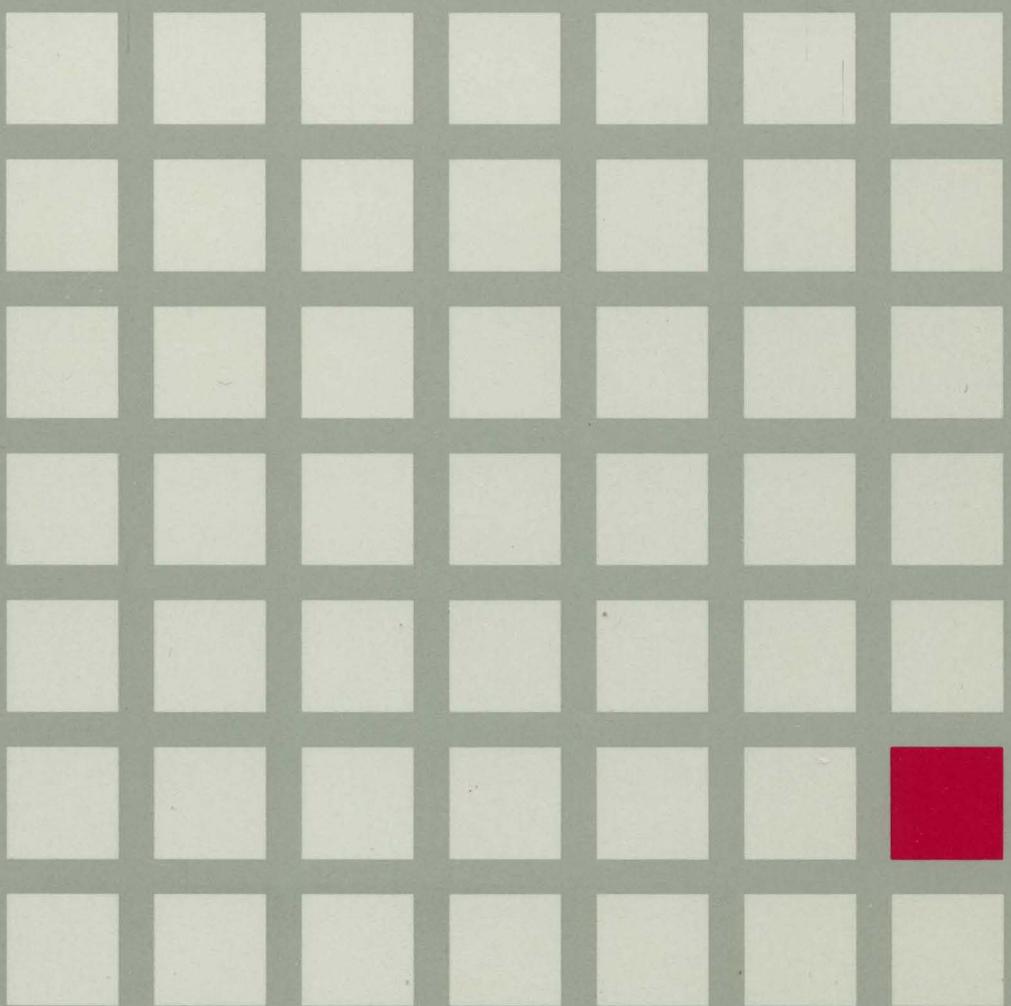
IBM

Virtual Machine/System Product

SC24-5220-04

System Product Editor User's Guide

Release 6



Virtual Machine/System Product

SC24-5220-04

System Product Editor User's Guide

Release 6

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.

Fifth Edition (July 1988)

This edition, SC24-5220-04, is a revision of SC24-5220-03, and applies to Release 6 Virtual Machine/System Product (VM/SP), program number 5664-167, until otherwise indicated in new editions or Technical Newsletters. Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370, 30xx, 4300, and 9370 Processors Bibliography,* GC20-0001, for the editions that are applicable and current.

Summary of Changes

For a detailed list of changes, see page 151.

Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

Ordering Publications

Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality. Publications are *not* stocked at the address given below.

A form for reader's comments is supplied at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Information Development, Dept. G60, P.O. Box 6, Endicott, NY, U.S.A. 13760. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Preface

This book was primarily written 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 directly enter selected subcommands on a displayed line
- The ability to tailor the full-screen layout
- The ability to divide the screen 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 online full-screen display of any XEDIT subcommand or macro (or any command in the CMS HELP facility) during an editing session.

The System Product Editor can manipulate Double Byte Character Set (DBCS) strings (KANJI, for example). Special considerations for editing files that contain double-byte characters are described in the *VM/SP System Product Editor Command and Macro Reference*.

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 uses a display terminal in full-screen mode. It defines a subset of XEDIT subcommands that perform commonly-used editing functions.
- *Chapter 2: Practice Exercises* gives 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: Using the Editor 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 move the line pointer and define the range 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.

Contents

Chapter 1. An XEDIT Subset: Full-Screen Text Processing

This chapter is primarily written for the person who has limited data processing experience; however, some Virtual Machine/System Product (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 in this chapter 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 later in this book and in the *VM/SP System Product Editor Command and Macro Reference*.

This subset has been selected for text processing on a display terminal. (If you have a typewriter terminal, see Chapter 3.)

Editing a File

Editing involves changing, adding, or deleting data within a CMS file. The editor lets you 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 on page 2, 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 on page 2.[1]

---

[1] If your screen layout differs from Figure 1 on page 2, or some of the commands or PF keys work differently than this guide says they will, you may have a PROFILE XEDIT macro tailoring your editing session. To keep the PROFILE XEDIT macro from executing, add the NOPROFILE option:

XEDIT filename filetype (NOPROFILE

See "A Profile Macro for Editing" on page 130 for more information on the PROFILE XEDIT macro.

If these problems continue while using NOPROFILE, be sure this guide was written for your system's release level of VM/SP. See the front cover of this guide for its release level and use QUERY CMSLEVEL (see "QUERY" in the *VM/SP CMS Command Reference*) to find the release level of your system's VM/SP. If they are different, use the guide that matches the VM/SP release level of your system.

