before the column pointer)

```
If anything can go_wrong, it will.
```

(In the CINSERT subcommand above, note that there are *two spaces* between "CINSERT" and "wrong": one is the required space between the subcommand name and the operand; one is the blank space needed between "go" and "wrong".)

If only one blank space were used, the result would be the following:

```
If anything can gowrong, it will.
```

The editor allows you to insert blanks with the CINSERT subcommand - simply type the required number of blanks (by pressing the space bar) in the operand. For example:

```
If anything can gowrong, it will.
```

```
CINSERT
```

(Press the space bar twice: once to separate the subcommand name and operand; once for the operand.)

```
If anything can go_wrong, it will.
```

CDELETE Subcommand

The CDELETE subcommand is used to delete one or more characters from the current line, starting at the column pointer.

A file contains the following line:

```
To be or not to be or not to be - that is the question.
```

The line contains one too many "or not to be". Since deletion starts at the column pointer, first move the column pointer with the following subcommand:

```
CLOCATE/or/
```

```
To be or not to be or not to be - that is the question.
```

Then, you can use the CDELETE subcommand to specify the number of characters to be deleted. Count the number of characters to be deleted, starting with the current column:

```
CDELETE 13
```

The resulting line looks like this:

```
To be or not to be - that is the question.
```

The CDELETE subcommand issued above specified a "13" as the operand; it means, "delete 13 characters, starting at the column pointer."

If you did not want to count the number of characters, you could have specified the operand of the CDELETE subcommand as a character string. For example:

```
CDELETE/or/
```

When this form of the CDELETE subcommand is used, it means, "delete characters *from* the column pointer *to* the first character of the string specified in the operand." The result would be the same as the line shown above; the extra "or not to be" would be removed.

In summary, the CDELETE subcommand removes characters from a line, from the column pointer to the column position specified in the operand. The operand may be specified as the number of characters to be removed, or it may be specified as a character string. After the CDELETE subcommand is executed, the editor displays the changed line.

CAPPEND Subcommand

Use the CAPPEND subcommand to append words to the end of the current line.

The format of the CAPPEND subcommand is:

```
CAPPEND text
```

where "text" represents the data you want to add to the end of the line.

For example, a file contains the following line:

```
It is an ancient mariner,
```

However, the line *should* read:

It is an ancient mariner, and he stoppeth one of three.

The following subcommand adds the desired text:

```
CAPPEND  and he stoppeth one of three.
```

(Two blanks separate the subcommand name and the operand.)

The resulting line looks like this:

```
It is an ancient mariner,_and he stoppeth one of three.
```

Notice that the column pointer has moved to the first character of the appended text, which was a blank.

CHANGE Subcommand

Changing One Word to Another

Replacing one word with another is the simplest type of change. Use the following form of the CHANGE subcommand to change the first occurrence of a word in the current line:

```
CHANGE/oldword/newword/
```

For example, the current line in a file contains the following:

```
A rose is a rose is a rose.
```

```
CHANGE/rose/daisy/
```

The resulting line looks like this:

```
A daisy is a rose is a rose.
```

Note that the editor automatically makes room in the line for "daisy", even though it is longer than "rose". Conversely, a word can be replaced by a shorter word; the editor removes extra blanks.

You can use the CLOCATE and CHANGE subcommands to locate and change any string in a file. If the line containing the string is the current line, you don't have to use a CLOCATE subcommand; the CHANGE subcommand both locates the string and changes it.

Making a Global Change

If you want to make a global change, that is, change *every occurrence* of a word, first move the line pointer to the line where you want the change to begin, and use the following form:

```
CHANGE/oldword/newword/ * *
```

In the following example, the word "rose" is changed to "daisy" every time it appears. (The line pointer is already positioned at the first line shown.)

```
A rose is a rose is a rose.
A rose is a rose is a rose.
A rose is a rose is a rose.
A rose is a rose is a rose.
```

```
CHANGE/rose/daisy/ * *
```

produces the following changes in the file (the editor displays only those lines that have been changed):

```
A daisy is a daisy is a daisy.
A daisy is a daisy is a daisy.
A daisy is a daisy is a daisy.
A daisy is a daisy is a daisy.
```

Another variation of the CHANGE subcommand can be used when you want to change a word throughout the file, but you want to change only the first occurrence in each line:

```
CHANGE/oldword/newword/ *
```

Making a Selective Change

Suppose that you want to change one word to another only some of the time. You can use repeated executions of the CLOCATE subcommand to scan the file, issuing a CHANGE subcommand only when you want to make the change.

Instead of typing the same CLOCATE subcommand over and over, you can use the = subcommand, which repeats the last subcommand you entered. Using the = subcommand saves you the time it takes to retype the subcommand. To enter the = subcommand, simply type an equal sign (=) and press the RETURN key.

Inserting and Deleting Lines

The following subcommands are discussed in this section:

INPUT line
DELETE
RECOVER
REPLACE

Inserting A Line

You can insert a single line of data between existing lines by using the INPUT subcommand followed by the line of data you want inserted. One blank must separate the subcommand name and the data line.

For example:

```
INPUT this is the line I want to insert
```

inserts a single line following the current line, without leaving edit mode. (If you want to insert more than one line, you would issue the INPUT subcommand with no operand to enter input mode.)

To insert a blank line in the file, enter the INPUT subcommand and press the space bar at least twice before pressing the RETURN key. A blank line will be inserted after the current line.

For example, if a file contains the following lines:

```
TOF:
Some primal termite knocked on wood
And tasted it, and found it good,
And that is why your Cousin May,
Fell through the parlor floor today.
```

The current line is the last line displayed above. To insert a title line, issue the following subcommand:

```
INPUT "The Termite", by Ogden Nash
```

Now the file looks like this (TOP and TYPE 6 are used to display the whole file):

```
TOF:
Some primal termite knocked on wood
And tasted it, and found it good,
And that is why your Cousin May,
Fell through the parlor floor today.
"The Termite", by Ogden Nash
```

To insert a blank line between the poem and the title line, you could issue the following subcommands:

```
UP
```

(move the line pointer up one line)

```
INPUT
```

(press the space bar twice before pressing the RETURN key)

Now the file looks like this:

```
TOF:
Some primal termite knocked on wood
And tasted it, and found it good,
And that is why your Cousin May,
Fell through the parlor floor today.

"The Termite", by Ogden Nash
```

Deleting Lines

Use the DELETE subcommand to delete one or more lines from a file, beginning with the current line.

To delete only the current line, use the form:

```
DELETE
```

To delete more than one line, specify the number of lines in the operand:

```
DELETE 5
```

deletes five lines, including the current line.

To delete the rest of the file, use the form:

```
DELETE *
```

If you want to delete a number of lines, and you don't want to bother counting how many, you can use the form:

```
DELETE/string/
```

Lines will be deleted, starting with the current line, up to (but not including) the line containing the specified string.

For example, if a file contains the following lines, and the first line shown is the current line:

```
a portable television
a transistor radio
a frisbee
a loaf of bread
a jug of wine
thou
```

The following subcommand:

```
DELETE/bread/
```

deletes all lines from the current line up to, but not including, the line containing "bread". Therefore, all that's left are the lines containing "a loaf of bread", "a jug of wine", and "thou".

Lost and Found Department

If you delete one or more lines and change your mind, all is not lost. You can recover the lines at any time during an editing session with the RECOVER subcommand.

The following subcommand returns lines deleted in an editing session:

```
RECOVER n
```

where n represents the number of lines you wish to recover.

The recovered line(s) is inserted immediately before the current line. If the lines were deleted from different places in the file, you have to put them back where they belong by using the MOVE subcommand, discussed below.

If you want to recover *all* lines that have been deleted during an editing session, use the form:

```
RECOVER *
```

Replacing a Line

You've seen how to insert a new line and delete a line, using INPUT *line* and DELETE. The REPLACE subcommand does both; it deletes the current line and replaces it with a line you specify.

The format of the REPLACE subcommand is:

```
REPLACE line
```

However, if you enter the REPLACE subcommand with no line, the editor deletes the current line and automatically places you in input mode.

Moving and Copying Lines

The following subcommands are discussed in this section:

MOVE
COPY

MOVE Subcommand

Suppose you want to remove some lines from their current location and insert them in another part of the file. You can use the MOVE subcommand to move one or more lines, beginning with the current line, to a different location in the file. The format of the MOVE subcommand is as follows:

```
MOVE from to
```

The first operand represents the number of lines to be moved, starting with the current line. The second operand represents the destination; the line(s) is inserted *after* the destination line and is deleted from its original location.

For example, to move the current line three lines down in the file, you can use the following subcommand:

```
MOVE 1 3
```

To move the current line and the two lines following it three lines down in the file, you can use the following subcommand:

```
MOVE 3 3
```

To move a line *backward* in the file, you can specify a minus (-) sign in front of the "to" operand. For example:

```
MOVE 1 -3
```

moves the current line up two lines in the file. Remember, the "to" operand represents the line *after* which a line is to be moved; therefore, if the destination is -3, the line is inserted after that line, or two lines up.

To eliminate the need for counting lines, you can specify the "to" operand as a character string. The editor searches the file for a line that contains the string and moves the "from" line(s) after that line.

For example:

```
MOVE 1 /string/
```

moves the current line after the line containing the string.

Similarly, you can move a line backward in the file by specifying a minus (-) sign before the string. For example:

```
MOVE 1 -/string/
```

moves the current line backward in the file after the line that contains the string.

Let's look at an example:

```
filberts
almonds
cashews
chestnuts
pecans
walnuts
```

The following subcommands would each move the line containing "filberts" (the current line) after the line containing "chestnuts".

```
MOVE 1 3   or   MOVE 1 /chestnuts/
```

```
almonds
cashews
chestnuts
filberts
pecans
walnuts
```

COPY Subcommand

The procedure for copying lines is the same as for moving lines. The COPY subcommand leaves the original line(s) in place and makes a duplicate at the indicated destination.

The format of the COPY subcommand is:

```
COPY from to
```

One or more lines, beginning with the current line, are copied after the destination line.

Ending an Editing Session

The following subcommands are discussed in this section:

FILE
QUIT
SET AUTOSAVE

FILE Subcommand

When you use the XEDIT command to create a new file, the file is created in virtual storage. When you make changes to an existing file, those changes are made to a copy of the file that is brought into virtual storage (when the XEDIT command is entered). However, virtual storage is *temporary.* To write a new or modified file on disk, which is *permanent* storage, you must enter the following subcommand:

```
FILE
```

When the FILE subcommand is executed, the file is written on disk and control is returned to CMS.

QUIT Subcommand

Use the QUIT subcommand to end an editing session and leave the permanent copy of the file intact on the disk. If the file is new, it is not written on disk.

The format of the QUIT subcommand is as follows:

```
QUIT
```

You would use the QUIT subcommand instead of the FILE subcommand when you edit a file merely to examine, but not to change, its contents, or if you discover you have made errors in changing a file and do not want them to be recorded.

When a file is new or has been changed, the editor gives you a warning message to prevent the inadvertent use of a QUIT instead of a FILE. The message is as follows:

```
FILE HAS BEEN CHANGED.  USE QQUIT TO QUIT ANYWAY.
```

If you really don't want to save the file, enter "QQUIT" (abbreviated as "QQ"). If you wish to save the changes, enter "FILE".

SET AUTOSAVE Subcommand

Files on disk are not affected if the system malfunctions, or "goes down." However, a new file that you're creating or the changes you're making to an existing file might be lost if the system fails. You can minimize this danger by using the SET AUTOSAVE subcommand, whose format is as follows:

```
SET AUTOSAVE n
```

The SET AUTOSAVE subcommand causes your file to be written to disk automatically, after you've typed in or changed a certain number of lines. You specify what that number will be with the "n" operand of the SET AUTOSAVE subcommand. If you want the file written to disk, or "saved", every time you've changed ten lines, the subcommand would be:

```
SET AUTOSAVE 10
```

The SET AUTOSAVE subcommand can be issued at any time during an editing session. It's a good idea, however, to issue the subcommand right after you issue an XEDIT command to create a new file or to call an existing file from disk.

If you have issued a SET AUTOSAVE subcommand and the system goes down, your file is written to disk with a new fileid. The filename is a number from 1 to 8, and the filetype is AUTOSAVE.

You can change the fileid back to its original filename and filetype by issuing the CMS command ERASE to erase the original file and then by issuing the CMS command RENAME.

For example, if your AUTOSAVE file is labeled "1 AUTOSAVE A1" and the original file is "POEM1 SCRIPT A1", use the following CMS commands to rename it:

```
ERASE POEM1 SCRIPT
RENAME 1 AUTOSAVE A1 POEM1 SCRIPT A1
```

Then you'll be back in business and can use the XEDIT command to start editing the file again.

A QUIT subcommand cancels a SET AUTOSAVE subcommand. If you issue a SET AUTOSAVE subcommand while you're creating a new file, and then issue a QUIT subcommand, the file is not saved. If you issue a SET AUTOSAVE subcommand while you're revising an existing file and then you issue a QUIT subcommand, no revisions are saved.

Inserting Data from Another File

To insert all or part of one file into another, you can use the GET subcommand. The chapters in this book were created as separate files and then combined into one file by using the GET subcommand. (A file that you "get" is not destroyed; a copy of that file is inserted.)

The GET subcommand inserts a file after the current line. Therefore, you must move the line pointer to the line after which you want to insert a file. If you want to insert another file at the *end* of your file, you can use the BOTTOM subcommand to make the last line current. If you want to insert another file somewhere in the *middle* of your file, you can use the UP or DOWN subcommands to make the desired line current.

Inserting a Whole File

Suppose you were writing a book of poetry, and you created a separate file for each poem. To combine two of the poems into one file, you would use the following form of the GET subcommand:

```
GET filename filetype
```

When the entire second file has been inserted, the editor displays the following message:

```
EOF REACHED
```

For example, if you were editing a file called POEM1 SCRIPT and wanted to insert another file called POEM2 SCRIPT, you would enter the following subcommands:

```
BOTTOM
```

(move the line pointer to the end of the file)

```
GET POEM2 SCRIPT
```

(insert the whole file)

Inserting Part of Another File

To insert part of another file, you can specify in the GET subcommand the line number of the first line and the number of lines you want to insert. The following GET subcommand inserts the first ten lines of a second file:

```
GET FILE2 1 10
```

If you don't know the line numbers, you can: call out a second file without ending your current editing session; put the lines you want to insert into a temporary file; and insert them into your current file.

This might sound complicated, but all you need to learn is one more subcommand - PUT.

First, let's identify the steps you would take to insert part of another file and then illustrate them with an example.

1. While editing the first file, enter an XEDIT subcommand to call out the second file. You do not have to end your current editing session, because the editor allows you to edit multiple files simultaneously.

2. Use the PUT subcommand to indicate which lines are to be inserted in the first file. The PUT subcommand stores lines in a temporary holding area, *starting with the current line,* up to an ending, or target, line. Its format is as follows:

   ```
   PUT target
   ```

   where "target" identifies the end of a group of lines to be inserted. It is a signal to the editor to stop "putting" lines.

   A target operand may be specified in various ways, which are described in detail in "Chapter 4: Using Targets". A brief description of two ways to specify a target follows. They are equivalent; you can choose whichever type you prefer.

   One way to specify the target is to count the number of lines you want to insert, starting with the current line. For example, if a file contains:

   ```
   a loaf of bread
   a jug of wine
   thou
   a portable television
   ```

   and the line containing "a loaf of bread" is current, the following subcommand stores all the above lines:

   ```
   PUT 4
   ```

   Another way to specify the target is with a character string; the editor will "put" all the lines, beginning with the current line, up to, but not including, the line containing the string.

   For example, the following subcommand will "put" the first three lines, but it will not "put" the line containing "a portable television".

```
PUT/television/
```

3. Enter a QUIT subcommand to return to your original file.
4. Make sure that the current line is the line after which you want to insert lines from the second file. Then enter the following subcommand:

   ```
   GET
   ```

   No operands are required. The lines that were stored by the PUT subcommand are inserted; the last line inserted becomes the new current line.

The following example illustrates how the PUT and GET subcommands are used to insert part of a file into another file:

A file, DESSERT COOKBOOK, is being compiled. It contains many recipes, among which is a recipe for cream puffs with chocolate sauce. The author of the cookbook keeps a separate file, called SAUCES COOKBOOK, which contains recipes for sauces. Whenever a recipe requires an accompanying sauce, the author can select a sauce recipe from the second file and insert it in the first. In this example, the recipe for chocolate sauce will be inserted after the recipe for cream puffs.

```
XEDIT DESSERT COOKBOOK
```

(Call out the first file.)

```
CLOCATE/CREAM PUFFS/
```

(Locate the recipe.)

```
TYPE 10
```

(Display the recipe. You could have displayed the whole file by using TYPE *, but it's not necessary.)

```
CREAM PUFFS WITH CHOCOLATE SAUCE
_
_    2     OUNCES BUTTER
_    1/2   TEASPOON SUGAR
_    1/2   CUP FLOUR
_    1     PINCH OF SALT
_    2     EGGS
_    2     CUPS HEAVY CREAM, WHIPPED
_
ALMOND COOKIES
```

```
UP 1
```

(Move the line pointer to the line after which you want to insert the sauce recipe. The editor displays the new current line, which is the blank line between "HEAVY CREAM" and "ALMOND COOKIES".)

```
_
```

```
XEDIT SAUCES COOKBOOK
```

(Edit the second file.)

```
CLOCATE/CHOCOLATE SAUCE/
```

(Locate the sauce recipe.)

```
TYPE 10
```

(Display 10 lines.)

```
CHOCOLATE SAUCE
_
_     12     OUNCES SEMI-SWEET CHOCOLATE
_      2     OUNCES UNSWEETENED CHOCOLATE
_      1     CUP HEAVY CREAM
_      2     OUNCES COGNAC
_
VINAIGRETTE SAUCE
_
_   1/2   CUP OLIVE OIL
```

```
UP 10
```

(Move the line pointer to the beginning of the recipe)

```
PUT/VINAIGRETTE/
```

Lines are stored, beginning with the line containing "CHOCOLATE SAUCE" and ending with the line preceding the one containing "VINAIGRETTE". The PUT subcommand could also be entered as PUT 7.

```
QUIT
```

The original file is now being edited.

```
GET
```

The sauce recipe is inserted.

The resulting file looks like this:

```
CREAM PUFFS WITH CHOCOLATE SAUCE
_
_      2     OUNCES BUTTER
_      1/2   TEASPOON SUGAR
_      1/2   CUP FLOUR
_      1     PINCH OF SALT
_      2     EGGS
_      2     CUPS HEAVY CREAM, WHIPPED
_
CHOCOLATE SAUCE
_
_     12     OUNCES SEMI-SWEET CHOCOLATE
_      2     OUNCES UNSWEETENED CHOCOLATE
_      1     CUP HEAVY CREAM
_      2     OUNCES COGNAC
_
ALMOND COOKIES
```

Using Special Characters

The following subcommands are discussed in this section:

SET IMAGE
SET TABS
QUERY TABS

The SET IMAGE subcommand controls how special characters, once entered on an input line, are going to be represented in a file. The special characters affected by the SET IMAGE subcommand are:

- tab characters (X'05')
- backspace characters (X'16')

The format of the SET IMAGE subcommand is:

```
SET IMAGE ON
          OFF
          CANON
```

Tab Characters

The important thing to remember about tab settings is that there are two kinds: physical and logical.

Physical tab settings are set manually on the typewriter; each time you press the TAB key, the type ball moves to the column you set up as the physical tab stop.

Logical tab settings indicate the column positions where fields within a record begin. They are defined by the SET TABS subcommand, whose format is:

```
SET TABS n1 n2 n3 ...
```

where n1... represents the column numbers for the logical tab settings.

These logical tab settings do not necessarily correspond to the physical tab settings.

How the data is entered in the file when you press the TAB key depends on whether the SET IMAGE subcommand has been issued with ON or OFF as the operand. (SET IMAGE ON is the initial setting for all filetypes except SCRIPT.)

If SET IMAGE ON is in effect when you press the TAB key, the logical tab settings determine how the data will be entered in the file. The editor replaces the tab characters with an appropriate number of blanks, starting at the column where you pressed the TAB key, and ending at the last column before the next logical tab setting. The next character entered after the tab becomes the first character of the next field.

For example, if you enter:

```
SET TABS 1 15
```

and then enter a line that begins with a tab character, the first data character following the tab is written into the file in column 15, regardless of the physical tab stop on the terminal.

If SET IMAGE OFF is in effect, a tab character is inserted in the record, just as any other data character is inserted. No blanks are inserted.

If you want to insert a tab character (X'05') into a record and SET IMAGE ON is in effect, you can issue a SET IMAGE OFF subcommand before entering the line, and then use the TAB key as a character key. Pressing the TAB key causes a tab character to be inserted in a line.

Setting Tabs

When you create a file, default logical tab settings are in effect; therefore, you do not need to set them. To determine the default tab settings for a particular filetype, you can use the following subcommand:

```
QUERY TABS
```

If you want to change the default tab settings, you can use the SET TABS subcommand. Then, regardless of what physical tab stops have been set up on your terminal, when you press the TAB key with SET IMAGE ON in effect, the data you enter is spaced to the columns you defined.

Note: When the INPUT subcommand is used to enter one line, the specified line is placed in the file starting in the first tab column defined by the SET TABS subcommand. For example, if you enter:

```
SET TABS 5 10 15 20
```

and then enter an input line:

```
INPUT This is the input line
```

columns 1, 2, 3, and 4 contain blanks; the text begins in column 5.

Therefore, make sure that the first number specified in the SET TABS subcommand is the column in which you want the data to begin.

Backspace Characters

If you use backspaces and underscores in your file, you should issue SET IMAGE OFF or SET IMAGE CANON. SET IMAGE CANON is the initial setting for SCRIPT files.

SET IMAGE OFF means that backspace characters (as well as tab characters) are left as they are entered.

SET IMAGE CANON means that regardless of how the characters are typed in (characters, backspaces, underscores), the editor orders the characters in the file as: character - backspace - underscore, character - backspace - underscore, and so forth. If, for example, you want an input line to look like this:

<u>ABC</u>

You could enter it as:

```
ABC, 3 backspaces, 3 underscores
```

- or -

```
3 underscores, 3 backspaces, ABC
```

A typewriter types out the line in the following order:

```
A, backspace, underscore
B, backspace, underscore
C, backspace, underscore
```

which results in:

<u>ABC</u>

If you need to modify a line that has backspaces, and you do not want to rekey all of the characters, you can use the ALTER subcommand to alter all of the backspaces to some other character. The following sequence shows how you can delete all of the backspace characters in a line:

```
AAAAA
ALTER 16 + 1 *
```

(alter all X'16's to +'s in this line)

```
_+A_+A_+A_+A_+A
CHANGE/_+// 1 *
```

(change all occurrences of "_+" to null in this line)

```
AAAAA
```

Summary of XEDIT Subset

This table summarizes the subcommands that have been presented in this chapter. When a subcommand can be abbreviated, its minimum abbreviation is shown in uppercase letters.

Function	Subcommand
To create or edit a file	XEDIT (CMS command)
To enter data	Input
To control case setting	SET CASE
To display file lines	Type
To move the line pointer	Down Up TOP Bottom
To move the column pointer	CLocate CFirst
To locate data	CLocate
To make changes to the file	Change CInsert CDelete CAppend
To recover deleted data	RECover
To insert one line	Input line
To delete lines	DELete
To replace a line	Replace
To move lines	MOve
To copy lines	COpy
To repeat a subcommand	=
To control special characters	SET IMage
To define logical tabs	SET TABS
To display tab settings	Query TABS
To display the logical record length	Query LRecl
To alter special character	ALter
To end an editing session without saving the changes	QUIT
To save automatically after changing a specified number of lines	SET AUtosave
To save the changed file when you have finished working on it	FILE
To store lines in temporary file for subsequent imbed in another	PUT
To imbed a complete or a partial copy of one file in another	GET

Chapter 4: Using Targets

What Is a Target?

The ability to locate a line from a target is one of the editor's most versatile functions.

Very simply, a target is a way that you identify a line to the editor. Targets are used to identify lines for two basic reasons:

1. to change which line is the current line
2. to define the scope of a subcommand's execution.

A target may be entered in the following ways:

- by itself
- as the operand of the LOCATE subcommand
- before any XEDIT subcommand
- as the operand(s) in many other XEDIT subcommands.

When a target is entered either by itself or as the operand of a LOCATE subcommand, the editor makes the target line the *new current line.* Entered before a subcommand, a target causes the editor to make the target line the new current line before it executes the subcommand.

When a target is entered as the operand of various other XEDIT subcommands, it defines the scope of that subcommand's execution. Most XEDIT subcommands *begin* their operation with the current line; the target operand is used to specify where the operation is to *end.*

The following XEDIT subcommands have target operands:

ALTER	DUPLICAT	REPEAT
CHANGE	EXPAND	SHIFT
COMPRESS	HEXTYPE	SORT
COPY	LOWERCAS	STACK
COUNT	MOVE	TYPE
DELETE	PUT/PUTD	UPPERCAS

Refer to the publication *VM/SP: System Product Editor Command and Macro Reference* for a complete description of the subcommand formats.

There are various ways to specify any given target; all achieve the same result. How fancy you want to be depends on you. If you are a new user, you can specify targets in a simple way. As you become more experienced, you can take advantage of the flexibility that targets offer.

A target can be expressed in the following ways:

- an absolute line number
- a relative displacement from the current line
- a line name
- a simple string expression
- a complex string expression.

You can use one or all of the above kinds of targets during an editing session; you can even use different kinds of target operands in the same subcommand.

Using a Target to Change Which Line is Current

A Target Entered By Itself

Look at Figure 4-1. When entered on the command line, any of the targets listed below would change the current line to the one shown in the bottom screen. (The current line is the line above the scale.) All the targets shown below are equivalent; which kind you use depends only on personal preference. How to use each kind of target is discussed throughout this chapter; the purpose of Figure 4-1 is to show you that there are various ways to identify any given line to the editor.

```
===> :11
```

(absolute line number)

```
===> +6
```

(relative displacement from the current line)

```
===> .CLAUDE
```

(line name previously assigned by SET POINT)

```
===> /egg/
```

(string)

The editor begins searching for the target with the line following the current line; if the target line is located, it becomes the new current line.

Notice that in the file identification line at the top of the screen, the "LINE=" indicator shows that the current line has changed from line 5 (top screen) to line 11 (bottom screen).

A Target as the Operand of a LOCATE Subcommand

The targets listed above could have been specified as operands of the LOCATE subcommand, like this:

```
===> LOCATE :11
===> LOCATE +5
===> LOCATE .CLAUDE
===> LOCATE /egg/
```

You do not need to type "LOCATE" unless you want to. A target specified by itself implies the LOCATE subcommand; the name "LOCATE" is optional.

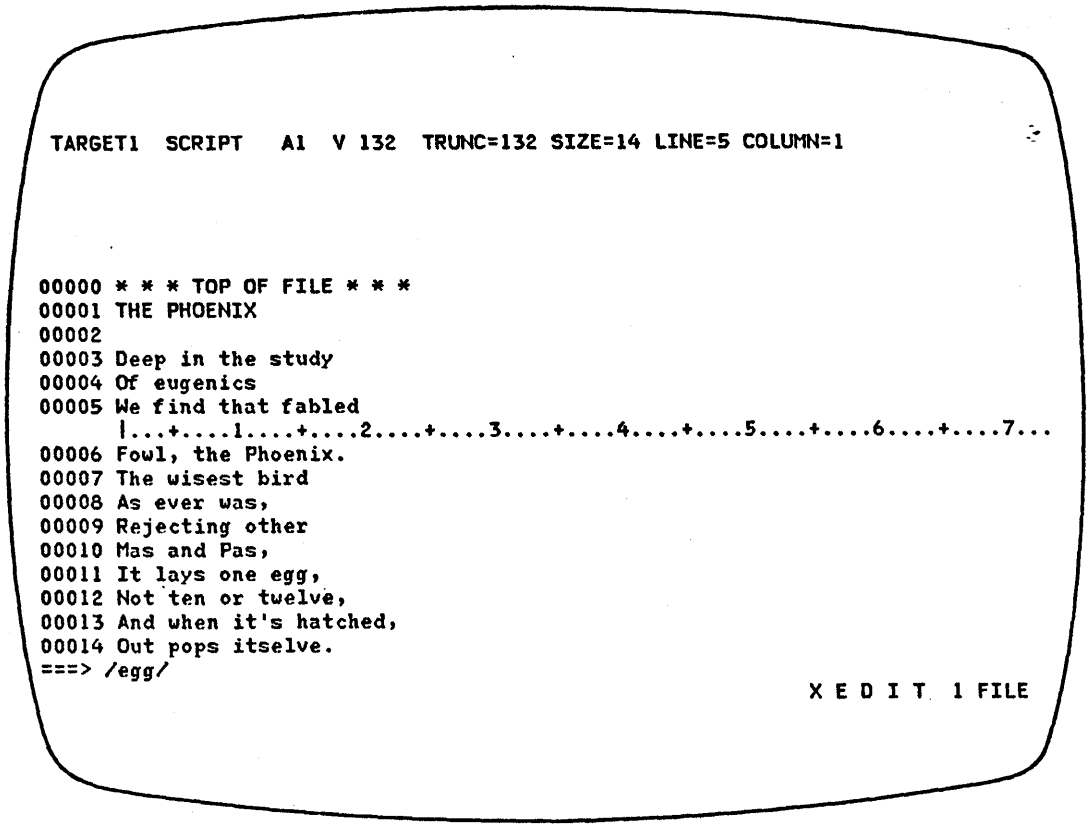
A Target Preceding a Subcommand

A target can be entered in the command line before any XEDIT subcommand. The editor first makes the target line the new current line, and then executes the subcommand. For example:

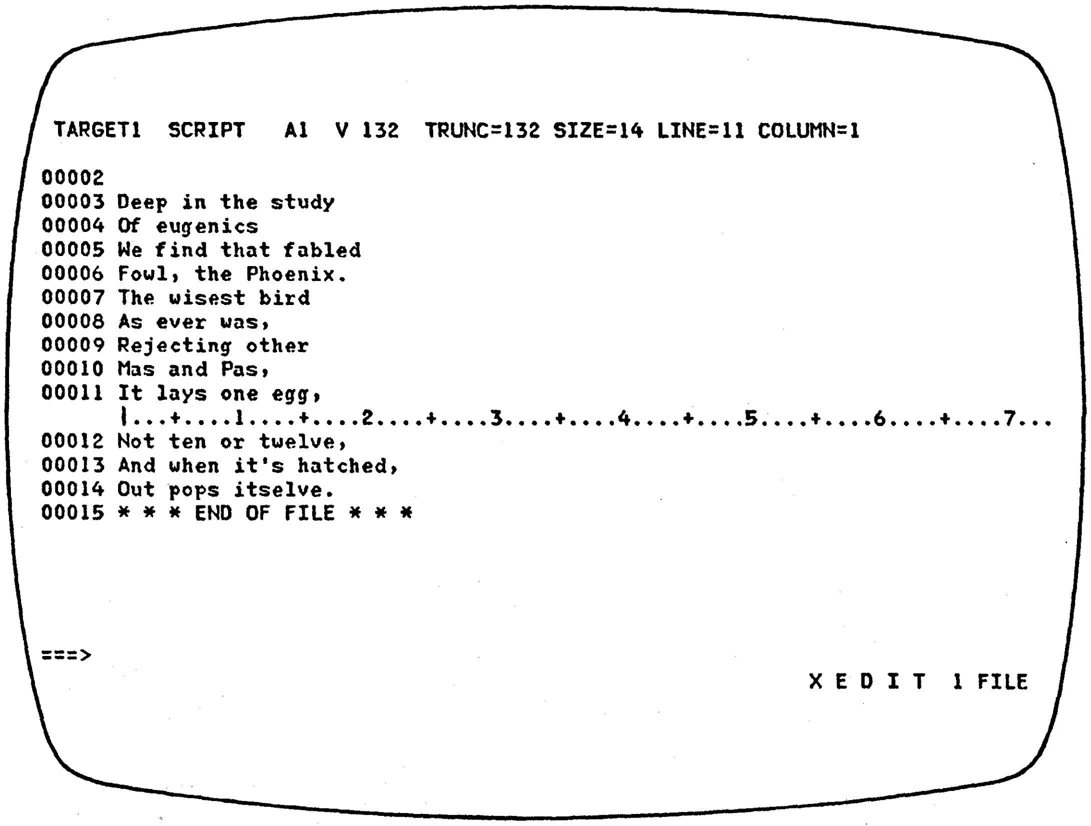
```
===> :10 ADD 5
```

The editor makes line 10 the new current line and then adds five lines to the file.

This method is equivalent to entering a target, pressing the ENTER key, entering the subcommand, and pressing the ENTER key. Typing both the target and the subcommand in the command line and pressing the ENTER key only once saves you time.



```
 TARGET1  SCRIPT   A1  V 132  TRUNC=132 SIZE=14 LINE=5 COLUMN=1



00000 * * * TOP OF FILE * * *
00001 THE PHOENIX
00002
00003 Deep in the study
00004 Of eugenics
00005 We find that fabled
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
00006 Fowl, the Phoenix.
00007 The wisest bird
00008 As ever was,
00009 Rejecting other
00010 Mas and Pas,
00011 It lays one egg,
00012 Not ten or twelve,
00013 And when it's hatched,
00014 Out pops itselve.
===> /egg/
                                                               X E D I T  1 FILE
```



```
 TARGET1  SCRIPT   A1  V 132  TRUNC=132 SIZE=14 LINE=11 COLUMN=1

00002
00003 Deep in the study
00004 Of eugenics
00005 We find that fabled
00006 Fowl, the Phoenix.
00007 The wisest bird
00008 As ever was,
00009 Rejecting other
00010 Mas and Pas,
00011 It lays one egg,
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
00012 Not ten or twelve,
00013 And when it's hatched,
00014 Out pops itselve.
00015 * * * END OF FILE * * *




===>
                                                               X E D I T  1 FILE
```

Figure 4-1. Using a Target to Move the Line Pointer

Using a Target as a Subcommand Operand

When a subcommand format shows that an operand may be specified as a target, the target is usually used to tell the editor how many lines the subcommand is to execute upon; in other words, it defines the scope of that subcommand's operation. For example, a format of the UPPERCAS subcommand is:

```
===> UPPERCAS target
```

This format means, "starting with the current line, translate all lowercase characters to uppercase, *up to, but not including,* the target line." The translation is not executed on the target line itself. After execution, the last line translated becomes the new current line.

Figure 4-2 is a before-and-after example of an UPPERCAS subcommand. When entered on the command line, any of the following subcommands would effect the translation shown in the bottom screen:

```
===> UPPERCAS   :14
```

(absolute line number)

```
===> UPPERCAS   +4
```

(relative displacement from current line)

```
===> UPPERCAS   .STOP
```

(line name previously assigned)

```
===> UPPERCAS   /son/
```

(string)

Types of Targets

Let's take a closer look at each of the ways to specify targets.

A Target as an Absolute Line Number

You can display line numbers in the prefix area by issuing the following subcommand:

```
===> SET NUMBER ON
```

An absolute line number is represented as a colon (:) followed by the line number, for example, :10.

The following examples illustrate targets specified as absolute line numbers:

```
===> :50
```

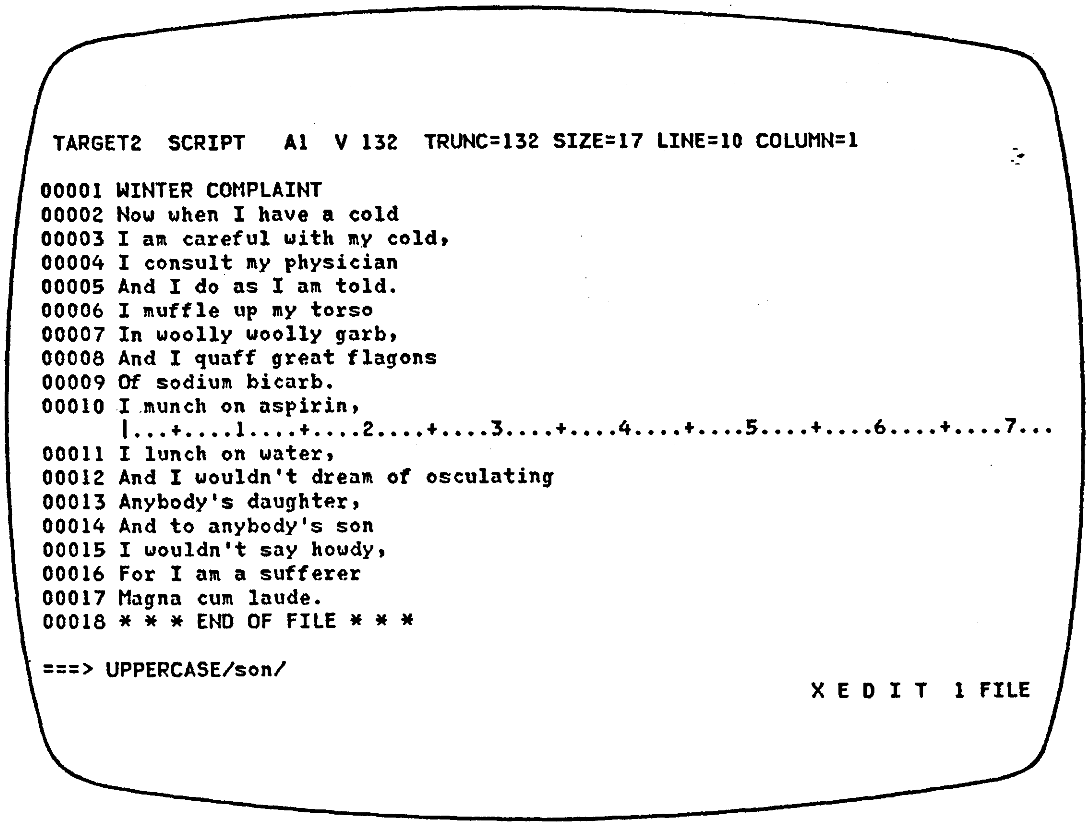
Make file line number 50 the new current line.

```
===> CHANGE /A/B/ :20
```

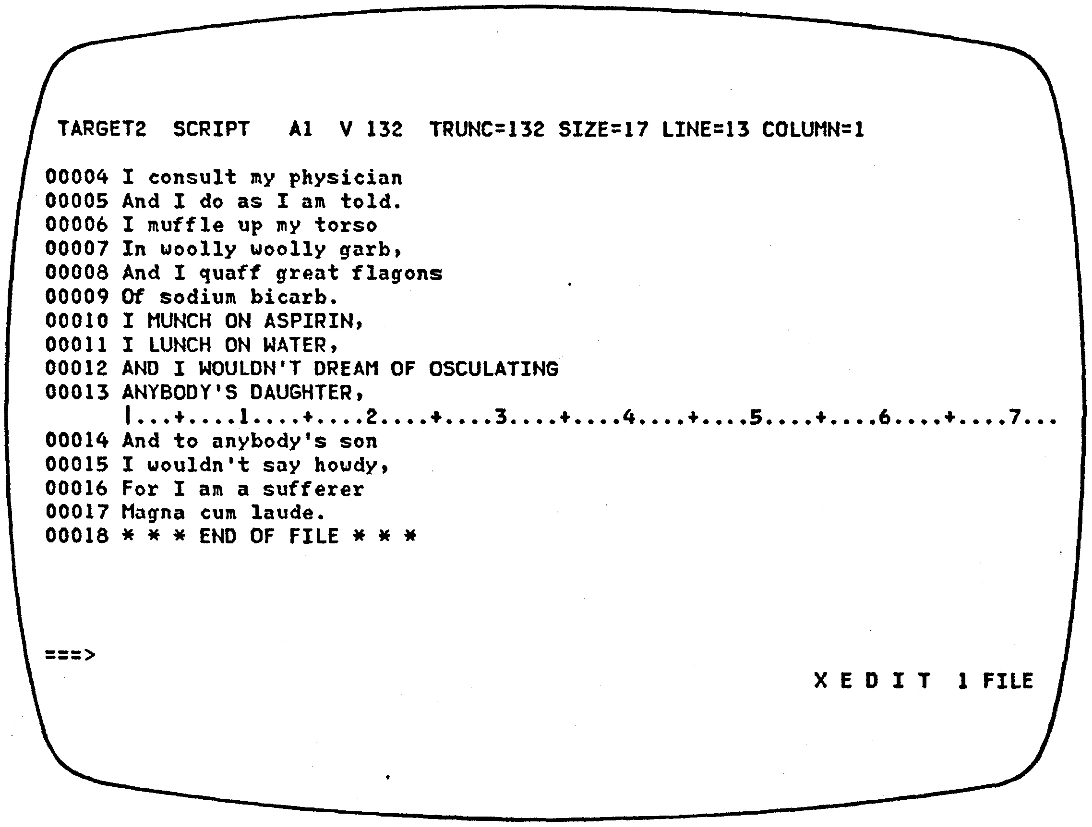
Beginning with the current line, change "A" to "B" in every line up to, but not including, line 20.

Figure 4-3 is a before-and-after example of a COUNT subcommand whose target operand is specified as an absolute line number. The COUNT subcommand (top screen) means, "beginning with the current line, count how many times the string 'cone' appears in all lines up to but not including line 14." The string is counted only if it appears in the file exactly the way it is specified in the subcommand (in lowercase).

When the ENTER key is pressed (bottom screen), notice that the last line searched (line 13) becomes the new current line, and the editor displays the message, "2 OCCURRENCES", in the message line.

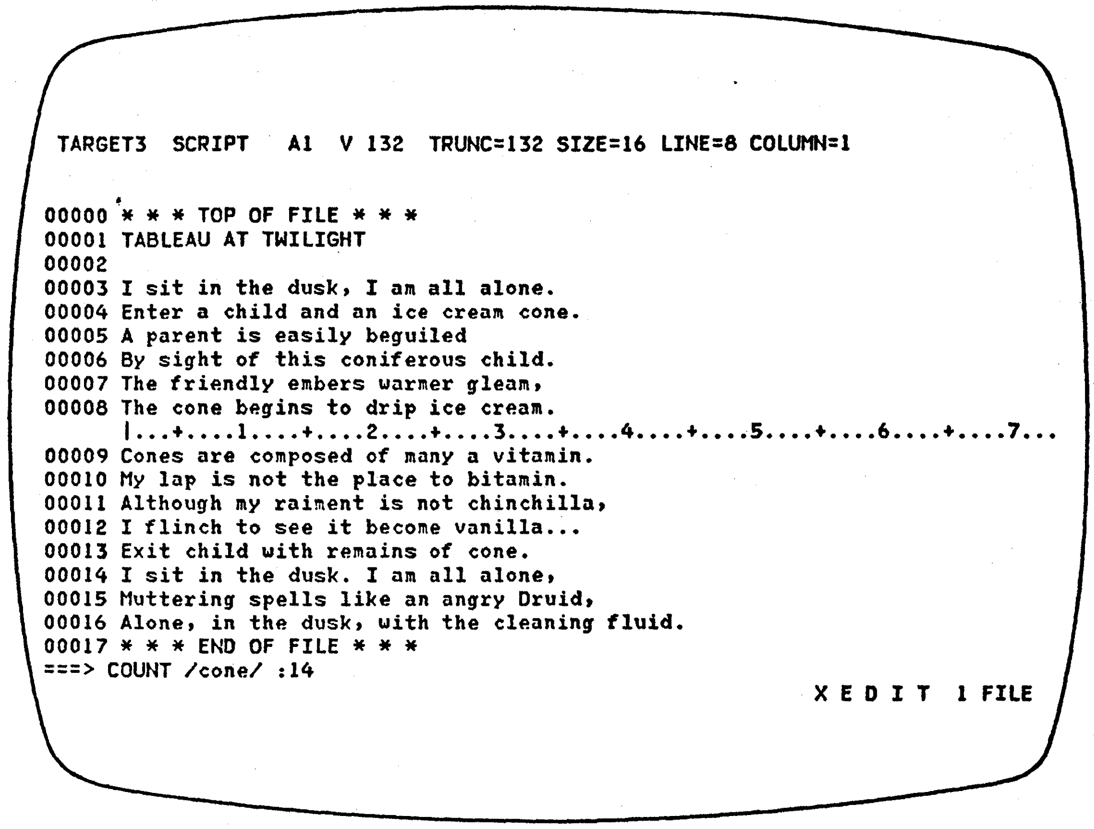


```
TARGET2  SCRIPT   A1  V 132  TRUNC=132 SIZE=17 LINE=10 COLUMN=1

00001 WINTER COMPLAINT
00002 Now when I have a cold
00003 I am careful with my cold,
00004 I consult my physician
00005 And I do as I am told.
00006 I muffle up my torso
00007 In woolly woolly garb,
00008 And I quaff great flagons
00009 Of sodium bicarb.
00010 I munch on aspirin,
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
00011 I lunch on water,
00012 And I wouldn't dream of osculating
00013 Anybody's daughter,
00014 And to anybody's son
00015 I wouldn't say howdy,
00016 For I am a sufferer
00017 Magna cum laude.
00018 * * * END OF FILE * * *

===> UPPERCASE/son/
                                                          X E D I T  1 FILE
```

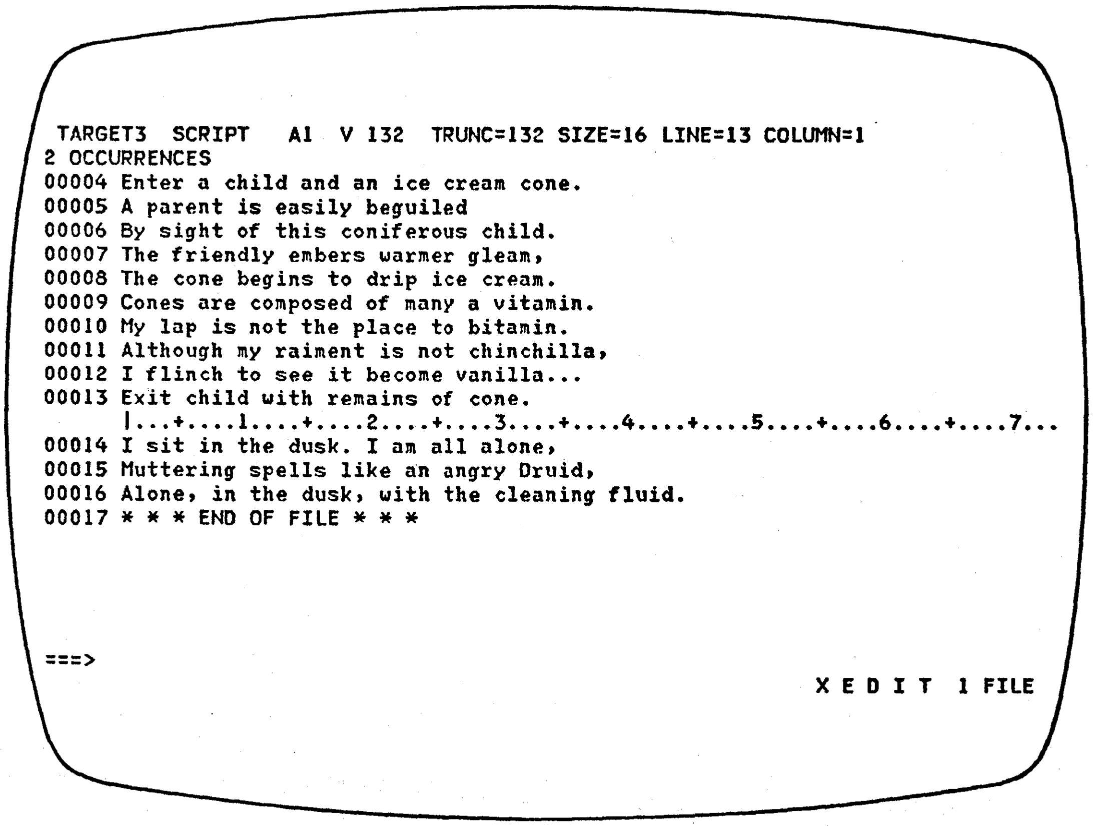


```
TARGET2  SCRIPT   A1  V 132  TRUNC=132 SIZE=17 LINE=13 COLUMN=1

00004 I consult my physician
00005 And I do as I am told.
00006 I muffle up my torso
00007 In woolly woolly garb,
00008 And I quaff great flagons
00009 Of sodium bicarb.
00010 I MUNCH ON ASPIRIN,
00011 I LUNCH ON WATER,
00012 AND I WOULDN'T DREAM OF OSCULATING
00013 ANYBODY'S DAUGHTER,
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
00014 And to anybody's son
00015 I wouldn't say howdy,
00016 For I am a sufferer
00017 Magna cum laude.
00018 * * * END OF FILE * * *



===>
                                                          X E D I T  1 FILE
```

Figure 4-2. Using a Target as a Subcommand Operand



```
 TARGET3  SCRIPT   A1  V 132  TRUNC=132 SIZE=16 LINE=8 COLUMN=1


00000 * * * TOP OF FILE * * *
00001 TABLEAU AT TWILIGHT
00002
00003 I sit in the dusk, I am all alone.
00004 Enter a child and an ice cream cone.
00005 A parent is easily beguiled
00006 By sight of this coniferous child.
00007 The friendly embers warmer gleam,
00008 The cone begins to drip ice cream.
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
00009 Cones are composed of many a vitamin.
00010 My lap is not the place to bitamin.
00011 Although my raiment is not chinchilla,
00012 I flinch to see it become vanilla...
00013 Exit child with remains of cone.
00014 I sit in the dusk. I am all alone,
00015 Muttering spells like an angry Druid,
00016 Alone, in the dusk, with the cleaning fluid.
00017 * * * END OF FILE * * *
===> COUNT /cone/ :14
                                                       X E D I T  1 FILE
```



```
 TARGET3  SCRIPT   A1  V 132  TRUNC=132 SIZE=16 LINE=13 COLUMN=1
2 OCCURRENCES
00004 Enter a child and an ice cream cone.
00005 A parent is easily beguiled
00006 By sight of this coniferous child.
00007 The friendly embers warmer gleam,
00008 The cone begins to drip ice cream.
00009 Cones are composed of many a vitamin.
00010 My lap is not the place to bitamin.
00011 Although my raiment is not chinchilla,
00012 I flinch to see it become vanilla...
00013 Exit child with remains of cone.
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
00014 I sit in the dusk. I am all alone,
00015 Muttering spells like an angry Druid,
00016 Alone, in the dusk, with the cleaning fluid.
00017 * * * END OF FILE * * *




===>
                                                       X E D I T  1 FILE
```

Figure 4-3. A Target as an Absolute Line Number

A Target as a Relative Displacement from the Current Line

A relative displacement from the current line is an integer that means the target is a number of lines, either forward or backward, from the current line. The number may be preceded by a plus or minus sign, which indicates a forward (+) or backward (-) displacement from the current line. If the sign is omitted, a plus (+) is assumed.