Screen Layout

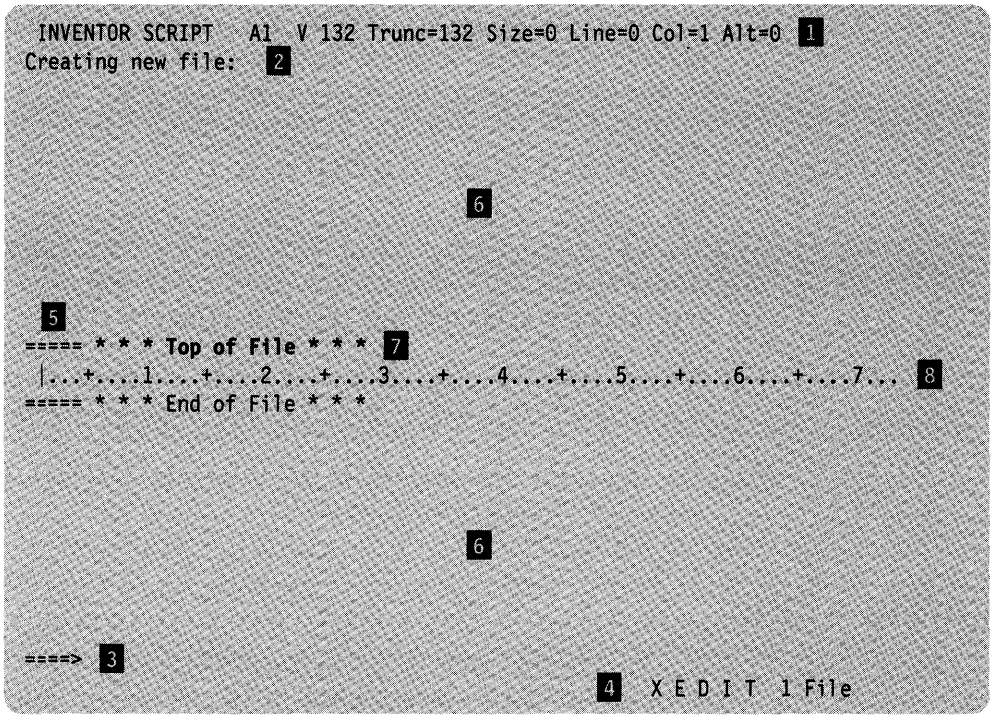


Figure 1. Screen Layout

1 File Identification Line

The first line on the screen identifies the file being edited. The following information is displayed:

1. file name, file type, file mode
   If you do not specify a file mode, the editor assigns a file mode of "A1." The file mode identifies an accessed minidisk or SFS (Shared File System) directory where the file resides.
2. record format and record length
   The record format and record length (V 132) mean that in this file, the length of a line can vary and the file can hold lines up to 132 characters long. Therefore, a file line can be longer than a screen line.
3. truncation column (Trunc=)
   Notice that the truncation column is the same as the record length (132). Because a file line can be only 132 characters long, any data that is entered beyond 132 characters (in total) can be truncated.
4. current number of lines in the file (Size=)
   (Because we have not yet entered data in the file, the number of lines is zero.)
5. file line number of the current line (Line=)
   (See number 7, following.)
6. position of the column pointer (Col=)
   (See number 8, following.)

7. alteration count (Alt=) The alteration count is the number of alterations that have been made to the file since the last AUTOSAVE (which is explained later in this chapter).

2 Message Line

The editor communicates with you by displaying messages on the second and third lines of the screen. These messages tell you if you have made an error, or they provide information. In Figure 1 on page 2, 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 on page 2 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 or PF12.)

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 on page 2 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 short 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 permanently records those changes on the copy of the file that resides on disk or directory.

Because a file can be too long to fit on one screen, various subcommands scroll the screen so 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; we say it is "highlighted."

In Figure 1 on page 2, 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 is 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 are discussed later. The column under which the column pointer is positioned is called the current column.

XEDIT and Full-Screen CMS

If you invoke XEDIT from full-screen CMS, the way you see messages sent to you by other users is not the same as when full-screen CMS is off. When you receive a message while full-screen CMS is off, the message appears on a cleared screen with a `HOLDING` status at the bottom. You can press the CLEAR key to get the XEDIT screen back.

If full-screen CMS is on, then any message you receive will appear in the message window which automatically pops up on top of your XEDIT screen. To scroll forward in the message window, type an "f" (forward) in one of the border corners (indicated by '+' signs) and press the ENTER key. Continue to use the "f" border command until you have seen all the information in the message window. When there is no more information to be displayed, the window is automatically removed from your screen.

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, enter the following subcommand on the command line:

```
====> 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 2 on page 5 through Figure 4 on page 6 is the same file, INVENTOR SCRIPT, that is shown in Figure 1 on page 2. 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 2, 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 3 on page 6.

```
 INVENTOR SCRIPT   A1  V 132  Trunc=132 Size=9 Line=0 Col=1 Alt=0
Input mode:




* * * Top of File * * *
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....
THE ELECTRONIC COMPUTER (1946)

 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 2. Input Mode — Typing the Data

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 4 on page 6 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 on page 2, 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, because 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 use input mode to insert lines between any two existing lines in the file.

```
 INVENTOR SCRIPT   A1  V 132  Trunc=132 Size=18 Line=9 Col=1 Alt=9

* * * Top of File * * *
THE ELECTRONIC COMPUTER (1946)

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 3. Input Mode — Continue Typing

```
 INVENTOR SCRIPT   A1  V 132  Trunc=132 Size=10 Line=10 Col=1 Alt=10
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 4. Input Mode — Data Entered in the File

POWERINP 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 are a skilled typist, you do not 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 POWERINP 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

To cause a break in the data you have entered in power typing mode, type a line end character just before the place in the text where you want the break to occur. The default line end character is a pound sign (#). Use the pound sign to signify the start of a new paragraph or to set off a SCRIPT/VS control word.

For example, suppose 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.

Example of Power Typing

Figure 5 on page 8 illustrates the same file, INVENTOR SCRIPT, but the data was typed in power typing mode, after the POWERINP 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 POWERINP 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 does not 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.

To leave power typing mode and return to the XEDIT environment, press the ENTER key twice. The screen layout is restored, and the words and lines are reconstructed. The lines you typed are entered and the pound sign line separator is interpreted. Any data that was preceded by a pound sign begins on a new line. The last line entered becomes the current line. To display the entire file on your screen, change the current line to a point above the End of File line by placing a slash (/) in the prefix area and pressing the ENTER key. Figure 6 on page 9 illustrates the same file, INVENTOR SCRIPT, after returning to the XEDIT environment. The current line was changed by placing a slash in the prefix area of the line beginning, "DOING EXPERIMENTS WITH . . ." and pressing the ENTER key.