A relative displacement may also be specified as an asterisk (*), which means the TOP OF FILE (-*) or END OF FILE (+* or *) line. When an asterisk is specified as the target operand of a subcommand, the subcommand executes to the end (or top) of the file.

Examples:

```
===> +3
```

The target is three logical lines down (toward the end of the file) from the current line.

```
===> -5
```

The target is five logical lines up (toward the top of the file) from the current line.

```
===> +*
```

The target is the null END OF FILE (or END OF RANGE) line.

```
===> -*
```

The target is the null TOP OF FILE (or TOP OF RANGE) line.

```
===> COPY +3 :25
```

Copy three lines, starting with the current line, after line number 25.

In this example, two targets are specified. The first (+3) is a relative displacement from the current line; the second is an absolute line number.

```
===> DELETE *
```

Delete all lines from the current line to the end of the file.

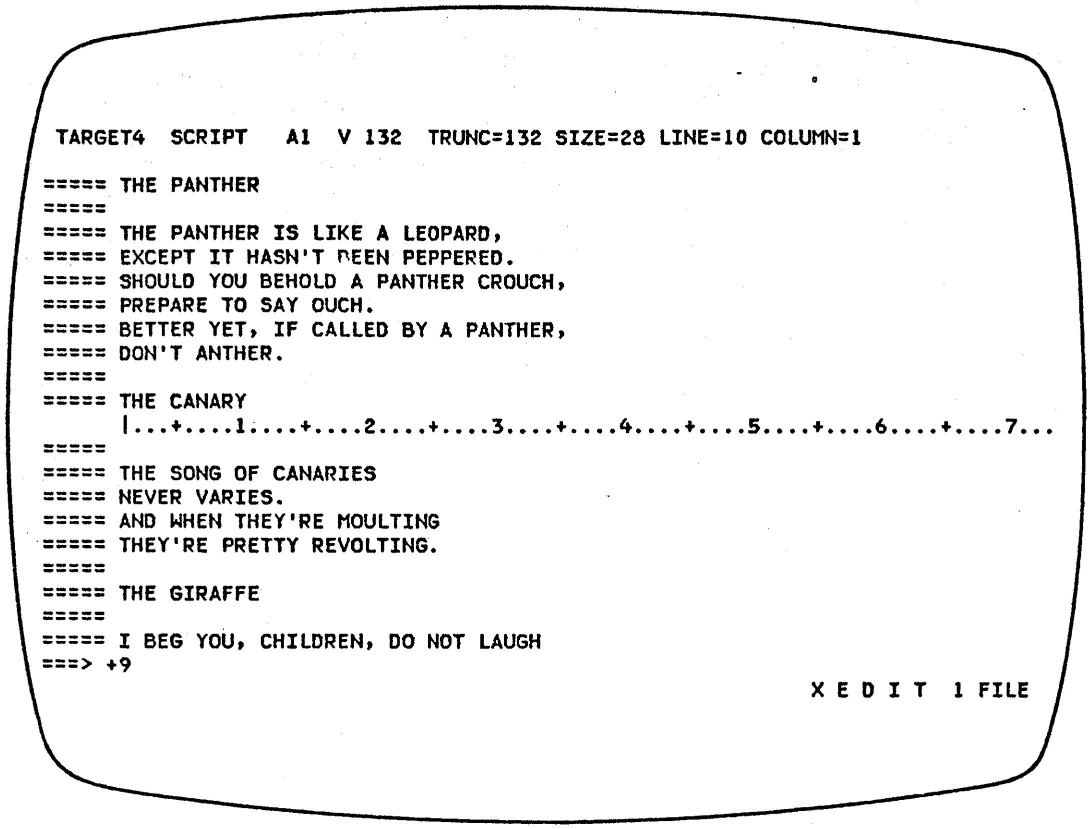
Figure 4-4 is a before-and-after example of a target specified as a relative displacement. The target typed in the command line, +9, means, "move the current line nine logical lines forward, toward the end of the file." Notice that line numbers do not have to be displayed in the prefix area to use this kind of target. However, the "LINE=" indicator in the file identification area shows the old (LINE=10) and new (LINE=19) numbers of the current line.

A Target as a Line Name

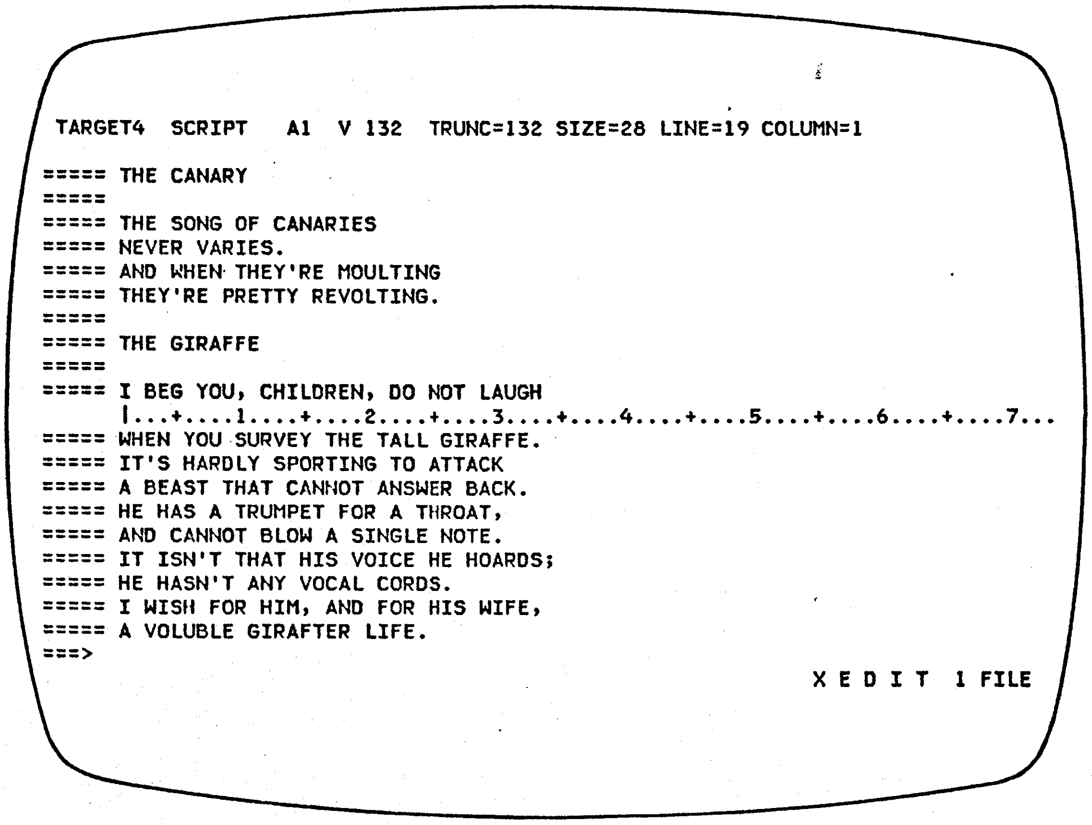
Any line in a file can be assigned a name of one to eight characters preceded by a period (.), for example, .PART2.

You can use either the SET POINT subcommand or the .xxxx prefix subcommand to define a name for a line. The SET POINT subcommand is used to define a name of one to eight characters, preceded by a period, to the current line. Using the .xxxx prefix subcommand allows you to define a name for any line in whose prefix area the name is entered; the name is one to four characters, preceded by a period.

Assigning a name to a line makes it unnecessary for you to look up its line number or determine its relative displacement. Although the absolute line number of any given line can change during an editing session as lines are added or deleted from the file, a name stays with a line for the entire editing session.



```
 TARGET4  SCRIPT   A1  V 132  TRUNC=132 SIZE=28 LINE=10 COLUMN=1

===== THE PANTHER
=====
===== THE PANTHER IS LIKE A LEOPARD,
===== EXCEPT IT HASN'T BEEN PEPPERED.
===== SHOULD YOU BEHOLD A PANTHER CROUCH,
===== PREPARE TO SAY OUCH.
===== BETTER YET, IF CALLED BY A PANTHER,
===== DON'T ANTHER.
=====
===== THE CANARY
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
=====
===== THE SONG OF CANARIES
===== NEVER VARIES.
===== AND WHEN THEY'RE MOULTING
===== THEY'RE PRETTY REVOLTING.
=====
===== THE GIRAFFE
=====
===== I BEG YOU, CHILDREN, DO NOT LAUGH
===> +9
                                                                 X E D I T  1 FILE
```



```
 TARGET4  SCRIPT   A1  V 132  TRUNC=132 SIZE=28 LINE=19 COLUMN=1

===== THE CANARY
=====
===== THE SONG OF CANARIES
===== NEVER VARIES.
===== AND WHEN THEY'RE MOULTING
===== THEY'RE PRETTY REVOLTING.
=====
===== THE GIRAFFE
=====
===== I BEG YOU, CHILDREN, DO NOT LAUGH
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== WHEN YOU SURVEY THE TALL GIRAFFE.
===== IT'S HARDLY SPORTING TO ATTACK
===== A BEAST THAT CANNOT ANSWER BACK.
===== HE HAS A TRUMPET FOR A THROAT,
===== AND CANNOT BLOW A SINGLE NOTE.
===== IT ISN'T THAT HIS VOICE HE HOARDS;
===== HE HASN'T ANY VOCAL CORDS.
===== I WISH FOR HIM, AND FOR HIS WIFE,
===== A VOLUBLE GIRAFTER LIFE.
===>
                                                                 X E D I T  1 FILE
```

Figure 4-4. A Target as a Relative Displacement

A line name is particularly useful if you plan to refer to a line many times during an editing session. You need assign the name only once; the line can then be referenced by its name at any time. It remains in effect only for the current editing session.

Examples:

1. Using the SET POINT subcommand to name a line:

   ```
   ===> SET POINT .PART2
   ```

   Assign the name ".PART2" to the current line.

   ```
   ===> TOP
   ```

   Move the line pointer to the TOP OF FILE line.

   ```
   ===> CHANGE /A/B/ .PART2
   ```

   Change "A" to "B" in every line, starting with the current line (in this case, the TOP OF FILE line) up to the line named ".PART2".

2. Using the .xxxx prefix subcommand to name a line:

   To use the .xxxx prefix subcommand, type a name preceded by a period in the prefix area of any line on the screen, as illustrated below:

   ```
   =====   data
   =====   data
   =====   data
   .STOP  This is the line I want to name.
   =====   data
   ```

   You can name any line on the screen with the .xxxx prefix subcommand; the line does not have to be the current line, as it does with the SET POINT subcommand. After the ENTER key is pressed the assigned name disappears from the prefix area and is replaced by equals signs or line numbers (depending on whether SET NUMBER ON or SET NUMBER OFF is in effect). Then, you can refer to the line by using its assigned name.

Examples of using lines that have been already named:

```
===> .STOP
```

Make the line named ".STOP" the new current line.

```
===> MOVE 1 .STOP
```

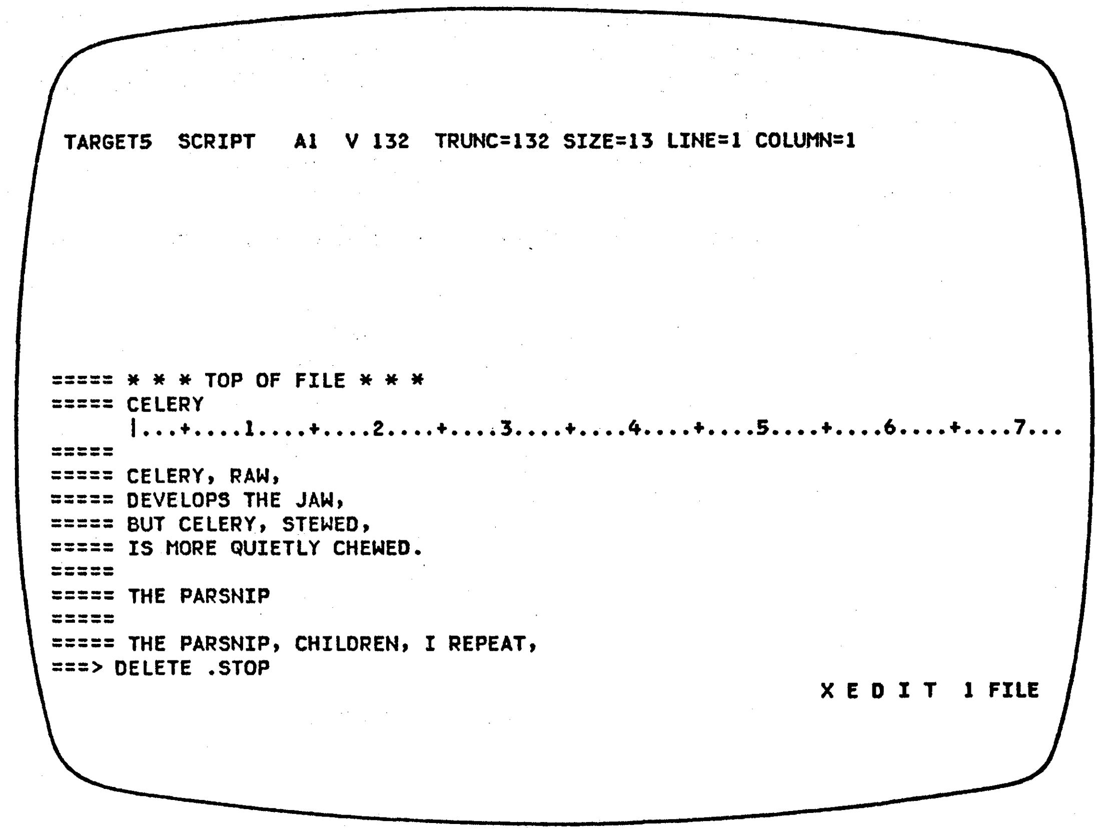
Move the current line after the line named ".STOP".

Note: After a name is assigned to a line, you must keep track of it. You can issue the subcommand QUERY POINT to display the name of the current line, or you can use QUERY POINT * to display all names that have been defined during the editing session.

Figure 4-5 is a before-and-after example of a DELETE subcommand that has its target operand specified as a line name. The line that contains "THE PARSNIP" was previously named ".STOP". The subcommand typed in the command line means, "beginning with the current line, delete lines up to but not including the line that has been assigned the name '.STOP'."

A Target as a Simple String Expression

A target can be specified as one or more characters, that is, a string, contained in a file line. The editor looks for the string, making the first line that contains it the target line.


```
 TARGET5  SCRIPT   A1  V 132  TRUNC=132 SIZE=13 LINE=1 COLUMN=1




===== * * * TOP OF FILE * * *
===== CELERY
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
=====
===== CELERY, RAW,
===== DEVELOPS THE JAW,
===== BUT CELERY, STEWED,
===== IS MORE QUIETLY CHEWED.
=====
===== THE PARSNIP
=====
===== THE PARSNIP, CHILDREN, I REPEAT,
===> DELETE .STOP
                                                        X E D I T  1 FILE
```

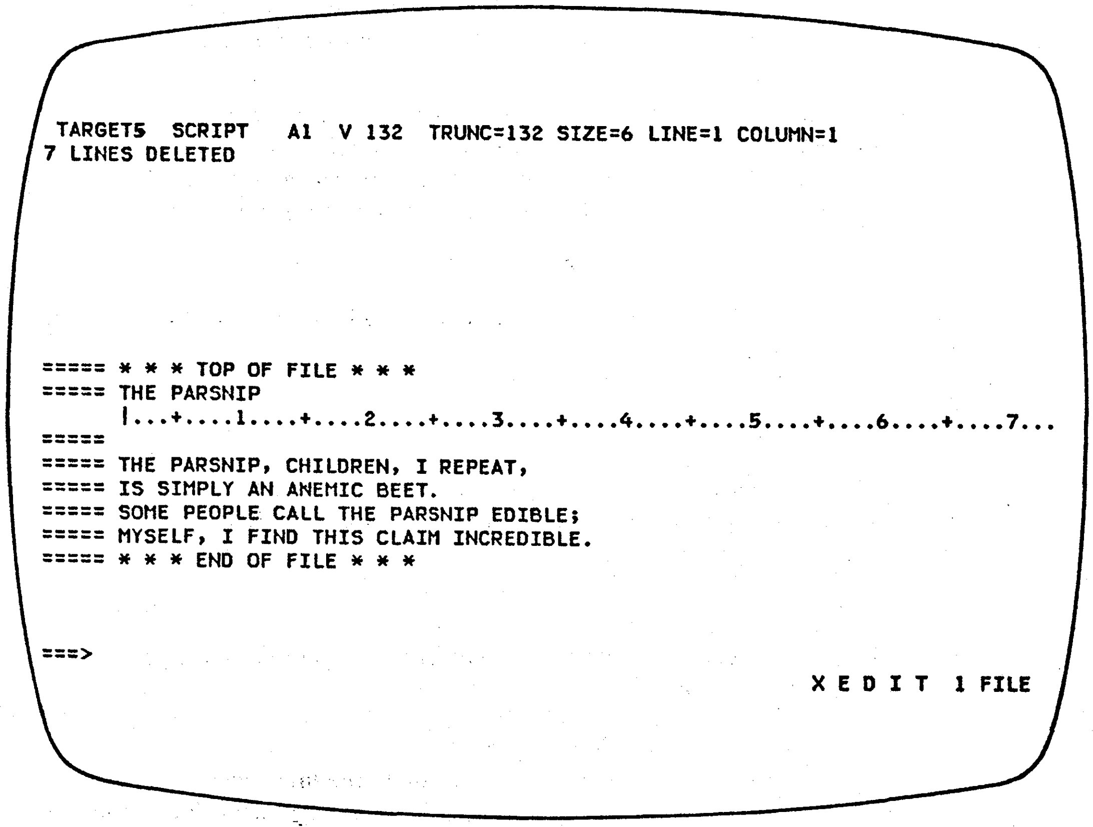

```
 TARGET5  SCRIPT   A1  V 132  TRUNC=132 SIZE=6 LINE=1 COLUMN=1
7 LINES DELETED




===== * * * TOP OF FILE * * *
===== THE PARSNIP
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
=====
===== THE PARSNIP, CHILDREN, I REPEAT,
===== IS SIMPLY AN ANEMIC BEET.
===== SOME PEOPLE CALL THE PARSNIP EDIBLE;
===== MYSELF, I FIND THIS CLAIM INCREDIBLE.
===== * * * END OF FILE * * *


===>
                                                        X E D I T  1 FILE
```

Figure 4-5. A Target as a Line Name

If the string target is specified alone or as the operand of a LOCATE subcommand, the line containing the string becomes the new current line. If the string target is an operand of one of the other XEDIT subcommands, the line that contains the string determines the scope of the subcommand's execution.

The string must be enclosed in delimiters, which can be any character that does not appear in the string itself.

For example, the following is a string target, entered alone on the command line:

```
===> /whatever/
```

This means, "beginning with the line following the current line, search for the string 'whatever' and make the line that contains it the new current line."

The following is an example of a string target used as the operand of a subcommand:

```
===> DELETE /whatever/
```

This means, "delete all lines from the current line up to, but not including, the line that contains 'whatever' ".

The simplest way to specify a string target, as shown above, is one or more characters surrounded by delimiters. You can also:

- determine the direction of the search
- search for a line that does *not* contain a given string
- search for any of several strings.

Specifying a Search Direction

By typing a plus (+) or minus (-) sign before a string target, you can tell the editor to search for a string in either a forward or backward direction from the current line.

A plus sign in front of a string target means that the search for the string starts at the line following the current line in a forward direction, toward the end of the file. If the string is found, the line that contains it becomes the new current line. If a sign is omitted, a plus is assumed. The following targets are equivalent:

`===> /whatever/` **and** `===> +/whatever/`

You can also specify that the search occur *backward* in the file by typing a minus sign before the string target.

For example:

```
===> -/whatever/
```

means, "search backward in the file, starting with the line preceding the current line, and make the line containing the string the new current line."

Let's look at some more examples:

```
===> DELETE /rosebud/
```

Delete lines beginning with the current line, up to but not including the line containing "rosebud".

```
===> COPY /daisy/ -/petunia/
```

Copy lines starting with the current line, up to the line containing "daisy", and insert them after the line containing "petunia", which is located in a backward direction from the current line.

```
===> PUT /Chapter2/
```

Put lines from the current line, up to the line that contains "Chapter2".

Using a "NOT" Symbol (¬)

You can precede any string target with a NOT symbol (¬), which means that the target is a line that does *not* contain the specified string. For example:

```
===>   ¬/Part Number/
```

Beginning with the line following the current line, locate a line that does not contain "Part Number" and make it the new current line.

```
===>  MOVE 1 ¬/Part Number/
```

Move the current line after the first line that does not contain "Part Number".

Using an "OR" Symbol (|)

A string target can comprise up to four strings, separated by an "OR" symbol, each enclosed in delimiters. The editor searches the file one line at a time. The first line that contains one of the specified strings becomes the current line. For example:

If a file contains the following lines:

```
===== apples
===== peaches
===== plums
===== pears
===== oranges
```

The following subcommand:

```
===> Locate /oranges/|/pears/|/peaches/
```

will make the following line current:

```
===== peaches
```

A Summary of Simple String Targets

You've seen how to specify a target as a single string, enclosed in delimiters. You've also seen how a plus or minus sign, a NOT symbol, and an OR symbol can be used to further define a string.

In addition, all of these features can be *combined* to define a single target, that is, a single string, enclosed in delimiters, can be preceded by a plus or minus sign and a NOT symbol, and up to four strings, separated by OR symbols, can be specified!

Furthermore, if the subcommand SET HEX ON is in effect, a string may be specified in hexadecimal notation, for example, /X'C3D4E2'/.

The following chart summarizes the format of a simple string expression:

```
[+|-][¬]/string1[/|[¬]/string2/]...
  1   2     3    4               5
```

1 The search direction is toward the end of the file (+) or toward the top of the file (-). If the sign is omitted, a plus (+) is assumed.