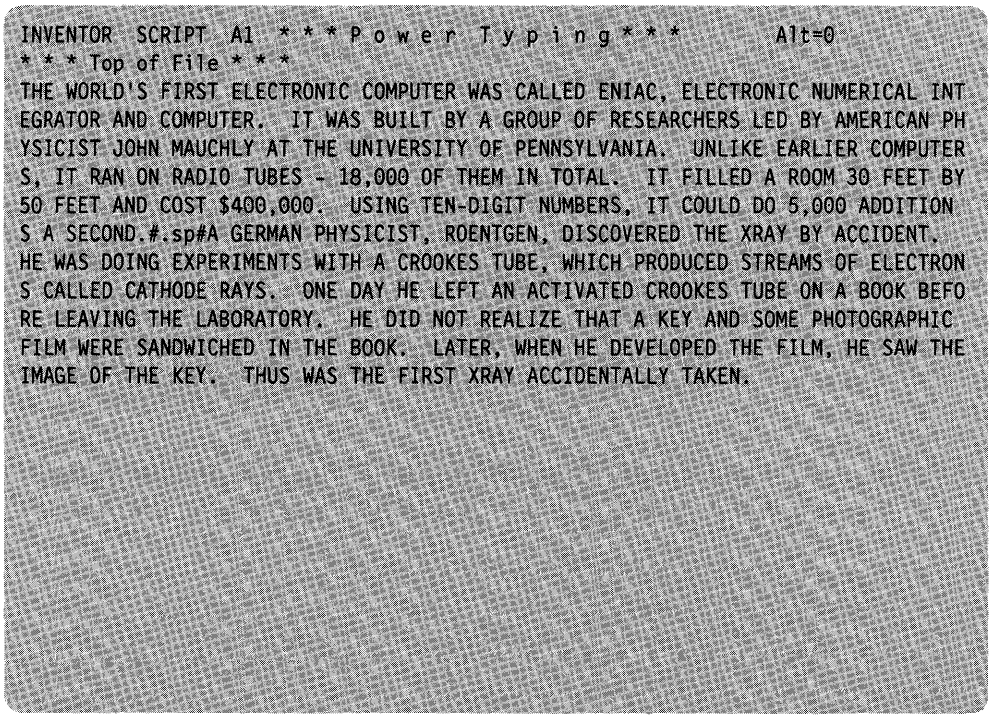


Figure 5. Power Typing

```
 INVENTOR SCRIPT   A1  V 132  Trunc=132 Size=14 Line=9 Col=1 Alt=15

===== 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 6. Power Typing — Data Entered in the File

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 following subcommands are initially assigned to the PF keys:

PF1 BEFORE HELP MENU
PF2 BEFORE SOS LINEADD
PF3 BEFORE QUIT
PF4 BEFORE TABKEY
PF5 BEFORE SCHANGE 6
PF6 ONLY ?
PF7 BEFORE BACKWARD
PF8 BEFORE FORWARD
PF9 ONLY =
PF10 BEFORE RGTLEFT
PF11 BEFORE SPLTJOIN
PF12 BEFORE CURSOR HOME
PF13 BEFORE HELP MENU
PF14 BEFORE SOS LINEADD
PF15 BEFORE QUIT
PF16 BEFORE TABKEY

PF17 BEFORE SCHANGE 6
PF18 ONLY ?
PF19 BEFORE BACKWARD
PF20 BEFORE FORWARD
PF21 ONLY =
PF22 BEFORE RGTLEFT
PF23 BEFORE SPLTJOIN
PF24 BEFORE CURSOR HOME

These are the subcommands that the editor assigns to the PF keys. Note that the editor assigns keys 13 through 24 to correspond to keys 1 through 12 (For example, both PF1 and PF13 are set to BEFORE HELP MENU.) For information on the "BEFORE" and "ONLY" operands shown previously, see the SET PF subcommand in the *VM/SP System Product Editor Command and Macro Reference.*

If full-screen CMS is on, the PF key definitions appear in the CMSOUT window which automatically pops up on top of your XEDIT screen. The top line of the CMSOUT window reminds you that full-screen CMS is on. To scroll forward in the CMSOUT window, type an "f" (forward) in one of the border corners (indicated by '+' signs) and press the ENTER key. Continue to use the "f" border command until you have seen all the information in the message window. When there is no more information to be displayed, the window will be automatically removed from your screen.

If full-screen CMS is off, the PF key definitions appear on a cleared screen with a "MORE" status at the bottom. You can press CLEAR to return to the XEDIT screen.

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:

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

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

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

The PF11 key lets you split a line or join two lines, *at the cursor position.* If the cursor is positioned *before* (or at) the last character in a line, the line is *split.* If the cursor is positioned *after* the data, the next line is *joined* to it.

Splitting a Line (PF11)

To split a line in two, simply move the cursor under the character where you want the line to be split, and press the PF11 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 PF11 key produces the following lines:

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

The PF11 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 PF11 key is pressed, the line is split in two:

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

Now there is 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, when the cursor is positioned after the end of the data in a line.

For example:

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

Note the cursor position above, after the end of the data. Pressing the PF11 key produces the following line:

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

The PF11 key also takes care of leading blanks when a line is split or joined.

For example:

```
=====      Things get worse under pressure.
```

When the line is split, the second line lines up under the first:

```
=====      Things get worse _
=====      under pressure.
```

The same is true when lines are joined:

```
=====      Join these _
=====      lines without leading blanks.
```

The leading blanks are removed:

```
=====      Join these lines without leading blanks.
```

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 repeatedly press either key 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. Sometimes, you would like to be able to see the last subcommand that was executed. Perhaps you did not 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 (from the command line).

You can then re-execute the subcommand simply by pressing the ENTER key. If the subcommand was incorrectly entered, 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

The PF9 key, which is set to the = subcommand, re-executes 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 retype the subcommand.

Inserting Words in a Line

Using the Insert Mode Key and a NULL Key (PA2)

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 is initially set to NULLKEY. For information about how to change the initial PA key settings (SET PAn), see the *VM/SP System Product Editor Command and Macro Reference.*

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 file line at a time; if you move the cursor to another file 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 can be used in both input mode and edit mode, but not in power typing mode.

Using the SET NULLS Subcommand

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

```
====>  set nulls off
```

(In power typing mode, you can use the insert mode key without entering a SET NULLS ON subcommand and without pressing the PA2 key.)

Using Prefix Subcommands

Prefix subcommands are one- or two-character commands that perform basic editing tasks on a particular line.

The following prefix subcommands are described in this section:

A (add)
D (delete)
SI (structured input)
" (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 immediately inserted 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 can 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 or directory.

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 block of lines is deleted.

The first and last lines of the block need not be on the same screen; you can 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 `'DD' pending...`. 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 7 on page 15 is a before-and-after example of the A and D prefix subcommands.

```
 ANIMALS  FACTS    A1  F 80  Trunc=80 Size=14 Line=9 Col=1 Alt=0

===== * * * 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 ARE EQUIPPED WITH 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 Col=1 Alt=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 7. Prefix Subcommands A and D — "Before" and "After"

Lost and Found Department

If you delete one or more lines, you can recover them anytime 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.

Recovered lines are inserted starting at the current line. The last lines deleted are the first lines recovered. If the lines were deleted from different places in the file, you will have to put them back where they belong (by using the M prefix subcommand, discussed later.)

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

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

In the previous example of the A and D prefix subcommands, four lines were deleted. Entering,

```
recover 2
```

results in:

```
 ANIMALS  FACTS    A1  F 80  Trunc=80 Size=15 Line=9 Col=1 Alt=2
2 line(s) recovered
===== * * * 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.
===== ANTS ARE EQUIPPED WITH FIVE DIFFERENT NOSES. EACH ONE IS DESIGNED TO
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== ACCOMPLISH A DIFFERENT TASK.
===== 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
```

Figure 8. RECOVER Subcommand — Replacing Two Lines

SI Prefix Subcommand

To continuously add lines of indented text, type the characters "SI" in the prefix area. When the ENTER key is pressed, a line is immediately added following the line that contains "SI." The cursor is positioned at the same column where the text on the previous line begins, making it easier for you to enter indented text.

Figure 9 on page 17 shows how the first new line is added.

The prefix subcommand "SI" is typed in the prefix area.

```
CHOCOLAT COOKIES  A1  F 80  Trunc=80 Size=6 Line=6 Col=1 Alt=0

===== * * * Top of File * * *
===== Chocolate-Nut Cookie Ingredients
=====
=====
SI===          1/2    Pound of butter
=====          1 1/2  Cups of graham cracker crumbs
=====          3 1/2  Ounces coconut flakes
               2      Ounces chopped nuts
     |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== * * * End of File * * *

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

When ENTER is pressed a new line is added.

```
CHOCOLAT COOKIES  A1  F 80  Trunc=80 Size=7 Line=6 Col=1 Alt=0

===== * * * Top of File * * *
===== Chocolate-Nut Cookie Ingredients
=====
=====
=====          1/2    Pound of butter
.....          1 1/2  Cups of graham cracker crumbs
=====          3 1/2  Ounces coconut flakes
     |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
=====          2      Ounces chopped nuts
===== * * * End of File * * *

====>                                                    '.....' pending...
```

Figure 9. Prefix Subcommand SI — Adding the First New Line

Text is entered on the new line.

```
CHOCOLAT COOKIES  A1  F 80  Trunc=80 Size=7 Line=6 Col=1 Alt=1



===== * * * Top of File * * *
===== Chocolate-Nut Cookie Ingredients
=====
=====          1/2    Pound of butter
=====          1 1/2  Cups of graham cracker crumbs
.....          8      Ounces sweetened condensed milk_
=====          3 1/2  Ounces coconut flakes
     |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
=====          2      Ounces chopped nuts
===== * * * End of File * * *









====>                                                  '.....' pending...
```

When ENTER is pressed, a new line is automatically added following the one you just typed on. Each time that you type on the new line and press the ENTER key, another new line will be added.

```
CHOCOLAT COOKIES  A1  F 80  Trunc=80 Size=8 Line=6 Col=1 Alt=1



===== * * * Top of File * * *
===== Chocolate-Nut Cookie Ingredients
=====
=====          1/2    Pound of butter
=====          1 1/2  Cups of graham cracker crumbs
=====          8      Ounces sweetened condensed milk
.....          _
     |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
=====          3 1/2  Ounces coconut flakes
=====          2      Ounces chopped nuts
===== * * * End of File * * *









====>                                                  '.....' pending...
```

Figure 10. Prefix Subcommand SI — Continuing to Add New Lines

If you do not want to add more lines, then press the ENTER key one more time without typing anything on the new line.

```
CHOCOLAT COOKIES  A1  F 80  Trunc=80 Size=7 Line=6 Col=1 Alt=1


===== * * * Top of File * * *
===== Chocolate-Nut Cookie Ingredients
=====
=====          1/2    Pound of butter
=====          1 1/2  Cups of graham cracker crumbs
=====          8      Ounces sweetened condensed milk
=====          3 1/2  Ounces coconut flakes
     |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
=====          2      Ounces chopped nuts
===== * * * End of File * * *




====>                                                  XEDIT 1 File
```

Figure 11. Prefix Subcommand SI — "After"

To add a blank line in a file while using SI, make at least one change on the line that contains ". . . . .". (Pressing the spacebar once will change the line.) Merely moving the cursor using the cursor position keys over a line does not change the line.

You can leave the line you are adding and make corrections elsewhere in the file if you type something on the new line first. When you press the ENTER key while the cursor is away from the new line, another new line is added following the last line that was added. SI will only be canceled if you press ENTER and no text has been typed on the new line.

Duplicating Lines

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

A number can 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 '""' pending.... This lets you scroll the screen before completing the block and pressing the ENTER key.

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 can precede or follow the "M" to indicate that more than one line is to be moved, for example, "5M" or "M5" in the prefix area.

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 a pending notice. This pending status lets you 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 can 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 12 on page 21 is a before-and-after example of the M prefix subcommand.

```
 ANIMALS  FACTS    A1  V 132  Trunc=132 Size=22 Line=10 Col=1 Alt=0

===== 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 Col=1 Alt=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 12. Prefix Subcommands M and F — "Before" and "After"

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 that inserts 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 are 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 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 equal signs (= = = = =).

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

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. Enter the TOP subcommand this way:

```
====> top
```

The BOTTOM subcommand makes the last line of the file the new current line. Enter the BOTTOM subcommand this way:

```
====> 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 POWERINP 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 13 on page 24 is a before-and-after example of the DOWN subcommand.

```
 PURIST   SCRIPT   A1  V 132  Trunc=132 Size=12 Line=5 Col=1 Alt=0


===== * * * 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 Col=1 Alt=0

===== "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 13. The DOWN Subcommand — "Before" and "After"

Making Changes in a File

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

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

The challenge is twofold: 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 column after the current column in the current line, for a character string that you specify.

If the string is located, two things happen:

1. 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.
2. 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 Col=nn).

One format of the CLOCATE subcommand is as follows:

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

The string should be enclosed in delimiters. In the examples used in this book, the delimiter is a diagonal (/); however, you can use any character except for a plus (+), minus (−), not (¬), or period (.) 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....

====>  clocate/be/

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

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 repeatedly enter the CLOCATE/be/ subcommand (or use the PF9 key, which is set to the = subcommand, for repeated execution). If a string appears more than once

in a line, as in the preceding example, 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....
```

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 are 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 do not 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 repeatedly locating the string you want to change, and by entering a CHANGE subcommand only when you want to change the string. However, there is 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 is 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 *do not 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 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 automatically scrolls the screen forward.

Figure 14 on page 28 is an example of using the PF5 and PF6 keys to locate and selectively change 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 successively pressed 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 do not 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.

For more information on making a selective change, see "SCHANGE" in the *VM/SP System Product Editor Command and Macro Reference.*

```
 ROSE     PETALS   A1  F 80  Trunc=80 Size=11 Line=1 Col=3 Alt=0
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 Col=23 Alt=1
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 14. Using PF5 and PF6 to Make a Selective Change

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 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 * * *
```

This form of the CHANGE subcommand can also make a global change starting in the middle of a file. The change starts with the current line, so you could use the / prefix subcommand to set the current line at the place 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 lets you 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....+

====>  clocate :4

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

The column pointer has moved to column four.

In the following example, the CLOCATE subcommand moves the column pointer; then the CINSERT subcommand immediately inserts characters before the column pointer position.

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

====>  clocate/,/   or   ====>  clocate :19

(move the column pointer)

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

====>  cinsert  wrong

(insert "wrong" before the column pointer)

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

(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 lets you to insert blanks with the CINSERT subcommand — simply type the required number of blanks (by pressing the spacebar) in the operand. For example:

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

====>  clocate/can/
====>  cinsert
```

(Press the spacebar 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.  You must type one blank after "right"
to avoid "rightthan".)