2 "NOT" symbol (locate something that is not the specified string)

3 Character (or hexadecimal) string, enclosed in delimiters

4 "OR" symbol (vertical bar) (locate one string or another)

5 Up to four strings may be specified.

Examples:

```
===> /horse/
```

searches downward in the file, beginning with the current line, for the first line that contains "horse" and makes it the current line.

```
===>  ¬/house/
```

searches downward in the file for the first line that does not contain "house" and makes it the current line.

```
===> /horse/|¬/house/
```

searches downward in the file for the first line that contains "horse" *and/or* does not contain "house."

```
===> -/X'C1'/|/X'C2'/
```

searches upward for the first line containing *either or both* of the strings specified here in hexadecimal (if SET HEX ON has been issued).

If SET HEX ON is in effect, the editor locates a line containing "A" or "B". If SET HEX OFF is in effect, the editor locates a line containing "X'C1'" or "X'C2'".

Figure 4-6 is a before-and-after example of a target specified as a simple string expression. The target typed in the command line means, "beginning with the line following the current line, search for a line that either does not contain 'Experience' or for a line that does contain 'experience', and make it the new current line."

A Target as a Complex String Expression

A complex string expression has the same format as a simple string expression (see above), but any string can be expressed as a "complex string", which is a string associated with one or more of the following SET subcommand options:

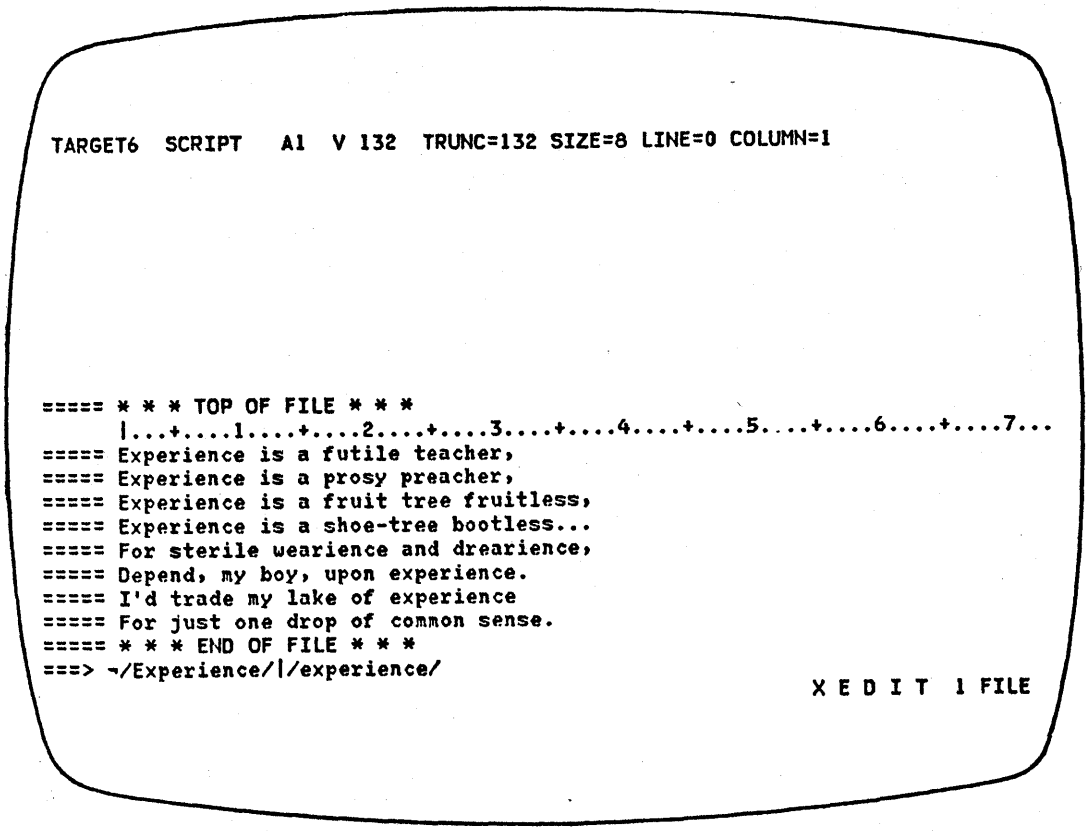
SET ARBCHAR
: allows you to specify only the beginning and end of a string, using an arbitrary character to represent all characters in the middle.

SET CASE
: allows you to specify whether or not the difference between uppercase and lowercase is to be significant in locating a string target.

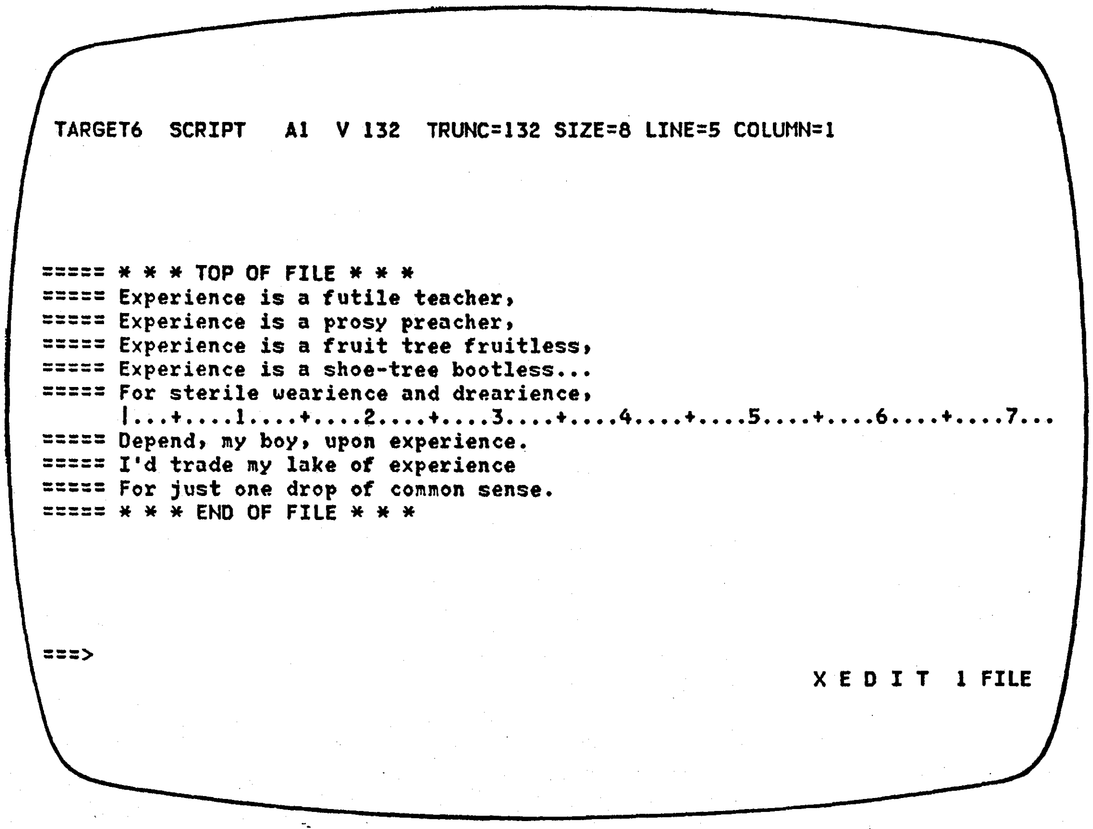
SET SPAN
: allows you to specify if a string target must be included in one file line or if it may span a specified number of lines.

SET VARBLANK
: allows you to control whether or not the number of blank characters between two words is significant in a target search.

You can use one or more of these options to suit your individual text processing needs. Each of the options is assigned an initial setting by the editor. You can alter the setting one or more times during an editing session by issuing the appropriate SET subcommand. (See the publication *VM/SP: System Product Editor Command and Macro Reference* for a complete description of these SET subcommand options.)



```
TARGET6  SCRIPT   A1  V 132  TRUNC=132 SIZE=8 LINE=0 COLUMN=1









===== * * * TOP OF FILE * * *
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== Experience is a futile teacher,
===== Experience is a prosy preacher,
===== Experience is a fruit tree fruitless,
===== Experience is a shoe-tree bootless...
===== For sterile wearience and drearience,
===== Depend, my boy, upon experience.
===== I'd trade my lake of experience
===== For just one drop of common sense.
===== * * * END OF FILE * * *
===> ¬/Experience/|/experience/
                                                          X E D I T  1 FILE
```



```
TARGET6  SCRIPT   A1  V 132  TRUNC=132 SIZE=8 LINE=5 COLUMN=1




===== * * * TOP OF FILE * * *
===== Experience is a futile teacher,
===== Experience is a prosy preacher,
===== Experience is a fruit tree fruitless,
===== Experience is a shoe-tree bootless...
===== For sterile wearience and drearience,
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== Depend, my boy, upon experience.
===== I'd trade my lake of experience
===== For just one drop of common sense.
===== * * * END OF FILE * * *






===>
                                                          X E D I T  1 FILE
```

Figure 4-6. A Target as a Simple String Expression

Using a Target with SET ARBCHAR

When SET ARBCHAR ON is in effect, you can use a dollar sign ($), which is the default arbitrary character, to represent all characters between the beginning and end of a string target.

Examples:

```
===> /air$plane/
```

The beginning of the string is "air"; the end of the string is "plane". The dollar sign is the arbitrary character and represents any characters between "air" and "plane". This string target causes the editor to locate either of the following file lines, and makes current whichever line comes first:

```
===== The airplane landed.
===== Cold air surrounded the plane.
```

Using a Target with SET CASE

You can specify whether the editor is to respect or ignore the difference between uppercase and lowercase representations of alphabetic letters by using the SET CASE subcommand.

The following subcommand tells the editor that uppercase and lowercase representations of the same letter do not match:

```
===> SET CASE MIXED RESPECT
```

For example, if the file contains the following line:

```
===== The Text Editor
```

The following string target will *not* locate that line:

```
===> /the text editor/
```

On the other hand, the following subcommand tells the editor to ignore the difference between uppercase and lowercase:

```
===> SET CASE MIXED IGNORE
```

With this setup, in the example above, the line would be located.

Using a Target with SET SPAN

Usually, a string must be included in a single file line in order to be located. You can use the SET SPAN subcommand to specify that a string target may span a specified number of lines and still be located. The line that contains the beginning of the string becomes the new current line.

In a text file, like a SCRIPT file, a blank separates each file line. The following subcommand tells the editor that a string target may span two lines, separated from each other by a blank:

```
===> SET SPAN ON BLANK 2
```

The string target

```
===> /twigs to probe/
```

would locate in the file:

```
===== Woodpecker finches of the Galapagos Islands use twigs
===== to probe holes in tree trunks for edible insects.
```

The string "twigs to probe" begins on one line and ends on the next.

Using a Target with SET VARBLANK

The SET VARBLANK subcommand can be used to control whether or not the number of blank characters between two words is significant in a target search.

SET VARBLANK ON means that the number of blanks between two words can vary; the number of intervening blanks specified in a string target does not have to be equal to the number in the file.

For example:

```
===> /the house/
```

would locate either of the following lines in the file:

```
===== the        house
===== the house
```

If SET VARBLANK OFF is in effect (the initial setting), the number of blanks between two words is significant in a target search. In the above example, only the second line would be located.

Combining the SET Options

You can tailor the SET options, ARBCHAR, CASE, SPAN, and VARBLANK to meet your particular text processing needs. For example, with SET ARBCHAR ON, SET CASE MIXED IGNORE, SET SPAN ON BLANK 2, and SET VARBLANK ON, you can:

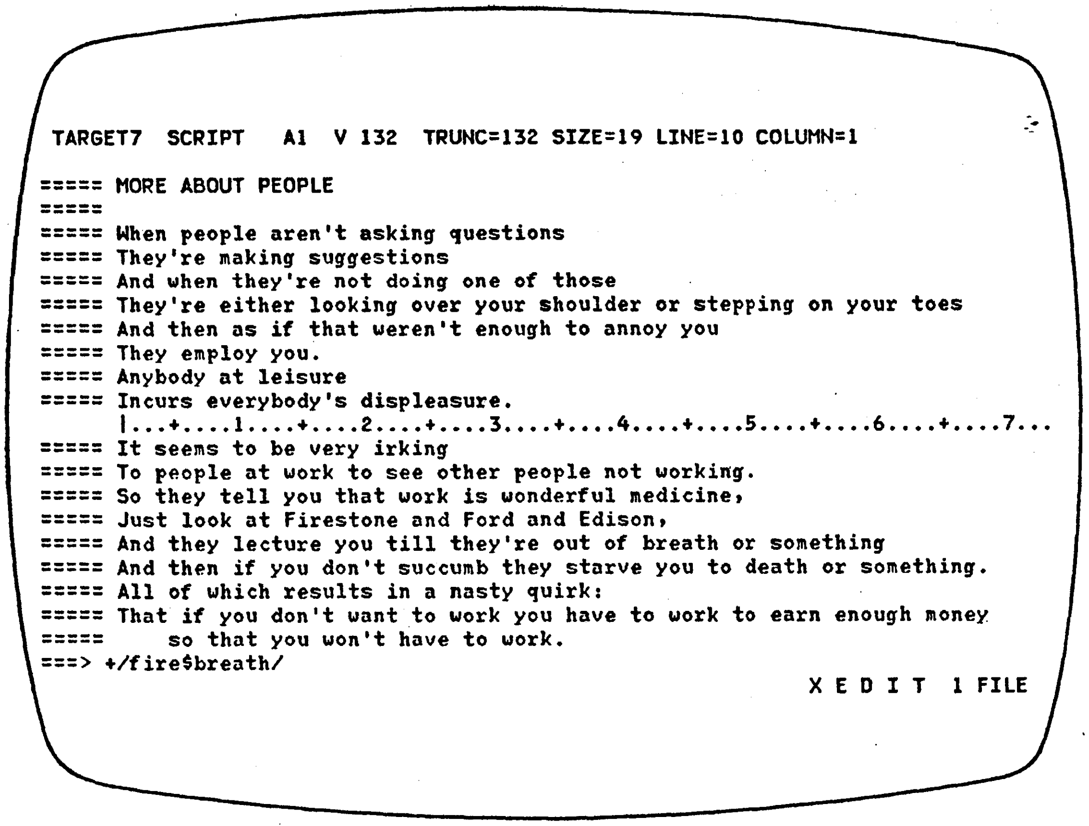
- specify only the beginning and end of a string target
- locate a string whether it is in uppercase or lowercase
- allow the string target to locate a string that starts on one line and ends on another
- disregard the number of intervening of blanks between two words.

Figure 4-7 is a before-and-after example of using a target specified as a complex string expression.

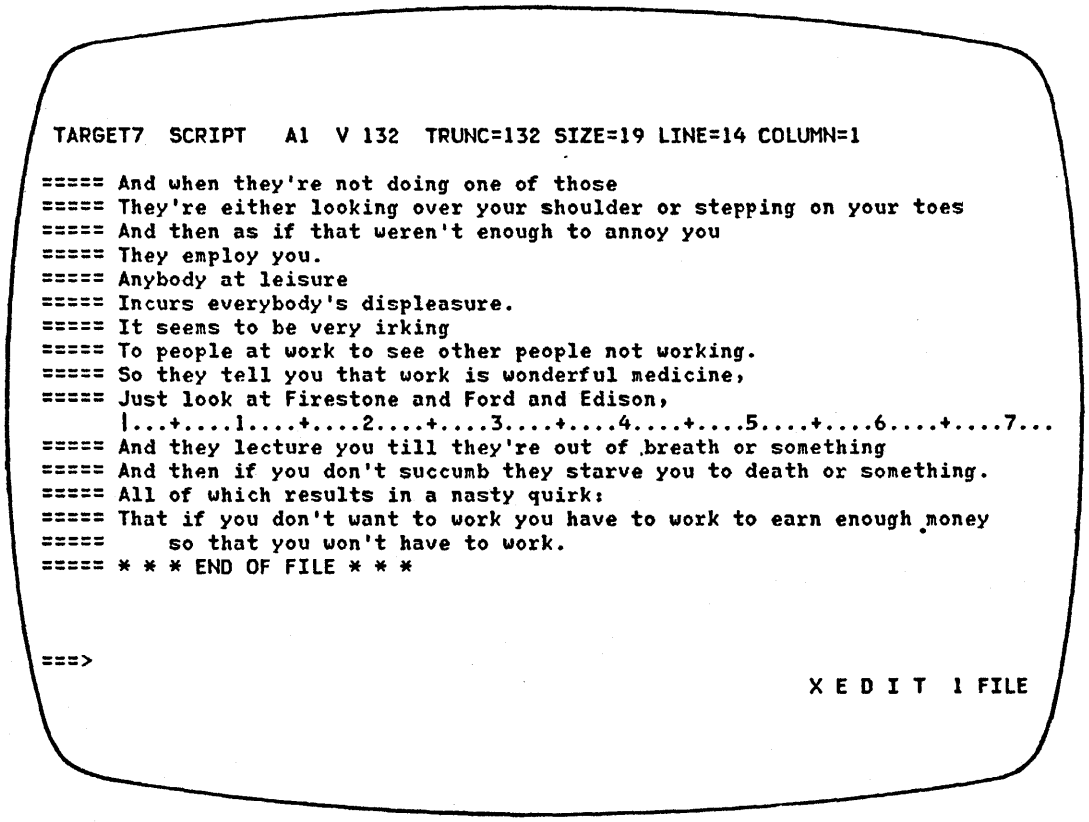
The following subcommands were issued:

```
===> SET ARBCHAR ON $
===> SET CASE MIXED IGNORE
===> SET SPAN ON BLANK 2
```

The string target typed in the command line locates the line shown in the bottom screen. The ARBCHAR option allows the beginning and end to be specified; the CASE option allows the string to be specified in lowercase even though it appears in the file in both uppercase and lowercase; the SPAN option allows the beginning and end of the string to be located on two consecutive lines.



```
TARGET7  SCRIPT   A1  V 132  TRUNC=132 SIZE=19 LINE=10 COLUMN=1

===== MORE ABOUT PEOPLE
=====
===== When people aren't asking questions
===== They're making suggestions
===== And when they're not doing one of those
===== They're either looking over your shoulder or stepping on your toes
===== And then as if that weren't enough to annoy you
===== They employ you.
===== Anybody at leisure
===== Incurs everybody's displeasure.
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== It seems to be very irking
===== To people at work to see other people not working.
===== So they tell you that work is wonderful medicine,
===== Just look at Firestone and Ford and Edison,
===== And they lecture you till they're out of breath or something
===== And then if you don't succumb they starve you to death or something.
===== All of which results in a nasty quirk:
===== That if you don't want to work you have to work to earn enough money
=====     so that you won't have to work.
===> +/fire$breath/
                                                          X E D I T  1 FILE
```



```
TARGET7  SCRIPT   A1  V 132  TRUNC=132 SIZE=19 LINE=14 COLUMN=1

===== And when they're not doing one of those
===== They're either looking over your shoulder or stepping on your toes
===== And then as if that weren't enough to annoy you
===== They employ you.
===== Anybody at leisure
===== Incurs everybody's displeasure.
===== It seems to be very irking
===== To people at work to see other people not working.
===== So they tell you that work is wonderful medicine,
===== Just look at Firestone and Ford and Edison,
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== And they lecture you till they're out of breath or something
===== And then if you don't succumb they starve you to death or something.
===== All of which results in a nasty quirk:
===== That if you don't want to work you have to work to earn enough money
=====     so that you won't have to work.
===== * * * END OF FILE * * *


===>
                                                          X E D I T  1 FILE
```

Figure 4-7. A Target as a Complex String Expression

Using Column-Targets

The targets discussed so far effect line pointer movement, that is, if the editor locates the target, the line pointer is moved. However, the column pointer is not moved. Furthermore, if a target is expressed as a string, only the first occurrence of the string is located in a line.

The CLOCATE subcommand operates on a specialized operand called a column-target. This subcommand is used to locate all occurrences of a string throughout a file and to move the column pointer. The format of the CLOCATE subcommand is as follows:

```
===> CLOCATE column-target
```

where the column-target can be expressed as an absolute column number, a relative displacement from the current column, or a string expression.

The following examples show the various ways to express a column-target. Notice how the column pointer moves after each subcommand is executed.

Current Line:

```
===== John Keats studied medicine and practiced as an apothecary.
      |...+....1....+....2....+....3....+....4....+....5....+....6....
===> CLOCATE :6
```

(absolute column number)

```
===== John Keats studied medicine and practiced as an apothecary.
      <...+|...1....+....2....+....3....+....4....+....5....+....6....
```

Current Line:

```
===== James Joyce was a school teacher in Dublin.
      |...+....1....+....2....+....3....+....4....+....5....+....6....
===> CLOCATE +6
```

(relative column number)

```
===== James Joyce was a school teacher in Dublin.
      <...+.|..1....+....2....+....3....+....4....+....5....+....6....
```

Current Line:

```
===== Herman Melville worked as a customs inspector in N.Y.C.
      |...+....1....+....2....+....3....+....4....+....5....+....6....
===> CLOCATE /customs/
===== Herman Melville worked as a customs inspector in N.Y.C.
      <...+....1....+....2....+...|3....+....4....+....5....+....6....
```

Current Line:

```
===== Charles Dickens served as a law clerk and was a reporter.
      |...+....1....+....2....+....3....+....4....+....5....+....6....
===> CLOCATE /reporter/|/clerk/
===== Charles Dickens served as a law clerk and was a reporter.
      <...+....1....+....2....+....3....+....4....+...|5....+....6....
```

The CLOCATE subcommand scans the file, starting with the column pointer in the current line, for the specified column target, and moves the column pointer to the target, if it is located. In addition, the line pointer is moved (if necessary), so that CLOCATE can be used successively to locate all occurrences of a string in a file.

CLOCATE is also necessary because various subcommands perform their operations based on the position of the column pointer. The CLOCATE subcommand is first used to position the column pointer; then one of the following subcommands can be used:

CAPPEND
: Appends text to the end of the current line, and moves the column pointer under the appended text.

CDELETE
: Deletes one or more characters from the current line, starting at the column pointer, up to a column-target.

CFIRST
: Moves the column pointer to the beginning of the line.

CINSERT
: Inserts character(s) in a line, starting at the column pointer.

CLAST
: Moves the column pointer to the end of the line.

CLOCATE
: Moves the column pointer to a specified column-target.

COVERLAY
: Replaces characters in the current line, starting at the column pointer; blanks in the operand do not overlay characters in the file line.

CREPLACE
: Replaces characters in the current line, starting at the column pointer; characters can be replaced with blanks.

These subcommands are discussed in detail in the publication *VM/SP: System Product Editor Command and Macro Reference.* Column-targets are discussed in that book in the "Usage Notes" section of the CLOCATE subcommand.

The following examples illustrate how to use the CLOCATE and CDELETE subcommands to delete a word:

```
===== If anything can go wrong, it will.
      |...+....1....+....2....+....3....+....4....+....5....+....6....
===> CLOCATE / wrong/
```

(Move column pointer under first character of string to be deleted.)

```
===== If anything can go wrong, it will.
      <...+....1....+...|2....+....3....+....4....+....5....+....6....
===> CDELETE /,/
```

(Delete from column pointer up to the comma.)

```
===== If anything can go, it will.
      <...+....1....+...|2....+....3....+....4....+....5....+....6....
```

Chapter 5: Editing Multiple Files

The XEDIT Subcommand

When you issue the CMS command XEDIT, a copy of the specified file is brought into virtual storage, where it remains until you issue a FILE or QUIT subcommand. In other words, the XEDIT *command* brings one file at a time into storage. By entering the XEDIT *subcommand* during an editing session, you can bring more than one file into virtual storage at a time.

The format of the XEDIT subcommand is identical to that of the XEDIT command and is as follows:

```
===> Xedit [fn[ft[fm]]][(options...[)]]
```

For a complete description of the XEDIT subcommand operands, refer to the publication *VM/SP: System Product Editor Command and Macro Reference.*

Creating a Ring of Files in Storage

Multiple files are kept in virtual storage in a "ring." Each time you issue an XEDIT subcommand with a new fileid, a file is added to the ring and becomes the current file, which is the file that is displayed.

A file remains in the ring until a FILE or QUIT subcommand is issued for that file; then the preceding file in the ring is displayed. The number of files you can edit simultaneously is limited only by your virtual storage size.

Figure 5-1 illustrates a ring of files in storage.

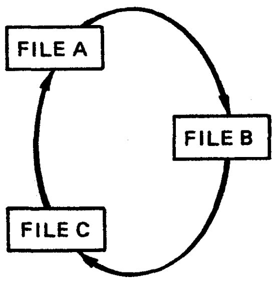


Figure 5-1. A Ring of Files in Storage

By issuing the following subcommand, you can display the number of files in the ring and the file identification line of each file:

```
===>  QUERY RING
```

Editing the Files in the Ring

The order in which you can edit the files in the ring depends on how you specify the XEDIT subcommand:

- If you issue the XEDIT subcommand without operands, the next file in the ring appears on the screen. (See Figure 5-2, Part 1.) Therefore, a series of XEDIT subcommands issued without operands allows you to switch from the first file to the second, the second to the third, and so forth, all the way around the ring and back to the first file.
- You can alter this sequence by issuing the XEDIT subcommand with the fileid of a file in the ring. The specified file becomes the current file and appears on

the screen, regardless of its relative position in the ring. (See Figure 5-2, Part 2.)

- If you issue an XEDIT subcommand with a fileid of a file that is not already in the ring, that file is added to the ring just after the current file and is displayed. (See Figure 5-2, Part 3.)
- If the XEDIT subcommand is issued with a fileid and the file does not exist, that file is created, added to the ring, and displayed.

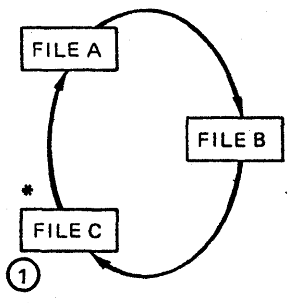
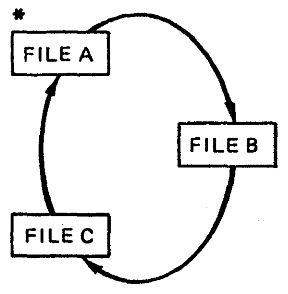
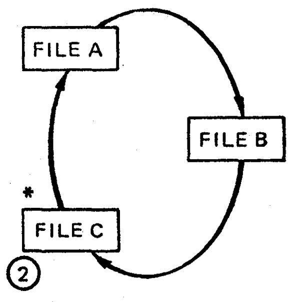
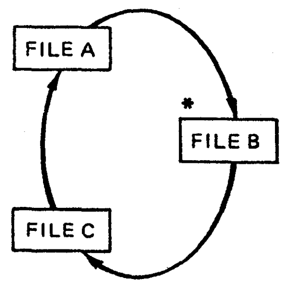
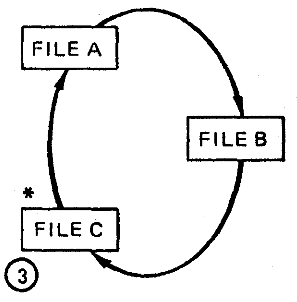
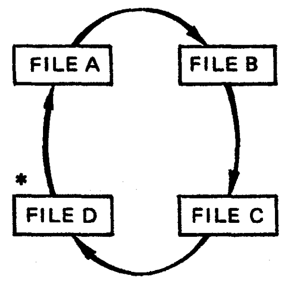
Current File (*)	XEDIT Subcommand	New Current File (*)
	===> XEDIT	
	===> XEDIT FILEB	
	===> XEDIT FILED	

Figure 5-2. Editing Files in the Ring

Ending an Editing Session

When you are finished editing a particular file, you can issue a FILE or QUIT subcommand for that file. The file is removed from the ring, and the previous file in the ring is displayed.

To end the editing session for all of the files and return control to CMS, use the CANCEL macro, whose format is as follows:

```
===>  CANCEL
```

Issuing the CANCEL macro is equivalent to issuing a QUIT subcommand for each file in the ring. If any of the files were modified, the usual warning message is displayed for each of those files:

```
FILE HAS BEEN CHANGED.  USE QQUIT TO QUIT ANYWAY.
```

You can then issue either QQUIT or FILE.

If none of the files being canceled were modified, control is immediately returned to CMS.

Multiple Logical Screens

Up until now, we have been discussing editing multiple files with one file, the current file in the ring, displayed at a time. By using the SET SCREEN subcommand, you can divide the physical screen into multiple logical screens. You can display a different file from the ring in each logical screen, or you can display multiple views of the same file.

Each logical screen looks and functions like the physical screen. Each one becomes, in effect, an independent terminal with its own file identification line, command line, message line, and status area.

SET SCREEN Subcommand

The format of the SET SCREEN subcommand is:

`[SET]`	`SCReen n`
	`Size n1 [n2]...`

where:

n

specifies the number of logical screens that the physical screen is to be divided into. The logical screens are equal in size but must be at least five lines long.

(Remember that the editor uses four lines (if the scale is displayed), so that it is possible to have a logical screen with only one data line. Therefore, certain subcommands will have no effect. For example, the INPUT subcommand puts you in input mode, but the input zone contains no lines. The FORWARD and BACKWARD subcommands have no effect.)

n1 n2...

specifies the number of lines for each logical screen, thus making it possible to have logical screens of different sizes.

The initial setting of the SCREEN option is SCREEN SIZE *n,* where *n* is the physical screen size.

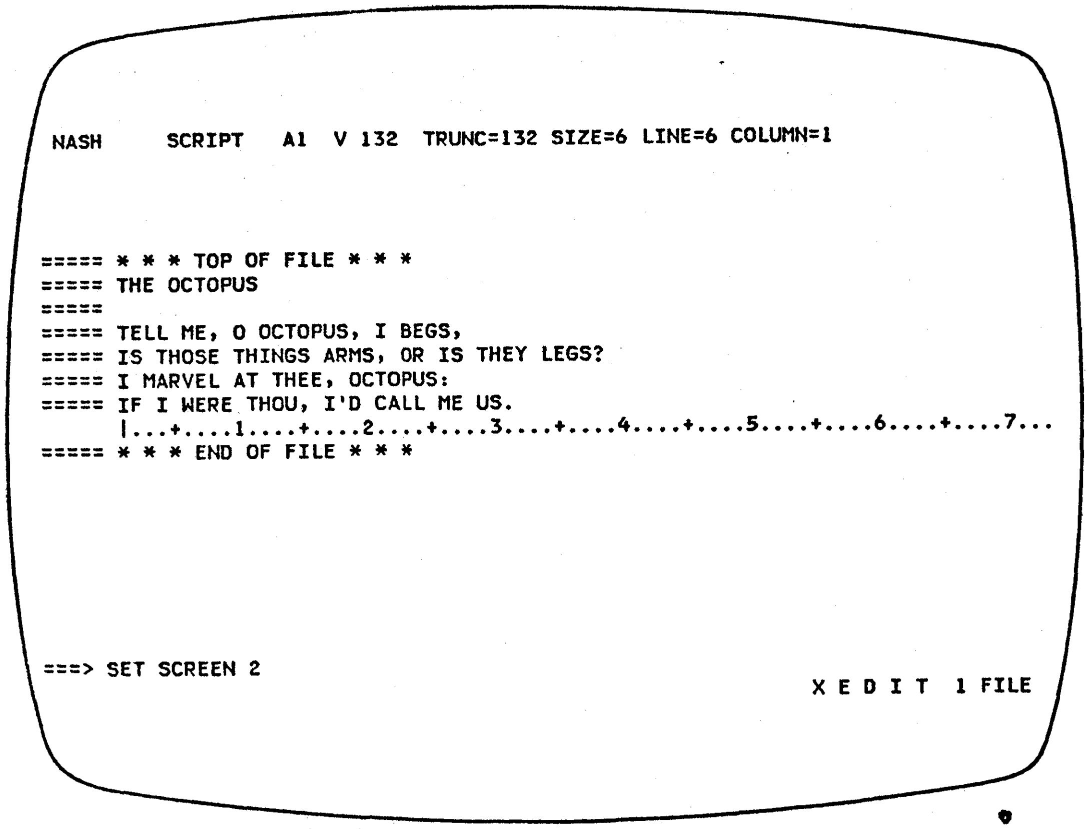
To return to the initial setting, issue the following subcommand:

```
===>  SET SCREEN 1
```

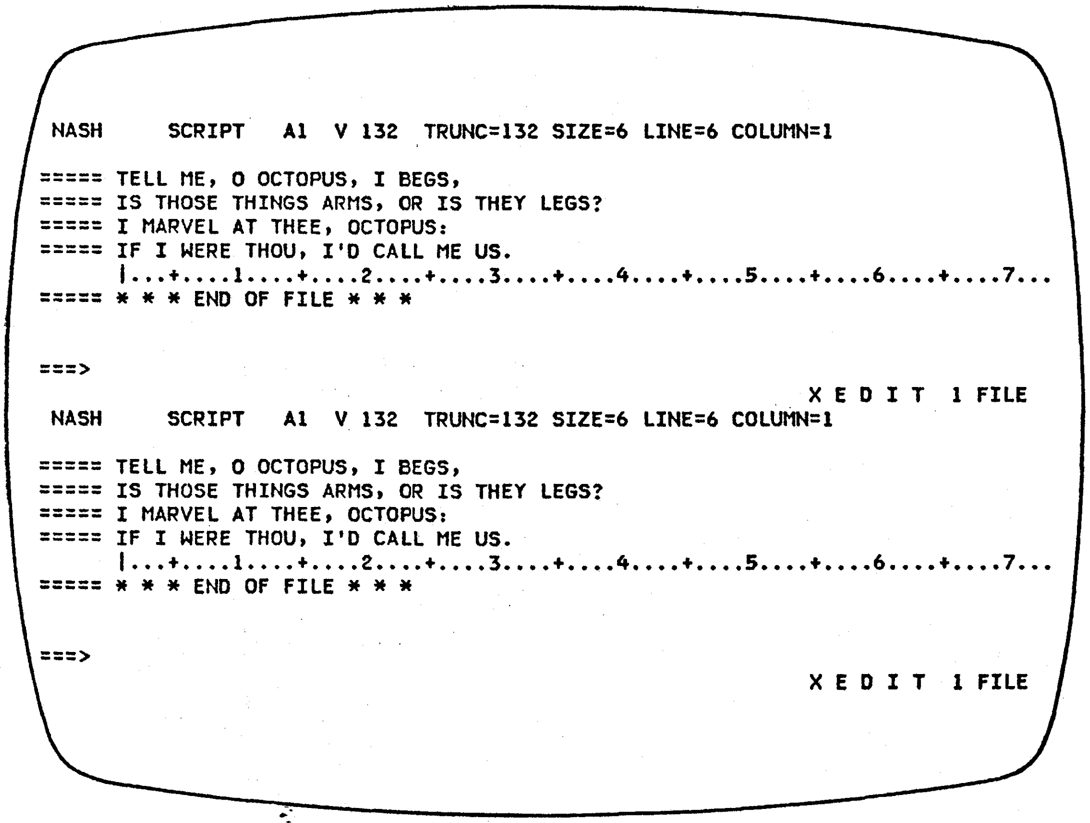
Multiple Views of the Same File

If only one file is in virtual storage and you issue a SET SCREEN subcommand, identical views of the file appear on the screen.

Figure 5-3 is a before-and-after example of a SET SCREEN subcommand that creates two views of the same file.



```
 NASH     SCRIPT   A1  V 132  TRUNC=132 SIZE=6 LINE=6 COLUMN=1



===== * * * TOP OF FILE * * *
===== THE OCTOPUS
=====
===== TELL ME, O OCTOPUS, I BEGS,
===== IS THOSE THINGS ARMS, OR IS THEY LEGS?
===== I MARVEL AT THEE, OCTOPUS:
===== IF I WERE THOU, I'D CALL ME US.
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== * * * END OF FILE * * *








===> SET SCREEN 2
                                                                X E D I T  1 FILE
```



```
 NASH     SCRIPT   A1  V 132  TRUNC=132 SIZE=6 LINE=6 COLUMN=1

===== TELL ME, O OCTOPUS, I BEGS,
===== IS THOSE THINGS ARMS, OR IS THEY LEGS?
===== I MARVEL AT THEE, OCTOPUS:
===== IF I WERE THOU, I'D CALL ME US.
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== * * * END OF FILE * * *


===>
                                                                X E D I T  1 FILE
 NASH     SCRIPT   A1  V 132  TRUNC=132 SIZE=6 LINE=6 COLUMN=1

===== TELL ME, O OCTOPUS, I BEGS,
===== IS THOSE THINGS ARMS, OR IS THEY LEGS?
===== I MARVEL AT THEE, OCTOPUS:
===== IF I WERE THOU, I'D CALL ME US.
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== * * * END OF FILE * * *


===>
                                                                X E D I T  1 FILE
```

Figure 5-3. Multiple Views of the Same File

Making Changes From Multiple Views of the Same File

You can edit a file by typing over the data in any of the views, and by entering subcommands in any of the command lines and prefix areas. Changes made to the file from one logical screen are reflected immediately in all screens.

However, subcommands that control the screen display, for example, FORWARD, affect only that screen from which they were issued. Therefore, you can see different parts of a file at the same time.

Similarly, PF keys assigned to screen movement subcommands are executed only on the view that contains the cursor when the PF key is pressed.

Order of Processing

You can type over the data, type subcommands on the command line, and type prefix subcommands in the prefix area of all views of a file before pressing the ENTER key.

The editor processes requests typed on different screens in the following order:

1. Changes typed over the data in all the views are made first, starting at the top view.
2. Prefix subcommands are executed in all views, starting at the top view.

   You can type related prefix subcommands in different logical screens, even when they display different parts of the file. For example, you can type a "C" (copy) prefix subcommand in one view, and a "P" (preceding) prefix subcommand in the next.
3. Subcommands typed on the command lines are executed, starting at the top view.

Multiple Views of Different Files

When multiple files are being edited and you issue a SET SCREEN subcommand that increases the number of logical screens, the additional screens are immediately filled with files selected from the ring.

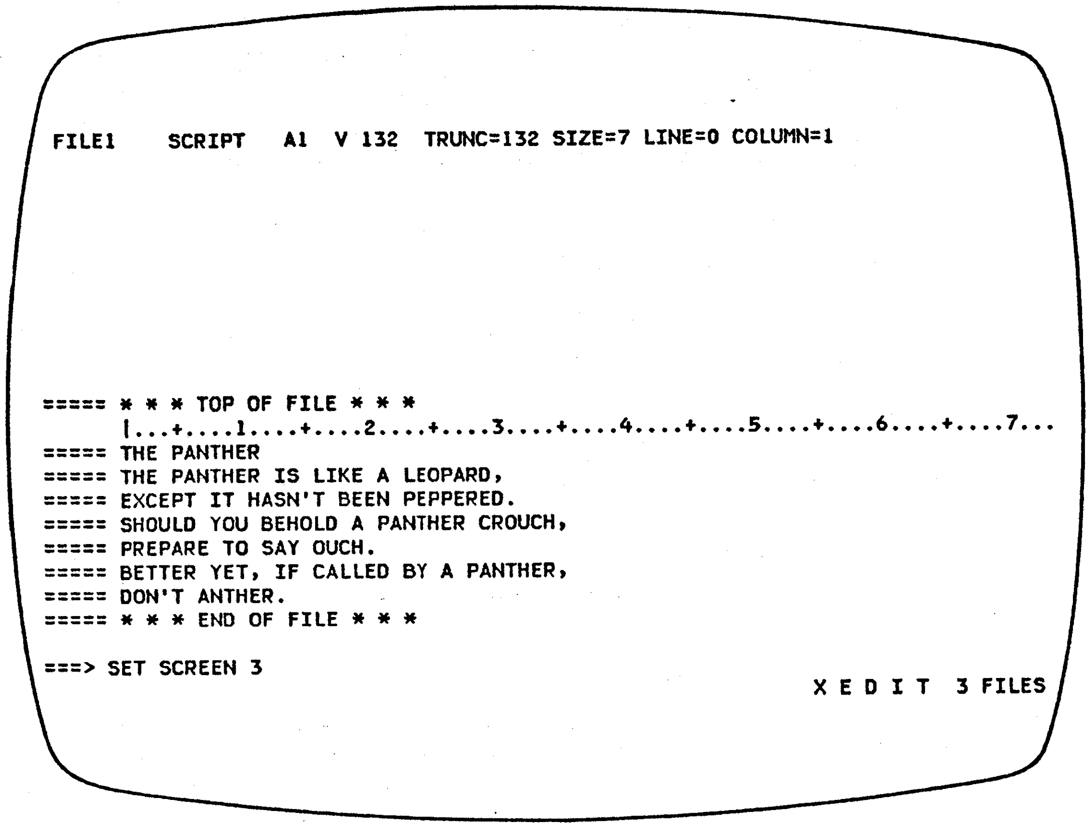
Figure 5-4 illustrates how additional logical screens are filled with files from the ring. The ring of files contains files named FILE1, FILE2, and FILE3; the current file is FILE1. The SET SCREEN subcommand shown in the top screen causes the rest of the files to be displayed.

If a SET SCREEN subcommand decreases the number of logical screens, files are displayed as long as logical screens are available, starting at the top of the screen. Those files for which logical screens are not available are removed from the display.

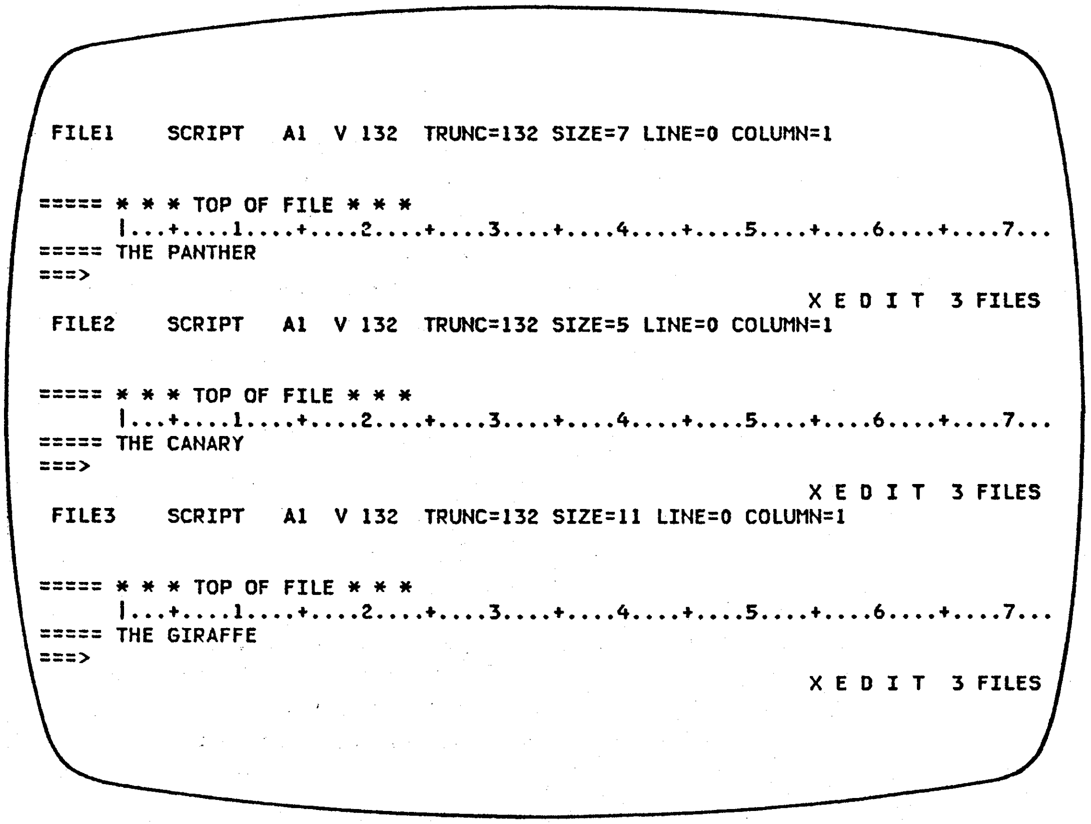
Issuing an XEDIT subcommand from one of multiple screens is just like issuing it when there is only one screen. It does not affect the other logical screens.

Issued without a filename, the XEDIT subcommand selects the next file in the ring; issued with a fileid, it creates a file, adds a file to the ring, or selects a file that is already in the ring. In all cases, the file is displayed only on the screen from which the XEDIT subcommand was issued.

The status area of all the screens displays the number of files in virtual storage, not the number of screens.



```
FILE1    SCRIPT   A1  V 132  TRUNC=132 SIZE=7 LINE=0 COLUMN=1









===== * * * TOP OF FILE * * *
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== THE PANTHER
===== THE PANTHER IS LIKE A LEOPARD,
===== EXCEPT IT HASN'T BEEN PEPPERED.
===== SHOULD YOU BEHOLD A PANTHER CROUCH,
===== PREPARE TO SAY OUCH.
===== BETTER YET, IF CALLED BY A PANTHER,
===== DON'T ANTHER.
===== * * * END OF FILE * * *

===> SET SCREEN 3
                                                                X E D I T  3 FILES
```



```
FILE1    SCRIPT   A1  V 132  TRUNC=132 SIZE=7 LINE=0 COLUMN=1


===== * * * TOP OF FILE * * *
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== THE PANTHER
===>
                                                                X E D I T  3 FILES
FILE2    SCRIPT   A1  V 132  TRUNC=132 SIZE=5 LINE=0 COLUMN=1


===== * * * TOP OF FILE * * *
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== THE CANARY
===>
                                                                X E D I T  3 FILES
FILE3    SCRIPT   A1  V 132  TRUNC=132 SIZE=11 LINE=0 COLUMN=1


===== * * * TOP OF FILE * * *
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== THE GIRAFFE
===>
                                                                X E D I T  3 FILES
```

Figure 5-4. Multiple Views of Different Files

Chapter 6: Tailoring the Screen

By using the following SET subcommand options, you can tailor the full screen layout to suit your preferences:

SET PREFIX
SET CMDLINE
SET CURLINE
SET SCALE
SET TABLINE

For a complete description of these options, refer to the SET subcommand description in the publication *VM/SP: System Product Editor Command and Macro Reference.*

The areas of the screen that can be changed are discussed below.

Prefix Area

Use the SET PREFIX subcommand to control the display of the prefix area. You can display the prefix area on the left or the right side of the screen, or you can remove the prefix area from the display. Initially, the prefix area is displayed on the left.

Command Line

Use the SET CMDLINE subcommand to move the command line to the same line as the message line (the second line of the screen) or to the last line of the screen. Initially, the command line is the last two lines of the screen. If you move the command line to the message line or the last line, the status area is not displayed.

Current Line

Use the SET CURLINE subcommand to define a specified line of the screen as the current line. Initially, the current line is in the middle of the screen.

Remember that the editor uses the first two lines of the screen, for the file identification line and the message line. Therefore, if you want the current line to be the first available screen line, use the subcommand SET CURLINE ON 3.

One reason you might want to change the position of the current line is to vary the size of the input zone. When you issue an INPUT subcommand, the editor provides an input zone between the current line and the command line. To get a larger input zone, move the current line higher on the screen; to get a smaller input zone, move it lower on the screen.

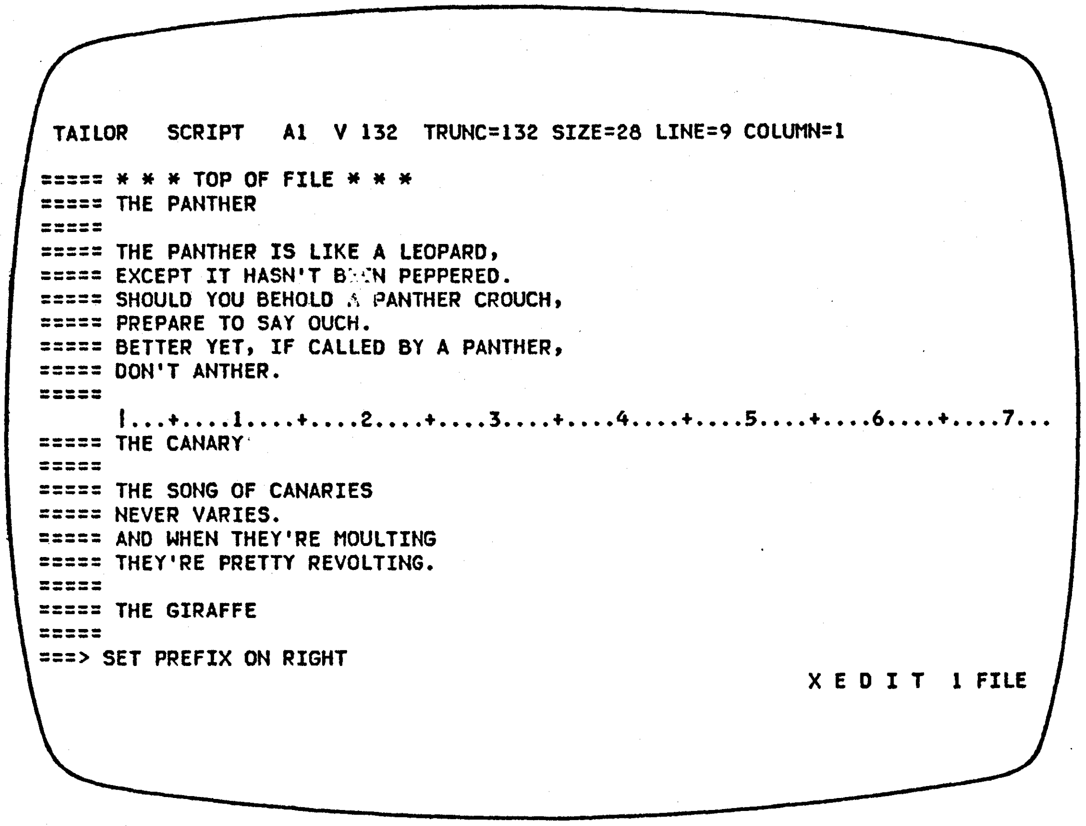
Scale

Use the SET SCALE subcommand to move the scale to a specified line, or to remove the scale from the display. Initially, the scale is positioned under the current line. If you move the current line, you probably also will want to move the scale.