===== 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 is a good idea to reset the column pointer to column one by entering the CFIRST subcommand.

For example:

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

====> cfirst

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

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 file type; 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 can be used for tabbing in input mode, but not in power typing mode.

You can change the tab settings by entering another SET TABS subcommand, or, if you would 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 15 on page 33 is an example of data that was entered using the PF4 key as a tab key. The following subcommand defines the tab columns:

```
====> set tabs  5 35 45
```

```
 TABS     EXAMPLE  A1  F 80  Trunc=80 Size=13 Line=9 Col=1 Alt=0

===== * * * 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 15. Using the PF4 Key for Tabbing

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 to disk or SFS directory, for *permanent* storage, you must enter the following subcommand:

```
====> file
```

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

QUIT Subcommand

The QUIT subcommand ends an editing session and leaves the permanent copy of the file intact on the disk or directory. If the file is new, it is not written to disk or directory.

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 do not want to save the file, enter "QQUIT" (abbreviated as "QQ"). If you wish to save the changes, enter "FILE".

SET AUTOSAVE Subcommand

Files on disks or directories are not affected if the system malfunctions. However, a new file that you are creating or the changes you are making to an existing file might be lost if the system fails. You can minimize the risk of losing your data by using the SET AUTOSAVE subcommand, whose format is as follows:

```
SET AUTOSAVE n
```

The SET AUTOSAVE subcommand causes your file to be automatically written to disk or directory after you have 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 directory, or "saved," every time you have changed 10 lines, enter the following subcommand:

```
set autosave 10
```

The number of alterations you have made to your file since the last AUTOSAVE are displayed in the alteration count (Alt=n) in the file identification line. When the alteration count is equal to the AUTOSAVE setting, the file is saved on disk or directory and the alteration count is reset to zero.

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

When a file is saved on disk or directory by the automatic save function, it is written into a new file. The file name of this file is a number and its file type is AUTOSAVE. If the system malfunctions during an editing session, you can recover all changes made up to the time of the last automatic save. To do this, replace the original file with the AUTOSAVE file using the CMS COPYFILE command with the REPLACE option. Then, erase the AUTOSAVE file and resume editing.

If you enter a SET AUTOSAVE subcommand while you are creating a new file, and then enter a QUIT subcommand, the file is not saved. However, the AUTOSAVE file is available from disk or directory. If you enter a SET AUTOSAVE subcommand while you are revising an existing file and then you enter a QUIT subcommand, no revisions are saved. However, the AUTOSAVE file is still available from disk or directory.

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 16 on page 36 shows how the GET subcommand inserts 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.

```
 DESSERT  COOKBOOK A1  F 80  Trunc=80 Size=8 Line=9 Col=1 Alt=0
===== * * * 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 Col=1 Alt=1
EOF reached
=====          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 16. Inserting a 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 10 lines of a second file:

```
====> get file2 data 1 10
```

If you do not 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 us 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 lets you 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 can 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 four 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/
   ```

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

   ```
   ====> set number on
   ```

The resulting lines might look like this:

```
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 17 on page 39 through Figure 20 on page 40 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 (use the / prefix subcommand) in the file DESSERT COOKBOOK. Because the sauce recipe must follow the cream puffs recipe, the current line is the last line of the cream puffs recipe (Figure 17 on page 39). 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. Use the UP subcommand or the / prefix subcommand to move the line pointer to the beginning of the lines to be inserted. The beginning line contains "CHOCOLATE SAUCE" (Figure 18 on page 39). Now enter the subcommand to store the chocolate sauce recipe:

```
====> put/VINAIGRETTE
```

The stored lines 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 entered. After the PUT subcommand is entered, you can quit this file by entering:

```
====> quit
```

The original file comes back on the screen (Figure 19 on page 40). Now enter the following subcommand to insert the lines that were "put":

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

The sauce recipe is inserted, as shown in Figure 20 on page 40. The last line inserted is the new current line.

```
DESSERT  COOKBOOK A1  F 80  Trunc=80 Size=15 Line=8 Col=1 Alt=0

===== * * * 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 17. Inserting Part of a File — Call Out the Second File

```
SAUCES   COOKBOOK A1  F 80  Trunc=80 Size=20 Line=8 Col=1 Alt=0

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 18. Inserting Part of a File — Put Lines to Be Inserted, Then QUIT

```
DESSERT  COOKBOOK A1  F 80  Trunc=80 Size=15 Line=8 Col=1 Alt=0

===== * * * 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 19. Inserting Part of a File — GET

```
DESSERT  COOKBOOK A1  F 80  Trunc=80 Size=22 Line=15 Col=1 Alt=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 20. Inserting Part of a File — The Lines Are Inserted

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 ENTER. The subcommand description appears on the screen, replacing the HELP Menu. Pressing the PF3 key returns you to the previous screen and pressing PF4 takes you out of the HELP display and restores your file on the screen.

Learning More about the Editor

The following is a partial list of XEDIT subcommands and macros that are useful in text processing. You can learn how to use these and other XEDIT subcommands and macros by using the Help facility or by referring to the *VM/SP System Product Editor Command and Macro Reference*.

ALL
: Lets you collect lines for editing, while excluding others from display.

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

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

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

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

MERGE
: Lets you combine two sets of lines.

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

SET CASE
: Lets you 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
: Lets you assign name(s) to any line; you can reference the name(s) in XEDIT subcommands.

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

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

SORT
Lets you arrange the file lines in alphabetical order.

< (SHIFT LEFT) MACRO
A prefix macro that lets you shift left one line or a block of lines one or more columns to the left.

> (SHIFT RIGHT) MACRO
A prefix macro that lets you shift right one line or a block of lines one or more columns to the right.

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**
To enter data	Input **POW**erinp
To scroll the screen	**BA**ckward **FO**rward **TOP B**ottom
To set PF keys	**SET PF**n
To display current PF key settings	**Q**uery **PF**n
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 specify whether trailing blanks are replaced with nulls to allow character insertion	**SET NUL**ls **ON**
To end an editing session without saving the changes	**QUIT**
To automatically save a file 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**

Function	Subcommand/PF Key
To cancel pending prefix subcommands	**RESet**

Function **Prefix Subcommands:**	Subcommand/PF Key
To add lines	**A**
To delete lines	**D**
To add lines and position cursor for indented text	**SI**
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, PF13**
To add a line	**PF2, PF14**
To end a session without saving	**PF3, PF15**
To use a tab key	**PF4, PF16**
To locate and selectively change	**PF5, PF17; PF6, PF18**
To redisplay a subcommand	**PF6, PF18**
To scroll one screen backward	**PF7, PF19**
To scroll one screen forward	**PF8, PF20**
To repeat previous subcommand	**PF9, PF21**
To move the display to the right, and move it back when key is pressed again	**PF10, PF22**
To split or join lines at the cursor	**PF11, PF23**
To move the cursor from the screen to the command line, or vice versa	**PF12, PF24**

Chapter 2. Practice Exercises

The goal of this chapter is to give you practice in using some of the XEDIT subcommands discussed in Chapter 1.