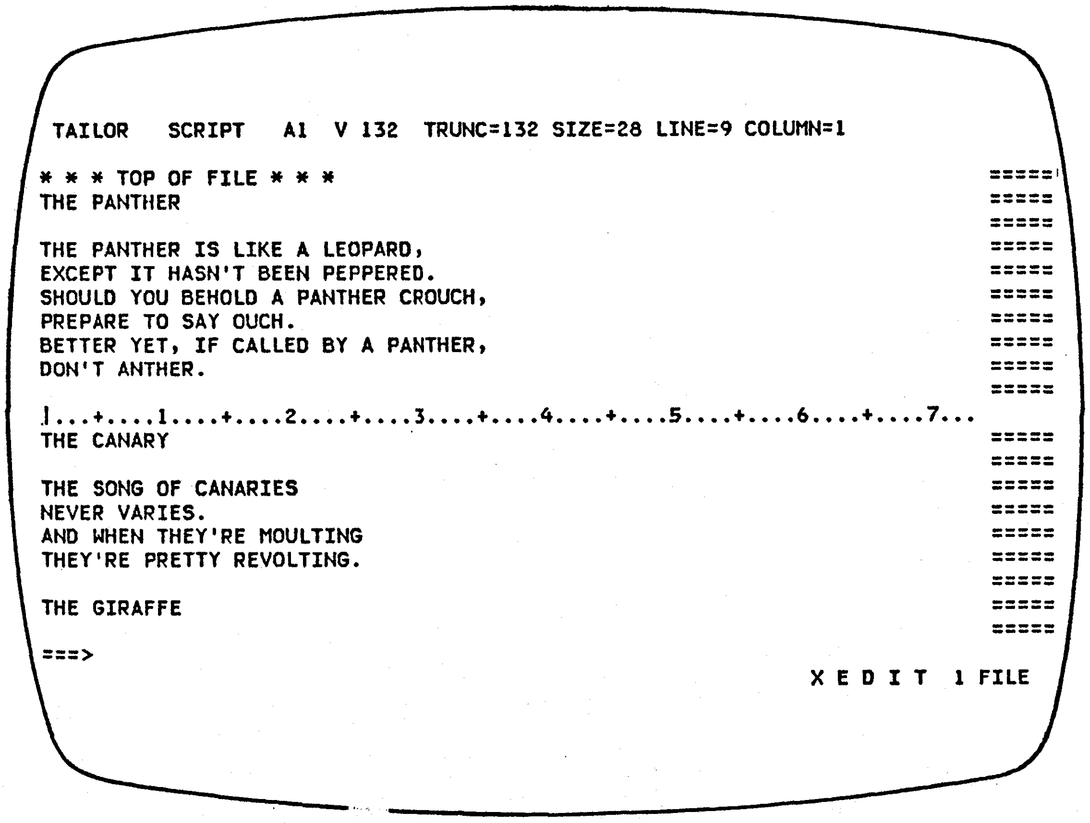
Tab Line

Use the SET TABLINE subcommand to display, on a specified line, a "T" in every tab column, according to the current tab settings (as defined by the SET TABS subcommand). Initially, a tab line is not displayed. If you change the tab settings during an editing session, the tab line will reflect that change, that is, the "T"s will be placed in the new tab columns.

Figures 6-1 through 6-5 illustrate how the subcommands discussed above are used to tailor the screen. Notice how the screen changes when the subcommand shown in the command line of each screen is executed.

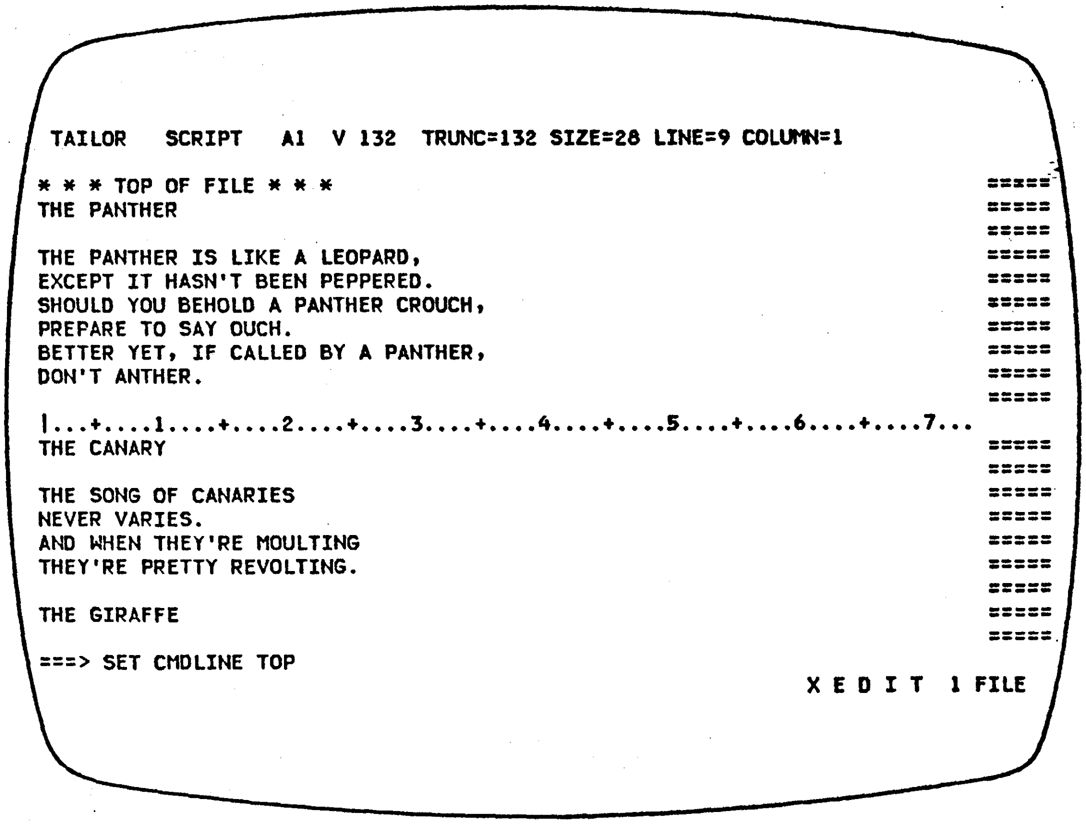


```
TAILOR   SCRIPT   A1  V 132  TRUNC=132 SIZE=28 LINE=9 COLUMN=1

===== * * * TOP OF FILE * * *
===== THE PANTHER
=====
===== THE PANTHER IS LIKE A LEOPARD,
===== EXCEPT IT HASN'T BEEN PEPPERED.
===== SHOULD YOU BEHOLD A PANTHER CROUCH,
===== PREPARE TO SAY OUCH.
===== BETTER YET, IF CALLED BY A PANTHER,
===== DON'T ANTHER.
=====
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== THE CANARY
=====
===== THE SONG OF CANARIES
===== NEVER VARIES.
===== AND WHEN THEY'RE MOULTING
===== THEY'RE PRETTY REVOLTING.
=====
===== THE GIRAFFE
=====
===> SET PREFIX ON RIGHT
                                                                   X E D I T  1 FILE
```

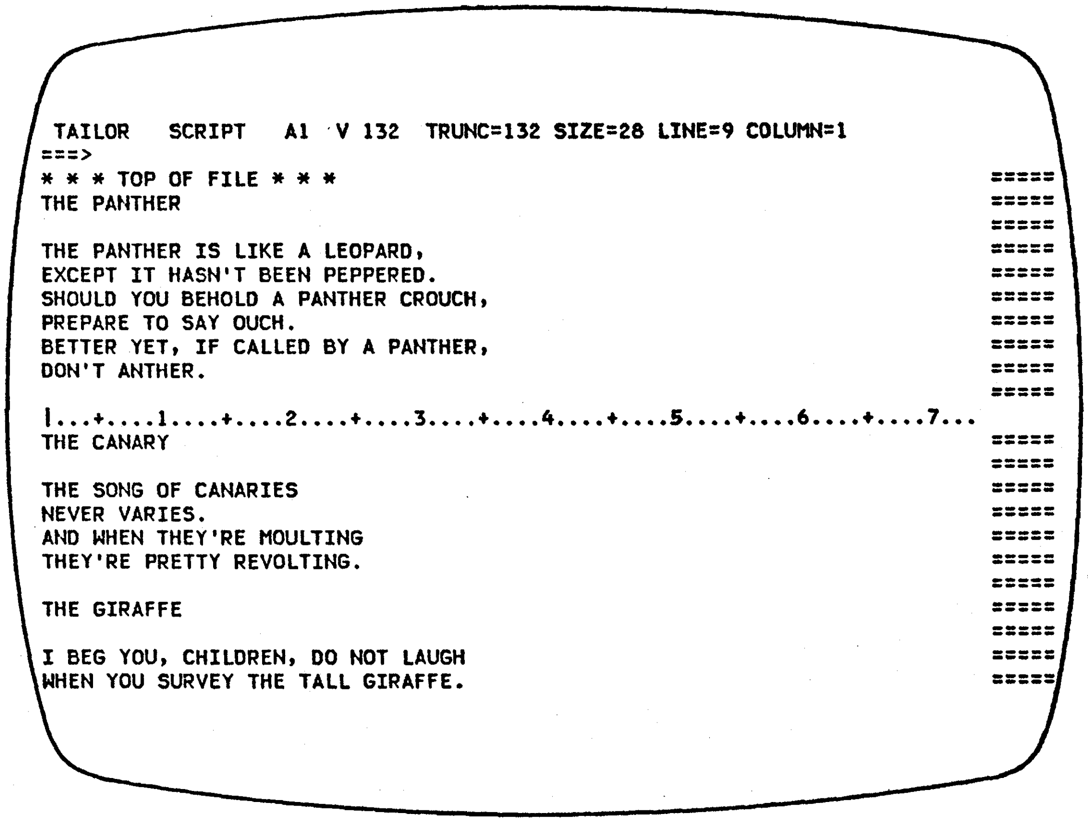


```
TAILOR   SCRIPT   A1  V 132  TRUNC=132 SIZE=28 LINE=9 COLUMN=1

* * * TOP OF FILE * * *                                                        =====
THE PANTHER                                                                    =====
                                                                               =====
THE PANTHER IS LIKE A LEOPARD,                                                 =====
EXCEPT IT HASN'T BEEN PEPPERED.                                                =====
SHOULD YOU BEHOLD A PANTHER CROUCH,                                            =====
PREPARE TO SAY OUCH.                                                           =====
BETTER YET, IF CALLED BY A PANTHER,                                            =====
DON'T ANTHER.                                                                  =====
                                                                               =====
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
THE CANARY                                                                     =====
                                                                               =====
THE SONG OF CANARIES                                                           =====
NEVER VARIES.                                                                  =====
AND WHEN THEY'RE MOULTING                                                      =====
THEY'RE PRETTY REVOLTING.                                                      =====
                                                                               =====
THE GIRAFFE                                                                    =====
                                                                               =====
===>
                                                                   X E D I T  1 FILE
```

Figure 6-1. The SET PREFIX Subcommand - "Before" and "After"

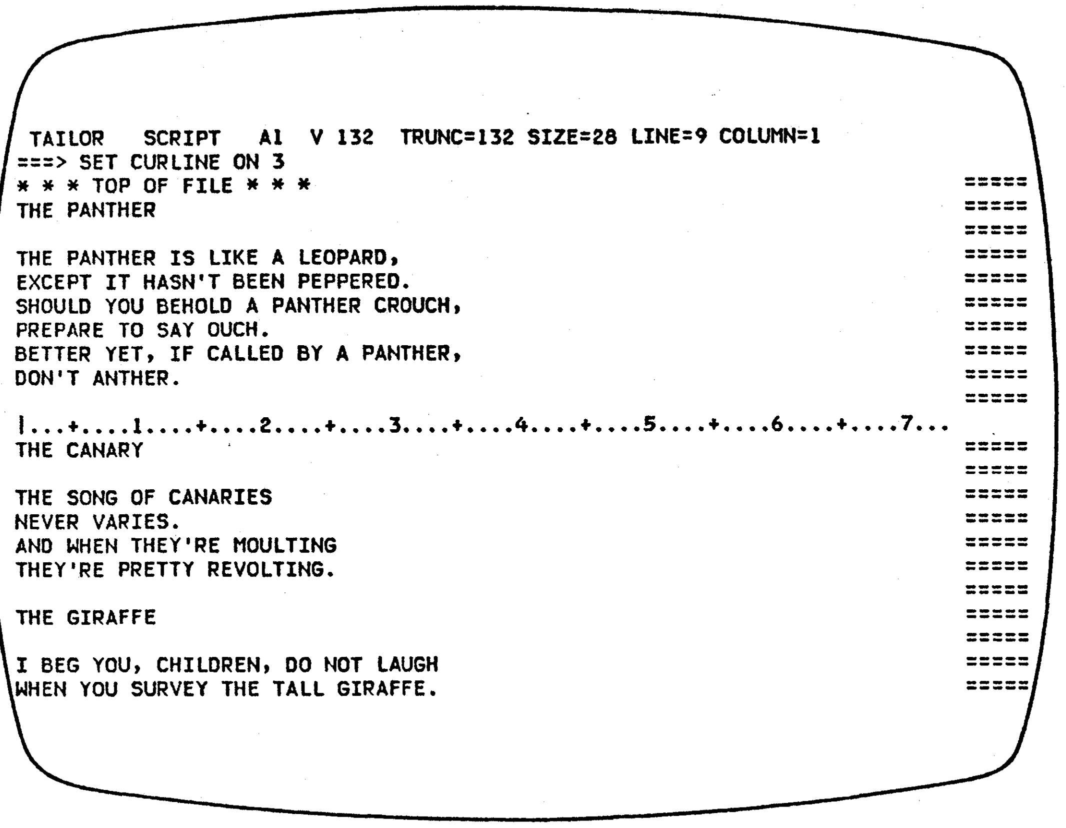


```
 TAILOR   SCRIPT   A1  V 132  TRUNC=132 SIZE=28 LINE=9 COLUMN=1

* * * TOP OF FILE * * *                                                    =====
THE PANTHER                                                                =====
                                                                           =====
THE PANTHER IS LIKE A LEOPARD,                                             =====
EXCEPT IT HASN'T BEEN PEPPERED.                                            =====
SHOULD YOU BEHOLD A PANTHER CROUCH,                                        =====
PREPARE TO SAY OUCH.                                                       =====
BETTER YET, IF CALLED BY A PANTHER,                                        =====
DON'T ANTHER.                                                              =====
                                                                           =====
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
THE CANARY                                                                 =====
                                                                           =====
THE SONG OF CANARIES                                                       =====
NEVER VARIES.                                                              =====
AND WHEN THEY'RE MOULTING                                                  =====
THEY'RE PRETTY REVOLTING.                                                  =====
                                                                           =====
THE GIRAFFE                                                                =====
                                                                           =====
===> SET CMDLINE TOP
                                                              X E D I T  1 FILE
```

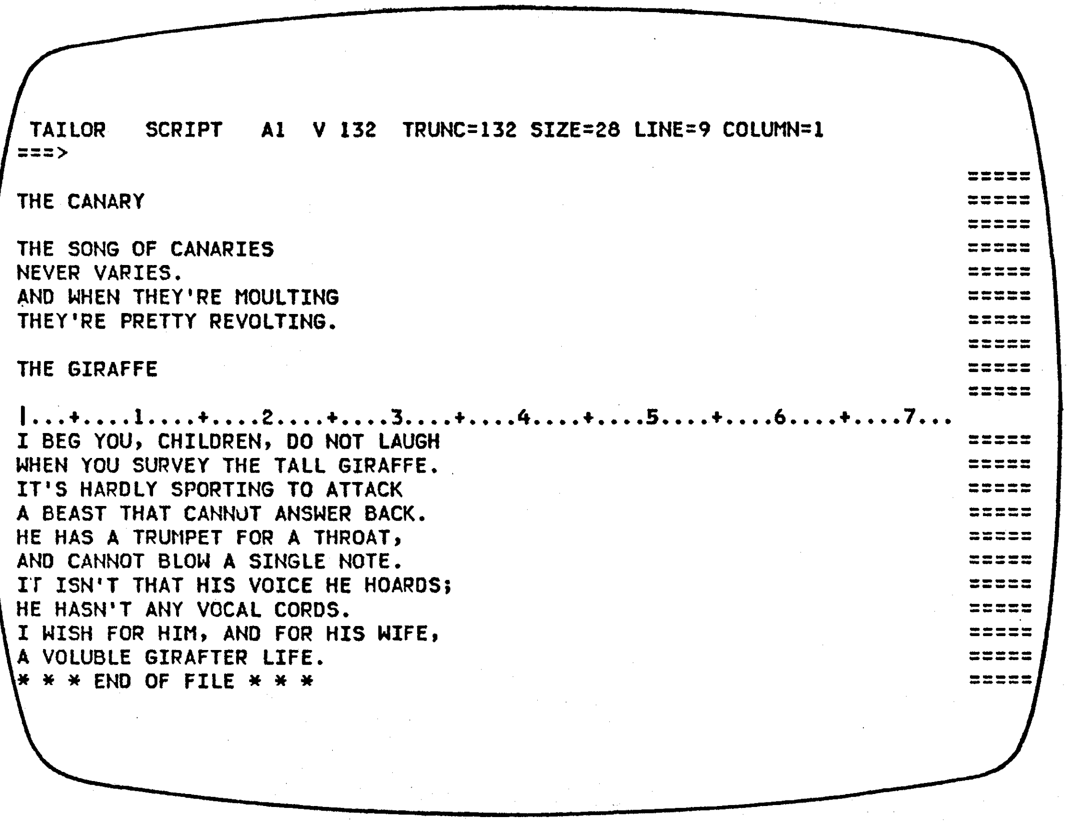


```
 TAILOR   SCRIPT   A1  V 132  TRUNC=132 SIZE=28 LINE=9 COLUMN=1
===>
* * * TOP OF FILE * * *                                                    =====
THE PANTHER                                                                =====
                                                                           =====
THE PANTHER IS LIKE A LEOPARD,                                             =====
EXCEPT IT HASN'T BEEN PEPPERED.                                            =====
SHOULD YOU BEHOLD A PANTHER CROUCH,                                        =====
PREPARE TO SAY OUCH.                                                       =====
BETTER YET, IF CALLED BY A PANTHER,                                        =====
DON'T ANTHER.                                                              =====
                                                                           =====
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
THE CANARY                                                                 =====
                                                                           =====
THE SONG OF CANARIES                                                       =====
NEVER VARIES.                                                              =====
AND WHEN THEY'RE MOULTING                                                  =====
THEY'RE PRETTY REVOLTING.                                                  =====
                                                                           =====
THE GIRAFFE                                                                =====
                                                                           =====
I BEG YOU, CHILDREN, DO NOT LAUGH                                          =====
WHEN YOU SURVEY THE TALL GIRAFFE.                                          =====
```

Figure 6-2. The SET CMDLINE Subcommand - "Before" and "After"

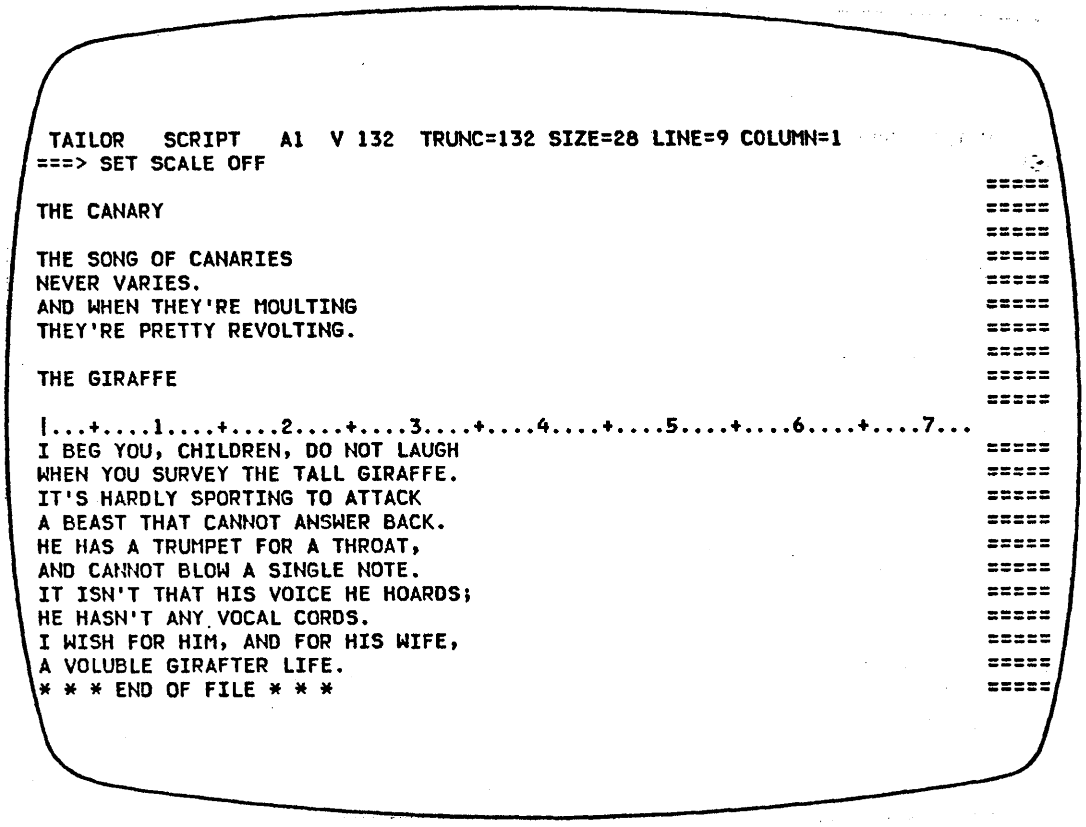


```
 TAILOR   SCRIPT   A1  V 132  TRUNC=132 SIZE=28 LINE=9 COLUMN=1
===> SET CURLINE ON 3
* * * TOP OF FILE * * *                                                  =====
THE PANTHER                                                              =====
                                                                         =====
THE PANTHER IS LIKE A LEOPARD,                                           =====
EXCEPT IT HASN'T BEEN PEPPERED.                                          =====
SHOULD YOU BEHOLD A PANTHER CROUCH,                                      =====
PREPARE TO SAY OUCH.                                                     =====
BETTER YET, IF CALLED BY A PANTHER,                                      =====
DON'T ANTHER.                                                            =====
                                                                         =====
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
THE CANARY                                                               =====
                                                                         =====
THE SONG OF CANARIES                                                     =====
NEVER VARIES.                                                            =====
AND WHEN THEY'RE MOULTING                                                =====
THEY'RE PRETTY REVOLTING.                                                =====
                                                                         =====
THE GIRAFFE                                                              =====
                                                                         =====
I BEG YOU, CHILDREN, DO NOT LAUGH                                        =====
WHEN YOU SURVEY THE TALL GIRAFFE.                                        =====
```

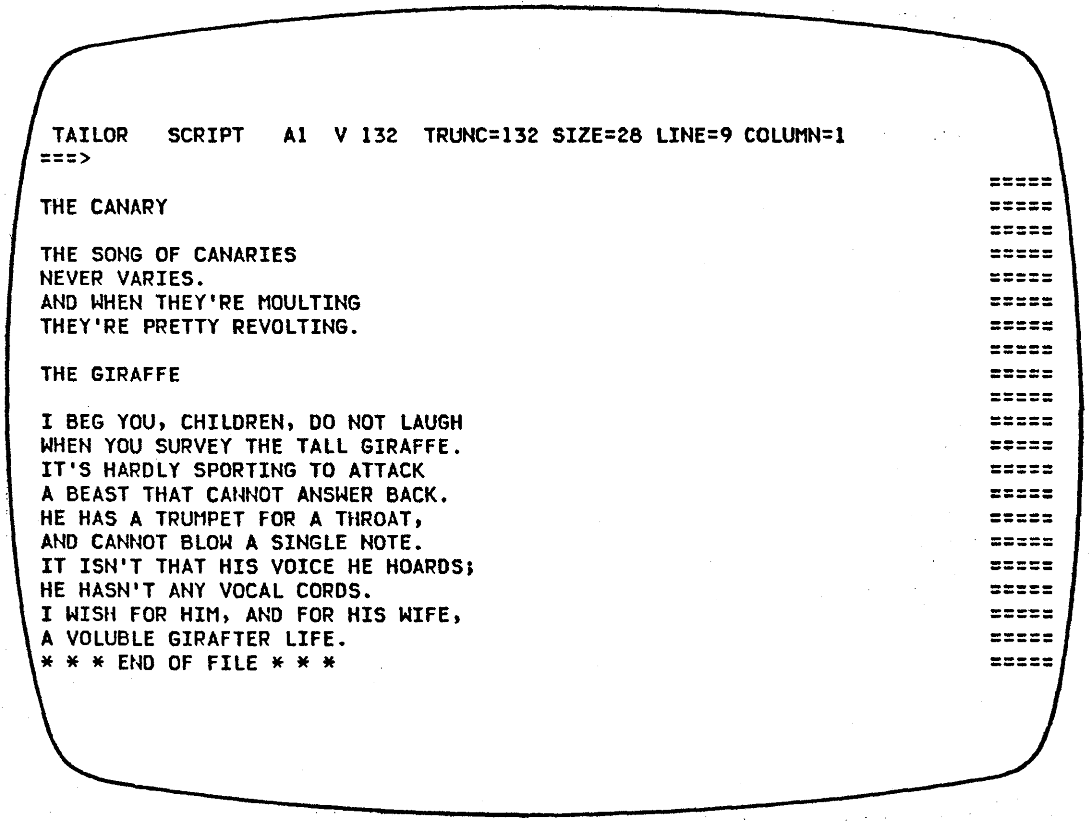


```
 TAILOR   SCRIPT   A1  V 132  TRUNC=132 SIZE=28 LINE=9 COLUMN=1
===>
                                                                         =====
THE CANARY                                                               =====
                                                                         =====
THE SONG OF CANARIES                                                     =====
NEVER VARIES.                                                            =====
AND WHEN THEY'RE MOULTING                                                =====
THEY'RE PRETTY REVOLTING.                                                =====
                                                                         =====
THE GIRAFFE                                                              =====
                                                                         =====
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
I BEG YOU, CHILDREN, DO NOT LAUGH                                        =====
WHEN YOU SURVEY THE TALL GIRAFFE.                                        =====
IT'S HARDLY SPORTING TO ATTACK                                           =====
A BEAST THAT CANNOT ANSWER BACK.                                         =====
HE HAS A TRUMPET FOR A THROAT,                                           =====
AND CANNOT BLOW A SINGLE NOTE.                                           =====
IT ISN'T THAT HIS VOICE HE HOARDS;                                       =====
HE HASN'T ANY VOCAL CORDS.                                               =====
I WISH FOR HIM, AND FOR HIS WIFE,                                        =====
A VOLUBLE GIRAFTER LIFE.                                                 =====
* * * END OF FILE * * *                                                  =====
```

Figure 6-3. The SET CURLINE Subcommand - "Before" and "After"

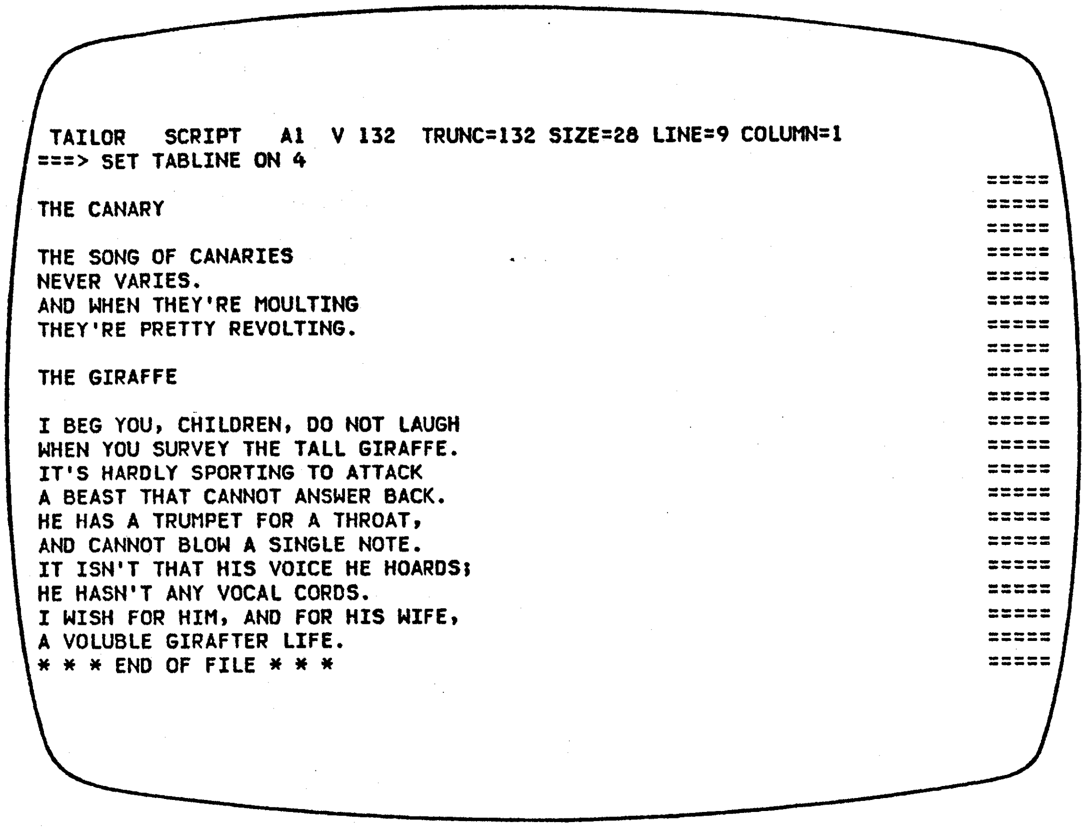


```
 TAILOR   SCRIPT   A1  V 132  TRUNC=132 SIZE=28 LINE=9 COLUMN=1
===> SET SCALE OFF
                                                                      =====
THE CANARY                                                            =====
                                                                      =====
THE SONG OF CANARIES                                                  =====
NEVER VARIES.                                                         =====
AND WHEN THEY'RE MOULTING                                             =====
THEY'RE PRETTY REVOLTING.                                             =====
                                                                      =====
THE GIRAFFE                                                           =====
                                                                      =====
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
I BEG YOU, CHILDREN, DO NOT LAUGH                                     =====
WHEN YOU SURVEY THE TALL GIRAFFE.                                     =====
IT'S HARDLY SPORTING TO ATTACK                                        =====
A BEAST THAT CANNOT ANSWER BACK.                                      =====
HE HAS A TRUMPET FOR A THROAT,                                        =====
AND CANNOT BLOW A SINGLE NOTE.                                        =====
IT ISN'T THAT HIS VOICE HE HOARDS;                                    =====
HE HASN'T ANY VOCAL CORDS.                                            =====
I WISH FOR HIM, AND FOR HIS WIFE,                                     =====
A VOLUBLE GIRAFTER LIFE.                                              =====
* * * END OF FILE * * *                                               =====
```

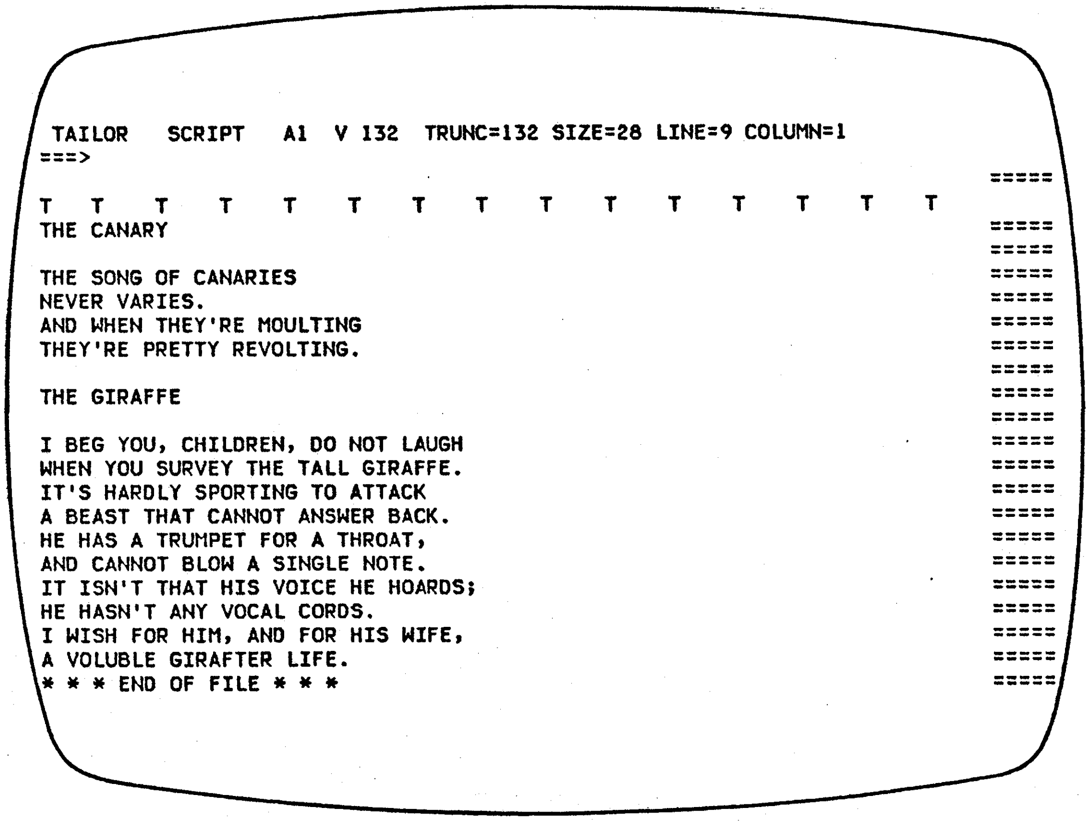


```
 TAILOR   SCRIPT   A1  V 132  TRUNC=132 SIZE=28 LINE=9 COLUMN=1
===>
                                                                      =====
THE CANARY                                                            =====
                                                                      =====
THE SONG OF CANARIES                                                  =====
NEVER VARIES.                                                         =====
AND WHEN THEY'RE MOULTING                                             =====
THEY'RE PRETTY REVOLTING.                                             =====
                                                                      =====
THE GIRAFFE                                                           =====
                                                                      =====
I BEG YOU, CHILDREN, DO NOT LAUGH                                     =====
WHEN YOU SURVEY THE TALL GIRAFFE.                                     =====
IT'S HARDLY SPORTING TO ATTACK                                        =====
A BEAST THAT CANNOT ANSWER BACK.                                      =====
HE HAS A TRUMPET FOR A THROAT,                                        =====
AND CANNOT BLOW A SINGLE NOTE.                                        =====
IT ISN'T THAT HIS VOICE HE HOARDS;                                    =====
HE HASN'T ANY VOCAL CORDS.                                            =====
I WISH FOR HIM, AND FOR HIS WIFE,                                     =====
A VOLUBLE GIRAFTER LIFE.                                              =====
* * * END OF FILE * * *                                               =====
```

Figure 6-4. The SET SCALE Subcommand - "Before" and "After"



```
 TAILOR   SCRIPT   A1  V 132  TRUNC=132 SIZE=28 LINE=9 COLUMN=1
===> SET TABLINE ON 4
                                                                  =====
THE CANARY                                                        =====
                                                                  =====
THE SONG OF CANARIES                                              =====
NEVER VARIES.                                                     =====
AND WHEN THEY'RE MOULTING                                         =====
THEY'RE PRETTY REVOLTING.                                         =====
                                                                  =====
THE GIRAFFE                                                       =====
                                                                  =====
I BEG YOU, CHILDREN, DO NOT LAUGH                                 =====
WHEN YOU SURVEY THE TALL GIRAFFE.                                 =====
IT'S HARDLY SPORTING TO ATTACK                                    =====
A BEAST THAT CANNOT ANSWER BACK.                                  =====
HE HAS A TRUMPET FOR A THROAT,                                    =====
AND CANNOT BLOW A SINGLE NOTE.                                    =====
IT ISN'T THAT HIS VOICE HE HOARDS;                                =====
HE HASN'T ANY VOCAL CORDS.                                        =====
I WISH FOR HIM, AND FOR HIS WIFE,                                 =====
A VOLUBLE GIRAFTER LIFE.                                          =====
* * * END OF FILE * * *                                           =====
```



```
 TAILOR   SCRIPT   A1  V 132  TRUNC=132 SIZE=28 LINE=9 COLUMN=1
===>
                                                                  =====
T  T   T   T   T   T   T   T   T   T   T   T   T   T   T
THE CANARY                                                        =====
                                                                  =====
THE SONG OF CANARIES                                              =====
NEVER VARIES.                                                     =====
AND WHEN THEY'RE MOULTING                                         =====
THEY'RE PRETTY REVOLTING.                                         =====
                                                                  =====
THE GIRAFFE                                                       =====
                                                                  =====
I BEG YOU, CHILDREN, DO NOT LAUGH                                 =====
WHEN YOU SURVEY THE TALL GIRAFFE.                                 =====
IT'S HARDLY SPORTING TO ATTACK                                    =====
A BEAST THAT CANNOT ANSWER BACK.                                  =====
HE HAS A TRUMPET FOR A THROAT,                                    =====
AND CANNOT BLOW A SINGLE NOTE.                                    =====
IT ISN'T THAT HIS VOICE HE HOARDS;                                =====
HE HASN'T ANY VOCAL CORDS.                                        =====
I WISH FOR HIM, AND FOR HIS WIFE,                                 =====
A VOLUBLE GIRAFTER LIFE.                                          =====
* * * END OF FILE * * *                                           =====
```

Figure 6-5. The SET TABLINE Subcommand - "Before" and "After"

Chapter 7: The Macro Language

The macro language is one of the most powerful facilities that the editor provides. By writing macros, you can:

- expand the basic subcommand language
- tailor the language to your own application
- eliminate repetitive tasks.

This chapter explains how to write an XEDIT macro, discusses those XEDIT subcommands designed for use in macros, describes an XEDIT macro written for a text processing application, and explains a profile macro. You should be familiar with the EXEC 2 interpreter, which is described in the publication *VM/SP: EXEC 2 Reference,* before you read this section.

What is an XEDIT Macro?

An XEDIT macro is an EXEC 2 file that is invoked from the XEDIT environment.

You execute a macro the same way you execute XEDIT subcommands: type the macro name on the command line and press the ENTER key. A macro may be executed by entering only its name, or its execution may also depend on arguments you enter when the macro is invoked.

A macro file can contain:

- XEDIT subcommands
- EXEC 2 control statements
- CMS and CP commands

Creating a Macro File

Because an XEDIT macro is a normal CMS file, it may be created in any of the ways that CMS provides for file creation. It can even be created dynamically, by using the XEDIT multiple file editing capability (see "Chapter 5: Editing Multiple Files"). As soon as a FILE subcommand is executed for the macro file, the macro can be used.

Like any CMS file, a macro file is identified by filename, filetype, and filemode. The file identifier for a macro file must follow certain rules:

- The filename is a string of one to eight alphameric characters. This name is used to invoke the macro. For example, if the filename is SEND, entering "SEND" during an editing session causes the macro to be executed.
- The filetype must be XEDIT.
- The filemode can specify any of your accessed disks, for example, A1.

Using XEDIT Subcommands in a Macro

A macro can contain any XEDIT subcommand. However, some subcommands perform functions that are meaningful only in the context of a macro, for example, one that passes information to the EXEC 2 interpreter.

The following list summarizes these subcommands; they are then discussed according to function. For detailed information on these subcommands, refer to the publication *VM/SP: System Product Editor Command and Macro Reference.*

CMS	PRESERVE
CMSG	READ
CP	RESTORE
CURSOR	SET MSGMODE
EMSG	SET RESERVED
MACRO	STACK
MSG	TRANSFER

Communicating Between the Editor and EXEC 2

The following subcommands are discussed in this section:

READ
STACK
TRANSFER

These subcommands all place information *in the console stack.* Once something is in the console stack, it cannot be used by the macro until it has been taken *out of the console stack.* The EXEC 2 &READ statement takes information out of the console stack and assigns it to EXEC 2 variables, which can then be examined by the macro.

The main difference between READ, STACK, and TRANSFER is that they get the data to be placed in the console stack from different places:

READ
: puts data that you enter on the *command line* in the console stack.

STACK
: puts the contents of the *current line of the file* being edited in the console stack.

TRANSFER
: puts one or more of the *editor's variables* in the console stack, that is, it gets data from the editor's work areas.

Data placed in the console stack is then read by an EXEC 2 &READ statement.

The following sections provide examples of using READ, STACK, and TRANSFER.

READ Subcommand

When a READ subcommand is issued from a macro, the editor displays "MACRO-READ" in the status area of your screen and waits for you to enter data on the command line. (Your file image remains on the screen.) After you type the data on the command line and press the ENTER key, the data is placed in the console stack. A subsequent EXEC 2 &READ statement assigns the data to an EXEC 2 variable, and the macro continues executing. This sequence is illustrated in Figure 7-1.

Normally, a macro displays a message requesting that you enter data on the command line before it issues the READ.

For example:

```
MSG ENTER FILE ID
```

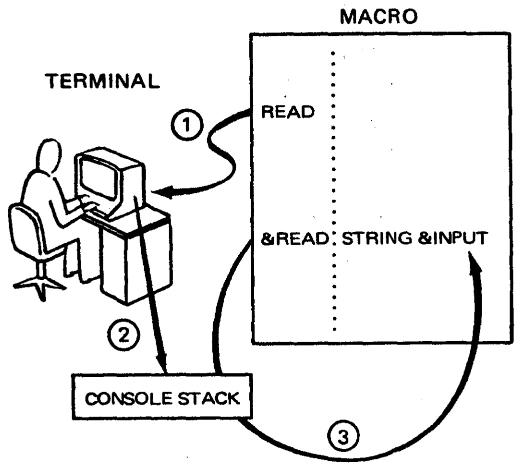
("ENTER FILE ID" is displayed in the message line.)

```
READ
```

(User enters MYFILE SCRIPT A in the command line and READ puts it in the console stack.)

```
&READ ARGS
```

(Takes the fileid out of the stack and assigns MYFILE, SCRIPT, and A to &1, &2, and &3, respectively.)



1 READ is issued. Editor waits for you to enter data on command line.

2 Data you enter is put in the console stack.

3 &READ takes it out of the console stack and assigns it to &INPUT.

Figure 7-1. READ and the Console Stack

The READ subcommand can also be used to place more than one line in the console stack; this is described in the READ subcommand description in the publication *VM/SP: System Product Editor Command and Macro Reference.*

STACK Subcommand

The STACK subcommand places the current line of the file from which the macro was invoked in the console stack. An EXEC 2 &READ statement takes it out of the console stack and assigns it to an EXEC 2 variable. This sequence is illustrated in Figure 7-2.

The STACK subcommand can also be used to put more than one line in the console stack; this is described in the STACK subcommand description in the publication *VM/SP: System Product Editor Command and Macro Reference.*

TRANSFER Subcommand

The current setting of all editing options, that is, those options that are defined by the SET subcommand, are available to the macro through the use of the TRANSFER subcommand. In addition, the values of other editing variables that are not explicitly "set" are available.

The TRANSFER subcommand places one or more of the editor's variables in the console stack. An EXEC 2 &READ statement assigns them to EXEC 2 variables. This is illustrated in Figure 7-3.

(For a complete list of the variables that can be "transferred", refer to the TRANSFER subcommand description in the publication *VM/SP: System Product Editor Command and Macro Reference.*)

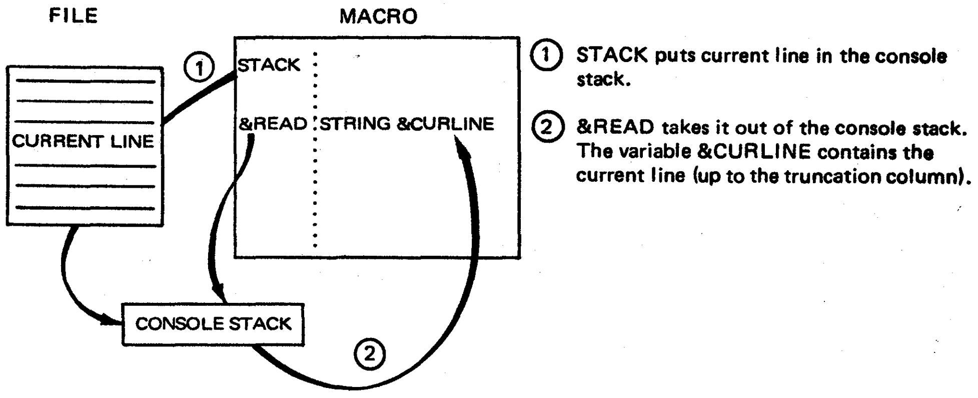

FILE
MACRO
CURRENT LINE
STACK
&READ STRING &CURLINE
CONSOLE STACK
① STACK puts current line in the console stack.
② &READ takes it out of the console stack. The variable &CURLINE contains the current line (up to the truncation column).

Figure 7-2. STACK and the Console Stack

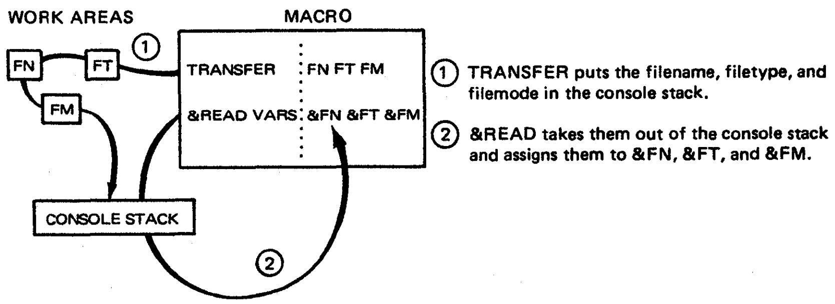

WORK AREAS
MACRO
FN
FT
FM
TRANSFER FN FT FM
&READ VARS &FN &FT &FM
CONSOLE STACK
① TRANSFER puts the filename, filetype, and filemode in the console stack.
② &READ takes them out of the console stack and assigns them to &FN, &FT, and &FM.

Figure 7-3. TRANSFER and the Console Stack

Displaying Data on the Editor's Screen

The following subcommands are discussed in this section:

MSG
EMSG
CMSG
SET MSGMODE
SET RESERVED
CURSOR

MSG, EMSG, and CMSG Subcommands

A macro can communicate with the user by displaying messages in the message line of the screen. Messages are used for various reasons, for example, requesting the user to enter data, telling a user that an error has occurred during processing, and so forth.

The following two subcommands display a message in the message line of the screen:

`MSG`

Displays a message in the message line.

`EMSG`

Displays a message in the message line and sounds the alarm.

For example:

```
MSG ENTER FILE NAME
```

Displays "ENTER FILE NAME" in the message line.

```
EMSG  MISSING OPERANDS
```

Displays "MISSING OPERANDS" in the message line and sounds the alarm.

The following subcommand displays a message in the command line of the screen:

```
CMSG
```

When issued from a macro, the CMSG subcommand can be used to re-display input that the user has entered incorrectly, so that it can be corrected and re-entered.

Note: EXEC 2 also provides a control statement, &TYPE, that displays one line of data at the terminal. However, the &TYPE statement causes the screen to be cleared before the data is displayed. The XEDIT subcommands MSG and EMSG keep the file image on the screen and display the data in the message line. Therefore, you should use them instead of &TYPE in a macro.