There are five exercises in the chapter. 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 file name is INVENTOR; the file type 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 on page 2. Notice that the cursor is positioned on the command line, after the large arrow (= = = = >).

To cause your file to be written to disk or directory 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 file type, enter:

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

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

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

Now you are 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:

```
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: POWERINP, TOP, BOTTOM, UP, DOWN, /, the PF11 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 are 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. You are now in edit mode.

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 PF11 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 spacebar once. The sentence has moved over to accommodate 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. If your cursor is not on the command line press PF12 to bring it down to the command line and now 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 to disk or directory:

```
====>  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 us 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 reenter edit mode.

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

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

To change the first occurrence of "Bitter," enter:

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

Now you are 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 file type when you GET a file if the file type 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 or directory; only a copy of it has been inserted.

Now you are 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 are 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 are 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 the file has been changed and to enter QQUIT if you want to quit anyway. Enter:

```
====> qquit
```

Chapter 3. Using the Editor on a Typewriter Terminal

This chapter is primarily written 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 *VM/SP System Product Editor Command and Macro Reference*.

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

Editing a File

Editing involves changing, adding, or deleting data within a CMS file. The editor lets you 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 and has a file mode of A, 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 entered in either uppercase or lowercase, or a combination of both.)

If the file does not already exist, 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 us create a simple file. Its file name and file type 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 messages:

```
Creating new file:
XEDIT:
```

Entering Data

The following subcommands are discussed in this section:

INPUT
QUERY LRECL.

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 can be longer than the logical record length of the file, which varies according to file type. 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 file type 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 us 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.*

The editor comes back with:

```
XEDIT:
```

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, because 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, display the data 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. Underscores will only show on terminals that allow underscoring (like a 2741). 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 preceding example, 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 calls 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:

1. Position the line pointer at the line you want to edit.
2. Edit the line (change characters in it, delete it, or insert new lines following it).
3. 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 target
```

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 five 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 us 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 displayed the entire file; because 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 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:

- Enter 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 beginning or end of the file.

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

```
top
```

To move the line pointer to the last file line, enter 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 characters in a line or change one word to another. The subcommands discussed in this section let you 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 moves 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 should be enclosed in delimiters. The diagonal (/) is the delimiter used in these examples, but it can be any character (except for a plus (+), minus (−), not (¬), or period (.)) 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 in the previous example, 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 is 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," entering a CFIRST subcommmand results in:

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

CINSERT Subcommand

The CINSERT subcommand inserts 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. Note that the spacebar 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 us look at another example. The CLOCATE subcommand moves the column pointer; then the CINSERT subcommand inserts 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, 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 lets you insert blanks with the CINSERT subcommand — simply type the required number of blanks (by pressing the spacebar) in the operand. For example:

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

```
cinsert
```

(Press the spacebar 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 deletes 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". Because 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 entered 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 can be specified as the number of characters to be removed, or it can be specified as a character string. After the CDELETE subcommand is executed, the editor displays the changed line.

CAPPEND Subcommand

The CAPPEND subcommand appends 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 do not 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, entering 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 enter 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 spacebar at least twice before pressing the RETURN key. A blank line is 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, enter the following subcommand:

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

Now the file looks like this (TOP and TYPE 6 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 enter the following subcommands:

`up`

(move the line pointer up one line)

`input`

(press the spacebar 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

The DELETE subcommand deletes 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 do not 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 is left is:

```
a loaf of bread
a jug of wine
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 last lines that were deleted are the first lines to be recovered. For instance, in our previous example of deleting lines, if you entered:

```
recover 2
```

you would get the radio and frisbee back:

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

The recovered lines are inserted starting at 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 later.

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

```
recover *
```

Replacing a Line

You have 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 the text you specify.

The format of the REPLACE subcommand is:

```
REPLACE text
```

However, if you enter the REPLACE subcommand with no text, 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 enter 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 enter 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 us look at an example. Suppose a file contains the following lines, and the first line shown is the current line:

```
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/
```

The resulting file looks like this:

```
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.

LPREFIX Subcommand

The LPREFIX subcommand simulates writing in the prefix area of the current line, even though the prefix area is not available on a typewriter terminal. LPREFIX performs some of the functions (for example, moving or copying lines) provided by prefix subcommands and macros on display terminals.

For a description of the LPREFIX subcommand, see the *VM/SP System Product Editor Command and Macro Reference.*

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 to disk or SFS directory, for *permanent* storage, you must enter the following subcommand:

```
file
```

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

QUIT Subcommand

The QUIT subcommand ends an editing session and leaves the permanent copy of the file intact on the disk or directory. If the file is new, it is not written to disk or directory.

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 do not want to save the file, enter "QQUIT" (abbreviated as "QQ"). If you wish to save the changes, enter "FILE".

SET AUTOSAVE Subcommand

Files on disks or directories are not affected if the system malfunctions. However, a new file that you are creating or the changes you are making to an existing file might be lost if the system fails. You can minimize the risk of losing your data by using the SET AUTOSAVE subcommand, whose format is as follows:

```
SET AUTOSAVE n
```

The SET AUTOSAVE subcommand causes your file to be automatically written to disk or directory after you have 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 directory, or "saved", every time you have changed 10 lines, enter the following subcommand:

```
set autosave 10
```

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

When a file is saved on disk or directory by the automatic save function, it is written into a new file. The file name of this file is a number and its file type is AUTOSAVE. If the system malfunctions during an editing session, you can recover all changes made up to the time of the last automatic save. To do this, replace the original file with the AUTOSAVE file using the CMS COPYFILE command with the REPLACE option. Then, erase the AUTOSAVE file and resume editing.

If you enter a SET AUTOSAVE subcommand while you are creating a new file, and then enter a QUIT subcommand, the file is not saved. However, the AUTOSAVE file is still available from disk or directory. If you enter a SET AUTOSAVE subcommand while you are revising an existing file and then you enter a QUIT subcommand, no revisions are saved. However, the AUTOSAVE file is still available from disk or directory.

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 10 lines of a second file:

```
get file2 data 1 10
```

If you do not 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 us 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 lets you 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:

   ```
   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 can 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 four 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 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, which contains recipes for sauces, called SAUCES COOKBOOK. 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 is 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 is 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 entered with ON or OFF as the operand.

(SET IMAGE ON is the initial setting for all file types except SCRIPT, MACLIB, MODULE, and TEXT.)