SET MSGMODE Subcommand

The SET MSGMODE subcommand is used to control whether or not messages are displayed:

```
SET MSGMODE ON
```

All messages are displayed.

```
SET MSGMODE OFF
```

No messages are displayed.

By turning the message mode on and off during a macro, you can select when you want messages to be displayed.

SET RESERVED Subcommand

When issued from a macro, the SET RESERVED subcommand reserves a specified line on the screen for use by the macro, thereby preventing the editor from using that line. The line can be used for displaying blank or specified information, which can optionally be highlighted.

For example, the following subcommand:

```
SET RESERVED 10 HIGH YOU CAN'T USE THIS LINE.
```

displays, on the tenth line of the screen, "YOU CAN'T USE THIS LINE". The line is highlighted.

CURSOR Subcommand

The CURSOR subcommand can be used to move the cursor to a specified position on the screen. For example, the editor has a macro called SCHANGE, which looks for a string and moves the cursor under the string if it is found.

Saving and Restoring Editing Variables

The PRESERVE subcommand is used to save the settings of various editing variables until a subsequent RESTORE subcommand is issued. For a complete list of the variables affected, refer to the PRESERVE subcommand description in the publication *VM/SP: System Product Editor Command and Macro Reference.*

Issuing CMS and CP Commands

As you have seen, an XEDIT macro can contain XEDIT subcommands, EXEC 2 control statements, and CMS and CP commands. CMS and CP commands can be issued as operands of the XEDIT subcommands CMS and CP, respectively.

For example:

```
CMS ERASE FILEA SCRIPT
```

(CMS and CP commands can also be issued by using the EXEC 2 control statement, &COMMAND.)

Avoiding Name Conflicts

Use the MACRO subcommand to cause the editor to execute a specified macro without first checking to see if a subcommand of the same name or a synonym exists.

When a subcommand has a number as its operand, a blank is not required between the subcommand name and the operand. For example, both "NEXT8" and "N8" are interpreted by the editor as being the subcommand "NEXT 8". Therefore, if a macro name were also "N8", the macro would not be executed; the subcommand "NEXT 8" would be executed instead. To execute the macro, you could enter the following:

```
MACRO N8
```

The macro whose name is "N8" would then be executed.

The SET MACRO subcommand can be used to control the order in which the editor searches for subcommands and macros:

SET MACRO ON tells the editor to look for macros before it looks for subcommands; SET MACRO OFF reverses the order.

Walking Through an XEDIT Macro

The following XEDIT macro is an example of the type of macro you might write to make life a little easier. The application is typical of a text processing file arrangement, where many SCRIPT files are imbedded in a master file, via the SCRIPT control word ".im".

The problem with this type of setup is that if you have to make a global change throughout all the files, you have to edit each file, make the change, and then file each file.

When issued from the master file, this macro edits each file, performs a global change, and files it.

The macro is invoked by entering the macro name, GLOBCHG; the arguments passed to the macro are the old data and the new data, enclosed in delimiters:

```
GLOBCHG /string1/string2/
```

For example, if a file called MASTER SCRIPT contains:

```
  .im FILE1
  .im FILE2
     •
     •
     •
  .im FILE100
```

and the following commands are issued:

```
XEDIT MASTER SCRIPT
GLOBCHG/WAR AND PEACE/SENSE AND NONSENSE/
```

"WAR AND PEACE" is changed to "SENSE AND NONSENSE" each time it occurs in every file. (In this macro, no attempt is made to execute the change on files that may be imbedded at the next level.)

The GLOBCHG macro can also be used to delete data throughout the files, by changing a string to a null string. For example:

```
GLOBCHG /bad data//
```

The following is a listing of the macro, whose fileid is GLOBCHG XEDIT A1. After the listing, each line in the macro is explained. For more information on the EXEC 2 statements used in the macro, see the publication *VM/SP: EXEC 2 Reference.*

```
00001 &IF &N = 0 &GOTO -MISSINGOPERANDS
00002 &OPERAND = &ARGSTRING
00003 PRESERVE
00004 SET MSGMODE OFF
00005 TOP
00006 FIND .im
00007 &IF &RC  ¬= 0 &GOTO -NOIMBED
00008 -LOOP
00009 STACK 1
00010 &READ ARGS
00011 &COMMAND STATE &2 SCRIPT *
00012 &IF &RC = 0 &SKIP 4
00013   SET MSGMODE ON
00014   EMSG IMBEDDED FILE ' &2 SCRIPT ' DOES NOT EXIST; BYPASSED.
00015   SET MSGMODE OFF
00016   &GOTO -ENDLOOP
00017 XEDIT &2 SCRIPT (NOPROFILE
00018 TRANSFER FNAME FTYPE FMODE
00019 &READ STRING &FILEID
00020 MSG PROCESSING FILE ' &FILEID '
00021 CHANGE &OPERAND * *
00022 FILE
00023 -ENDLOOP FIND .im
00024 &IF &RC = 0 &GOTO -LOOP
00025 RESTORE
00026 MSG GLOBAL CHANGE COMPLETED.
00027 &EXIT
00028 -NOIMBED
00029 RESTORE
00030 EMSG NO IMBED FOUND.
00031 &EXIT
00032 -MISSINGOPERANDS   EMSG EXE545E MISSING OPERAND(S)
00033 CMSG &0
00034 &EXIT
```

Figure 7-4. A Sample Macro

Now, let's walk through the macro, a line at a time.

```
00001 &IF &N = 0 &GOTO -MISSINGOPERANDS
```

If the number of arguments you passed to the macro (&N) is zero, go to the statement labelled "-MISSINGOPERANDS", where an error message is issued. Obviously, this macro cannot work unless you tell it what to change.

```
00002 &OPERAND = &ARGSTRING
```

The user-defined variable (&OPERAND) is assigned the value of the arguments passed to the macro. (The arguments are the old and new strings of data to be changed.)

The next four statements in the macro are XEDIT subcommands:

```
00003 PRESERVE
```

This subcommand saves the editor settings until a subsequent RESTORE subcommand is issued (statement 29).

```
00004 SET MSGMODE OFF
```

No messages will be displayed. By turning the message mode on and off, you can select which messages you want displayed. Message mode is set OFF here to prevent messages from the FIND subcommand (statement 6) from being displayed, because the macro issues its own message (statement 30) if no imbedded files are found.

```
00005 TOP
```

Move the line pointer to the top of the master file, which is the file from which the macro was invoked.

```
00006 FIND .im
```

Search forward in the master file for the first line that contains ".im" in column 1, that is, locate the first line that imbeds a file.

```
00007 &IF &RC  ¬= 0 &GOTO -NOIMBED
```

If there is a non-zero return code from the FIND subcommand (statement 6), go to the statement labelled "-NOIMBED". This situation occurs if no ".im" statements are found in the master file.

Statements 8 through 23 are the major loop in the macro, in which the global change is made on each imbedded file:

```
00008 -LOOP
```

This is the statement label that begins the loop.

```
00009 STACK 1
```

When the FIND subcommand (statement 6) locates a ".im filename" statement in the master file, it makes that line the current line. This STACK subcommand places the current line in the console stack, so that its contents can be read by the following statement.

```
00010 &READ ARGS
```

This statement reads a line from the console stack.

```
00011 &COMMAND STATE &2 SCRIPT *
```

The STATE command is a CMS command that verifies the existence of a file. This statement checks to see if the file named in the ".im filename" statement exists. (EXEC 2 transmits the STATE command directly to CMS.)

```
00012 &IF &RC = 0 &SKIP 4
```

If the return code from the STATE command is zero, the file exists, so skip down to statement 17. If it is not zero, execute the next four statements (13-16), which comprise "file not found" processing:

```
00013   SET MSGMODE ON
```

Since a message will be issued by the next statement, turn message mode back on so that it will be displayed.

Statements 14, 15, and 16, issue the message, turn message mode off, and branch to the statement label which begins the "FIND" loop again.

```
00014   EMSG IMBEDDED FILE ' &2 SCRIPT ' DOES NOT EXIST; BYPASSED.
00015   SET MSGMODE OFF
00016   &GOTO -ENDLOOP
```

```
00017 XEDIT &2 SCRIPT (NOPROFILE
```

The XEDIT subcommand brings the imbedded file into virtual storage.

```
00018 TRANSFER FNAME FTYPE FMODE
```

Place the fileid of the imbedded file in the console stack, so that it can be accessed by the following EXEC 2 statement.

```
00019 &READ STRING &FILEID
```

Read the line from the console stack, and assign its contents to the variable "&FILEID".

```
00020 MSG PROCESSING FILE ' &FILEID '
```

Display a message in the message line of the terminal (without sounding the alarm). Even though SET MSGMODE OFF was executed (statement 4), this message will be displayed, because the subcommand XEDIT was subsequently issued (statement 17); this causes the editor's initial setting (MSGMODE ON) to be in effect.

```
00021 CHANGE &OPERAND * *
```

The global change is executed. (The arguments you entered when the macro was invoked were assigned to the "&OPERAND" variable in statement 2.)

Messages issued by the CHANGE subcommand are displayed, for example, "nn OCCURRENCES CHANGED ON nn LINES".

```
00022 FILE
```

The changed file is written to disk.

```
00023 -ENDLOOP FIND .im
```

Then, the editor resumes editing the master file, searching for the next ".im filename" statement.

```
00024 &IF &RC = 0 &GOTO -LOOP
```

If the "FIND" (statement 23) is successful, go through the loop again.

```
00025 RESTORE
```

If the "FIND" (statement 23) is not successful, restore the settings of XEDIT variables to the values they had when the PRESERVE subcommand was issued (statement 3).

```
00026 MSG GLOBAL CHANGE COMPLETED.
```

Display the message.

```
00027 &EXIT
```

Return control to the editor; you can then issue a QUIT subcommand for the master file.

Statements 28 through 31 are executed if no ".im" statements were found in the master file:

```
00028 -NOIMBED
00029 RESTORE
00030 EMSG NO IMBED FOUND.
00031 &EXIT
```

```
00032 -MISSINGOPERANDS   EMSG EXE545E MISSING OPERAND(S)
```

This message is displayed in the message line if no arguments, which are required, were entered when the macro was invoked.

```
00033 CMSG &0
```

In addition, the macro name (GLOBCHG) is displayed in the command line, so that you can type the arguments (/string1/string2/) and press the ENTER key to invoke the macro again.

```
00034 &EXIT
```

The end.

A Profile Macro for Editing

As a CMS user, you are familiar with a PROFILE EXEC macro, which contains the CMS and CP commands you normally issue at the start of a terminal session and is executed automatically after you issue the IPL CMS command.

The editor offers a similar profile capability with a PROFILE XEDIT macro, which contains XEDIT subcommands that tailor each editing session to suit your needs and is executed automatically after you issue an XEDIT command (or subcommand).

Executing a Profile Macro

The filetype of a profile macro must be "XEDIT". If the fileid is PROFILE XEDIT, the macro is executed automatically when an XEDIT command (or subcommand) is issued. You can write a PROFILE XEDIT macro, file it, and forget about it. It will be executed before each file is brought into storage.

If you do *not* want a PROFILE XEDIT macro to be executed for a particular editing session, you can issue the following XEDIT command:

```
XEDIT fn ft (NOPROFILE
```

The PROFILE XEDIT macro is bypassed, and the file is brought into storage.

Although the filetype of a profile macro must be "XEDIT", the filename does not have to be "PROFILE". If your profile macro has a name other than "PROFILE", you must indicate its filename in the PROFILE option of the XEDIT command.

For example, if the fileid is MYPROF XEDIT, you must issue the following XEDIT command:

```
XEDIT fn ft (PROFILE MYPROF
```

The macro labelled MYPROF XEDIT is executed, even if a macro labelled PROFILE XEDIT exists.

Writing a Profile Macro

A profile macro can be as simple or complex as you wish. Like any macro, it can contain EXEC 2 statements, CMS and CP commands, and any XEDIT subcommands or macros. It usually contains one or more SET subcommands that create an editing environment to your liking.

It can also contain a LOAD subcommand, which can be issued *only* from a profile macro. When the profile macro begins execution, a copy of the file has not yet been brought into virtual storage. Therefore, a LOAD subcommand, which has the same format and options as the XEDIT command, can be used to supply editing options that are not specified in the XEDIT command itself.

Within the profile macro, the LOAD subcommand must be the first XEDIT subcommand. If it is not, a LOAD subcommand is automatically issued by the editor; its operands are the same as those issued in the XEDIT command. (EXEC 2 statements and CMS commands can be issued before the LOAD.)

The profile macro can be used to prompt the user for XEDIT command options or to assign values to editing variables before issuing the LOAD subcommand. For example, a SCRIPT user might program his profile to use a LOAD subcommand that does defaulting of filetype.

The options specified in the LOAD subcommand have a lower priority than those specified in an XEDIT command. For example, an UPDATE option specified in the LOAD subcommand would be overridden by a NOUPDATE option specified in the XEDIT command.

When the LOAD subcommand is executed, the file is brought into virtual storage.

If the LOAD fails, a non-zero return code is generated. All subsequent subcommands in the profile macro are rejected with a unique "6" return code.

For detailed information on the LOAD subcommand, refer to the publication *VM/SP: System Product Editor Command and Macro Reference.*

An Example of a Profile Macro

An example of a profile macro is shown in Figure 7-5.

```
00001 *
00002 * SET DEFAULT FILETYPE TO SCRIPT
00003 *
00004          &IF &N = 0 &EXIT
00005          &IF &N = 1 &ARGS &1 SCRIPT
00006          &COMMAND STATE &1 &2
00007          &IF &RC = 0 &GOTO -LOAD
00008          & = &LOCATION OF &2 SCRIPT
00009          &IF &  ¬= 1 &GOTO -LOAD
00010          &ARGS &1 SCRIPT
00011 -LOAD    LOAD &1 &2
00012          &IF &RC  ¬= 0 &EXIT &RC
00013 * SET PFKEYS
00014 *
00015          SET PF1 TOP
00016 *
00017 * SET VARIOUS OPTIONS
00018 *
00019          SET SYNONYM / 1 CLOCATE /
00020 *
00021          SET ARBCHAR ON .
00022          SET SPAN ON BLANK 3
00023          SET VARBLANK ON
00024          SET CASE MIXED IGNORE
```

Figure 7-5. A PROFILE XEDIT Macro

The EXEC 2 control statements (4-10) supply a SCRIPT filetype if no filetype is specified in the XEDIT command, or accept any abbreviation of "SCRIPT". Therefore, when the LOAD subcommand (statement 11) is executed, the proper filetype is supplied.

The XEDIT statements assign a subcommand to a PF key (statement 15), define a synonym (statement 19), and set up editing variables (statements 21-24) that the user wants.

Appendix: A Summary of XEDIT Subcommands and Macros

Subcommand	Purpose
Add	Add n lines after current line.
ALter	Change a single character to another (character or hex).
BAckward	Scroll backward n frames.
Bottom	Go to last line of file.
CANCEL	Terminate all files.
CAppend	Add text to end of current line.
CDelete	Delete characters, starting at column pointer.
CFirst	Move column pointer to beginning of line (zone).
Change	Change one string to another.
CInsert	Insert text in the current line.
CLAst	Move the column pointer to the end of the line (zone).
CLocate	Locate a string; move the column pointer and the line pointer.
CMS	Pass a command to CMS, or enter CMS subset mode.
CMSG	Display message in command line of user's screen.
COMMAND	Execute a subcommand without checking for synonym or macro.
COMPress	Prepare line(s) for realignment by replacing blanks with tab characters.
COpy	Copy line(s) at specified location.
COUnt	Display the number of times a string appears.
COVerlay	Replace characters, starting at column pointer.
CP	Pass command to VM/SP control program.
CReplace	Replace characters, starting at the column pointer.
CURsor	Move the cursor to specified position on the screen.
DELete	Delete line(s).
Down	Move line pointer n lines toward end of file (same as NEXT).
DUPlicat	Duplicate line(s).
EMSG	Display a message and sound the alarm.
EXPand	Reposition data according to new tab settings.

Subcommand	Purpose
FILE	Write file on disk.
Find	Search for line that starts with specified text.
FINDUp	Search for a line that starts with specified text; searches in a backward direction.
FOrward	Scroll forward n frames.
GET	Insert lines from another file.
Help	Request on-line display of XEDIT subcommands and macros; invoke the CMS HELP facility.
HEXType	Display line(s) in hexadecimal and EBCDIC.
Input	Insert a single line, or enter input mode.
Join	Join lines.
LEft	View data to the left of column one.
LOAD	Read file into storage; use in profile macro only.
Locate	Move line pointer to specified target.
LOWercas	Change uppercase letters to lowercase.
MACRO	Execute macro without checking for subcommand or synonym.
MODify	Display a SET subcommand current values in the command line, so it can be overtyped and reentered.
MOve	Move line(s) to another place in the file.
MSG	Display message in message line.
Next	Move line pointer n lines toward end of file (same as DOWN).
NFind	Search for first line that does not match specified text.
NFINDUp	Search backward for first line that does not match specified text.
Overlay	Replace characters in current line.
PARSE	Scan a line of a macro to check the format of its operands.
POWerinp	Enter an input mode for continuous typing.
PREServe	Save settings of variables until RESTORE.

Subcommand	Purpose
PURge	Remove macro from virtual storage.
PUT	Insert lines into another file (new or existing), or into a buffer (to be retrieved by GET from another file).
PUTD	Same as PUT, but delete original lines.
Query	Display the current value of editing options.
QUIT	End an editing session without saving changes.
READ	Place information from the terminal in the console stack.
RECover	Replace deleted lines.
RENum	Renumber VSBASIC or FREEFORT file.
REPEat	Advance line pointer and re-execute last subcommand.
Replace	Replace current line, or delete current line and enter input mode.
RESet	Remove prefix subcommands when screen is in "pending" or "incomplete" status.
RESTore	Restore settings of XEDIT variables to values they had when PRESERVE was issued.
RIght	View data to the right of the last (right-most) column.
SAVE	Write file on disk and remain in edit mode.
SCHANGE	Make a selective change, using PF keys.
SET APL	Inform the editor if APL keys are used.
SET ARBchar	Define an arbitrary character, which allows you to specify only the beginning and the end of a character string that is the object of a target search.
SET AUtosave	Automatically issue a SAVE subcommand at specified intervals.
SET CASE	Upper or lower case control; specify if case is significant in target searches.
SET CMDline	Move the position of the command line.
SET COLPtr	Specify if column pointer is displayed (typewriter terminals only).

Subcommand	Purpose
SET CURLine	Define the position of the current line on the screen.
SET ESCape	Define a character that allows you to enter a subcommand while in input mode (typewriter terminals only).
SET FILler	Define a character that is used when a line is expanded.
SET FMode	Change the filemode of the current file.
SET FName	Change the filename of the current file.
SET FType	Change the filetype of the current file.
SET HEX	Allows string targets to be specified in hexadecimal.
SET IMage	Control how tabs and backspaces are handled.
SET IMPcmscp	Control whether subcommands not recognized by the editor are transmitted to CMS and CP.
SET LINENd	Define a line end character.
SET LRecl	Define a new logical record length.
SET MACRO	Control the order in which the editor searches for subcommands and macros.
SET MASK	Define a new mask, which is the contents of added lines and the input zone.
SET MSGMode	Control the message display.
SET NONDisp	Define a character that is used in place of non-displayable characters.
SET NULls	Specify whether trailing blanks are replaced with nulls to allow character insertion.
SET NUMber	Specify whether file line numbers are displayed in the prefix area.
SET PACK	Specify if the file is to be written to disk in packed format.
SET PFn	Define a meaning for a PF key.
SET Point	Define a symbolic name for the current line.
SET PREfix	Control the display of the prefix area; define a synonym for a prefix subcommand.
SET RANge	Define a new "top" and "bottom" for the file.

Subcommand	Purpose
SET RECFm	Define the record format.
SET RESERved	Reserve a line, which cannot be used by the editor.
SET SCALe	Control the display of the scale line.
SET SCReen	Divide the screen into logical screens, for multiple views of the same or of different files.
SET SERial	Control file serialization.
SET SPAN	Allows a string target to span a number of lines.
SET STAY	Specify whether the line pointer moves when a string is not located.
SET STReam	Specify whether the editor searches only the current line or the whole file for a column-target.
SET SYNonym	Specify whether the editor looks for synonyms; assign a synonym.
SET TABLine	Control the display of the tab line.
SET TABS	Define the logical tab stops.
SET TERMinal	Specify whether a terminal is used in line mode or full screen mode.
SET TEXT	Inform the editor if TEXT keys are used.
SET TOFEOF	Control the display of TOF/EOF lines.
SET TRunc	Define the truncation column.
SET VARblank	Specify whether the number of blanks between two words is significant in a target search.
SET Verify	Control whether lines changed by subcommands are displayed; define the columns displayed and whether displayed in EBCDIC or hexadecimal or both.
SET WRap	Control whether the editor wraps around the file if EOF is reached during a search.
SET Zone	Define new limits within each line for target searches.
SET =	Insert string into the equal buffer.
SHift	Move data right or left (data loss possible).

Subcommand	Purpose
SORT	Sort all or part of a file, in ascending or descending order.
SOS	Specify functions for screen operation simulation.
SPlit	Split a line into two or more lines.
STAck	Place line(s) from the file into the console stack.
STATus	Display SET subcommand current settings; create a macro that contains these settings.
TOP	Move line pointer to null TOP OF FILE line.
TRAnsfer	Place editing variable(s) in the console stack, for use by a macro.
Type	Display lines.
Up	Move line pointer toward top of file.
UPPercas	Translate all lowercase characters to uppercase.
Xedit	Edit multiple files.
&	Use before a subcommand for repeated execution.
=	Re-execute the last subcommand or macro.
?	Display the last subcommand executed.

Prefix Subcommands	Purpose
A	Add line(s).
C	Copy line(s).
D	Delete line(s).
E	Extend a line.
F	Move or copy following this line.
I	Insert line(s).
M	Move line(s).
P	Move or copy preceding this line.
"	Duplicate line(s).
/	Make this line the current line.
SCALE	Display the scale on this line.
TABL	Display the tab line on this line.
.xxxx	Assign symbolic name to this line.

Index

SC24-5220-0

IBM VM/SP: System Product Editor User's Guide (File No. S370-39) Printed in U.S.A. SC24-5220-0

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N. Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N. Y., U. S. A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N. Y., U. S. A. 10601

IBM VM/SP: System Product
Editor User's Guide
SC24-5220-0

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

	Yes	*No*
• Does the publication meet your needs?	☐	☐
• Did you find the material:		
Easy to read and understand?	☐	☐
Organized for convenient use?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐

• What is your occupation? ______________________________

• How do you use this publication:

As an introduction to the subject?	☐	As an instructor in class?	☐
For advanced knowledge of the subject?	☐	As a student in class?	☐
To learn about operating procedures?	☐	As a reference manual?	☐

Your comments:

If you would like a reply, please supply your name and address on the reverse side of this form.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

SC24-5220-0

Reader's Comment Form

Fold and Tape | Please Do Not Staple | Fold and Tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

Fold | Fold

If you would like a reply, *please print:*

Your Name ____________________

Company Name ____________________ *Department* ____________

Street Address ____________________

City ____________________

State ____________________ *Zip Code* ____________

IBM Branch Office serving you ____________________

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N. Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N. Y., U. S. A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N. Y., U. S. A. 10601

Cut or Fold Along Line

IBM VM/SP: System Product Editor User's Guide (File No. S370-39) Printed in U.S.A. SC24-5220-0

IBM VM/SP: System Product
Editor User's Guide
SC24-5220-0

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

	Yes	*No*
• Does the publication meet your needs?	☐	☐
• Did you find the material:		
Easy to read and understand?	☐	☐
Organized for convenient use?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐

• What is your occupation? ____________________

• How do you use this publication:

As an introduction to the subject?	☐	As an instructor in class?	☐
For advanced knowledge of the subject?	☐	As a student in class?	☐
To learn about operating procedures?	☐	As a reference manual?	☐

Your comments:

If you would like a reply, please supply your name and address on the reverse side of this form.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

SC24-5220-0

Reader's Comment Form

Fold and Tape | Please Do Not Staple | Fold and Tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

Fold | Fold

If you would like a reply, *please print:*

Your Name ______________________

Company Name ______________ *Department* ________

Street Address ______________________

City ______________________

State ______________ *Zip Code* ________

IBM Branch Office serving you ______________________

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N. Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N. Y., U. S. A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N. Y., U. S. A. 10601

Cut or Fold Along Line

IBM VM/SP: System Product Editor User's Guide (File No. S370-39) Printed in U.S.A. SC24-5220-0