If SET IMAGE ON is in effect when you press the TAB key, the logical tab settings determine how the data is 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 enter 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 file type, you can enter 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, and SET IMAGE ON is in effect, 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 enter 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:

ABC

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:

ABC

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 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**
To enter data	**Input**
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 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**
To simulate writing in the prefix area of the current line	**LPrefix**

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:

- To change which line is the current line
- To define the operating range of a subcommand's execution.

A target can be entered in the following ways:

1. By itself
2. As the operand of the LOCATE subcommand
3. Before any XEDIT subcommand
4. 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 range of that subcommand's execution. Most XEDIT subcommands *begin* their operation with the current line; the target operand specifies where the operation is to *end*.

The following XEDIT subcommands have target operands:

ALL	EXPAND	REPEAT
ALTER	EXTRACT	SET RANGE
CHANGE	HEXTYPE	SET SELECT
COMPRESS	LOWERCAS	SHIFT
COPY	MERGE	SORT
COUNT	MOVE	STACK
DELETE	PUT	TYPE
DUPLICAT	PUTD	UPPERCAS

See the *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 21 on page 79. When entered on the command line, any of the following targets would change the current line to the one shown in the bottom screen. (The current line is the line above the scale.) All of the following targets 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 21 on page 79 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).

```
TARGET1  SCRIPT   A1  V 132  Trunc=132 Size=14 Line=5 Col=1 Alt=0




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 Col=1 Alt=0

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 21. Using a Target to Move the Line Pointer

A Target as the Operand of a LOCATE Subcommand

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

```
locate :11
locate +6
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.

Using a Target as a Subcommand Operand

When a subcommand format shows that an operand can 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 range 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 22 on page 81 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)

```
 TARGET2  SCRIPT   A1  V 132  Trunc=132 Size=17 Line=10 Col=1 Alt=0

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 * * *

====> UPPERCAS/son/
                                                        X E D I T  1 File
```

```
 TARGET2  SCRIPT   A1  V 132  Trunc=132 Size=17 Line=13 Col=1 Alt=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's say howdy,
00016 For I am a sufferer
00017 Magna cum laude.
00018 * * * End of File * * *




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

Figure 22. Using a Target as a Subcommand Operand

Types of Targets

Let us 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 entering 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 23 on page 83 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.

```
 TARGET3  SCRIPT   A1  V 132  Trunc=132 Size=16 Line=8 Col=1 Alt=0


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 Col=1 Alt=0
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 23. 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 24 on page 85 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.

```
TARGET4  SCRIPT   A1  V 132  Trunc=132 Size=28 Line=10 Col=1 Alt=0

===== 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 Col=1 Alt=0

===== 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 24. A Target as a Relative Displacement

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(s) for a line. The SET POINT subcommand defines a name of one to eight characters, preceded by a period, to the current line. Using the .xxxx prefix subcommand lets 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.

A line name is particularly useful if you plan to refer to a line many times during an editing session. You need to 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. Remember to type the line name exactly as it was when originally assigned to the line; the editor always pays attention to uppercase and lowercase characters when looking for a line name.

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 but not including, 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 OFF or SET NUMBER ON 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 enter the subcommand QUERY POINT to display the name(s) of the current line, or you can use QUERY POINT * to display all names that have been defined during the editing session.

Figure 25 on page 88 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'."

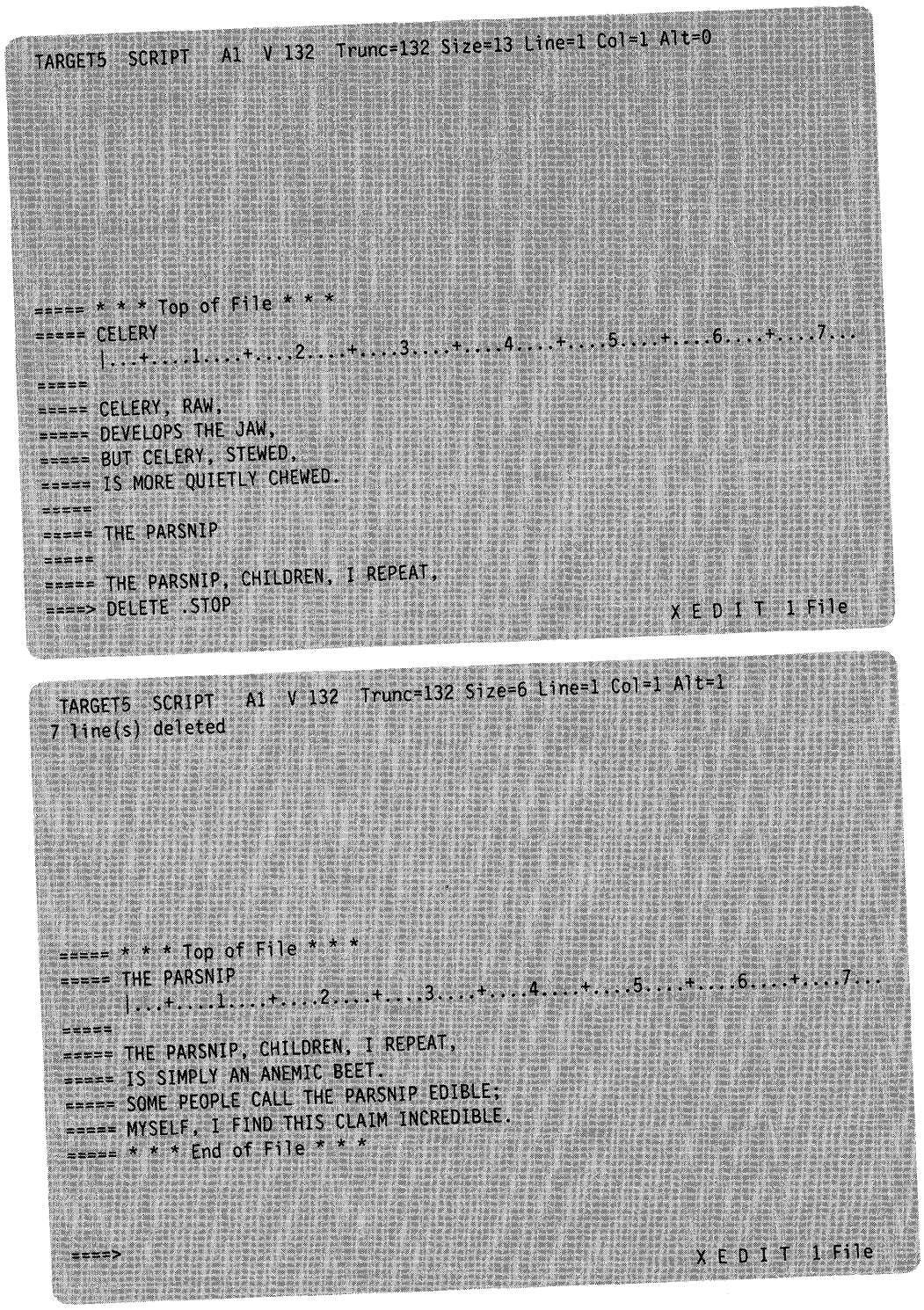


Figure 25. A Target as a Line Name

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.

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 range 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. However, if you use one of the following special characters as a delimiter, you must also specify a search direction (+ or −): plus (+), minus (−), not (¬), or period (.). The search direction is explained below.

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:

1. Determine the direction of the search
2. Search for a line that does *not* contain a given string
3. 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 us 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 but not including, 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 but not including, 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 be comprised of multiple 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
```

Using an "AND" Symbol (&)

You can use an "AND" symbol in the same way that you use the "OR" symbol. The editor searches the file one line at a time and the first line that contains all of the strings specified becomes the current line. For example:

If a file contains the following lines:

```
===== Truffles, Leg of Lamb, Chocolate Mousse
===== Turkey eggs, Leg of Lamb, Savarin
===== Escargot, Leg of Lamb, Bombe
```

the following subcommand:

```
====> locate /Leg of Lamb/&/Bombe/
```

will make the following line current:

```
===== Escargot, Leg of Lamb, Bombe
```

A Summary of Simple String Targets

You have seen how to specify a target as a single string, enclosed in delimiters. You have also seen how a plus or minus sign, a NOT symbol, an OR symbol, and an AND 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. Two or more strings can be separated by OR and/or AND symbols.

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/[|[¬]/string3/]]...
  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.
4. "AND" symbol (ampersand)(Locate the line containing string1 and string2.)
5. "OR" symbol (vertical bar)(Locate the line containing string1 and string2 or string3.)

Examples:

```
====> /horse/
```
searches downward in the file, beginning after 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/ | /hay/
```
searches downward in the file for the first line that contains both "horse" and "house" or that contains "hay," whichever occurs first.

```
====> /horse/|¬/house/
```
searches downward in the file for the first line that contains "horse" *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 entered).

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 26 on page 93 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. A string can be expressed as a "complex string" by associating it with one or more of the following SET subcommand options:

SET ARBCHAR
: lets you specify only the beginning and end of a string, using an arbitrary character to represent all characters in the middle.

SET CASE
: lets you specify whether or not the difference between uppercase and lowercase is to be significant in locating a string target.

SET SPAN
: lets you specify if a string target must be included in one file line or if it may span a specified number of lines.

SET VARBLANK
: lets you 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.)

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.
```

```
TARGET6  SCRIPT   A1  V 132  Trunc=132 Size=8 Line=0 Col=1 Alt=0




=====> * * * 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 Col=1 Alt=0




===== * * * 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 26. A Target as a Simple String Expression

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 a 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 preceding example, 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 can 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 can 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 controls 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 blanks between two words.

Figure 27 on page 96 is a before-and-after example of using a target specified as a complex string expression.

The following subcommands were entered:

```
====>  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 lets the beginning and end be specified; the CASE option lets the string be specified in lowercase even though it appears in the file in both uppercase and lowercase; the SPAN option lets the beginning and end of the string be located on two consecutive lines.

```
 TARGET7  SCRIPT   A1  V 132  Trunc=132 Size=19 Line=10 Col=1 Alt=0

===== 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 Col=1 Alt=0

===== 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 27. A Target as a Complex String Expression

Using Column-Targets

The targets discussed so far affect 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 can be 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....+....7...

====> clocate :6
```

(absolute column number)

```
===== John Keats studied medicine and practiced as an apothecary.
      <...+|...1....+....2....+....3....+....4....+....5....+....6....+....7...
```

Current Line:

```
===== James Joyce was a school teacher in Dublin.
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...

====> clocate +6
```

(relative column number)

```
===== James Joyce was a school teacher in Dublin.
      <...+.|..1....+....2....+....3....+....4....+....5....+....6....+....7...
```

Current Line:

```
===== Herman Melville worked as a customs inspector in N.Y.C.
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...

====> clocate /customs/

===== Herman Melville worked as a customs inspector in N.Y.C.
      <...+....1....+....2....+...|3....+....4....+....5....+....6....+....7...
```

Current Line:

```
===== Charles Dickens served as a law clerk and was a reporter.
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...

====> clocate /reporter/|/clerk/
```

(searches for "reporter," even if "clerk" occurs first. If "reporter" is not found, then searches for "clerk").

```
===== Charles Dickens served as a law clerk and was a reporter.
      <...+....1....+....2....+....3....+....4....+...|5....+....6....+....7...
```

The CLOCATE subcommand scans the file, starting with the column following (or preceding, depending on the search direction) 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 another subcommand that operates based on the position of the column pointer can be used.

The following is a list of all subcommands that operate based on the position of the column pointer.

CAPPEND
: Appends text to the end of the current line, and moves the column pointer under the first character of 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
: Selectively replaces characters in corresponding positions 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....+....7...

====> 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....+....7...

====> cdelete /,/
```

(Delete from column pointer up to, but not including, the comma.)

```
===== If anything can go, it will.
      <...+....1....+...|2....+....3....+....4....+....5....+....6....+....7...
```

Chapter 5. Editing Multiple Files

The XEDIT Subcommand

When you enter the CMS command XEDIT, a copy of the specified file is brought into virtual storage, where it remains until you enter 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, see the *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 enter an XEDIT subcommand with a new file ID, 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 entered 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 28 illustrates a ring of files in storage.

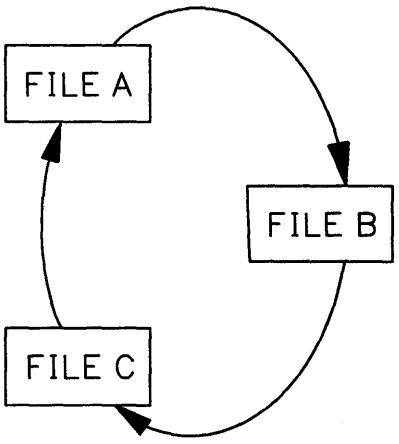


Figure 28. A Ring of Files in Storage

By entering 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 enter the XEDIT subcommand without operands, the next file in the ring appears on the screen. (See Part 1 of Figure 29 on page 103.) Therefore, a series of XEDIT subcommands entered without operands lets you 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 entering the XEDIT subcommand with the file ID 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 Part 2 of Figure 29 on page 103.)
- If you enter the XEDIT subcommand and specify the file ID of a new or existing file that is not already in the ring, that file is added to the ring just after the current file and is displayed. (See Part 3 of Figure 29 on page 103.)

Ending an Editing Session

When you are finished editing a particular file, you can enter 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
```

Entering the CANCEL macro is equivalent to entering 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 enter 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 screen into multiple logical screens. The screen can be split vertically, horizontally, or in a combination of vertical and horizontal segments. 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 an independent terminal with its own file identification line, command line, and message line. For more information about multiple logical screens, see the *VM/SP System Product Editor Command and Macro Reference*.

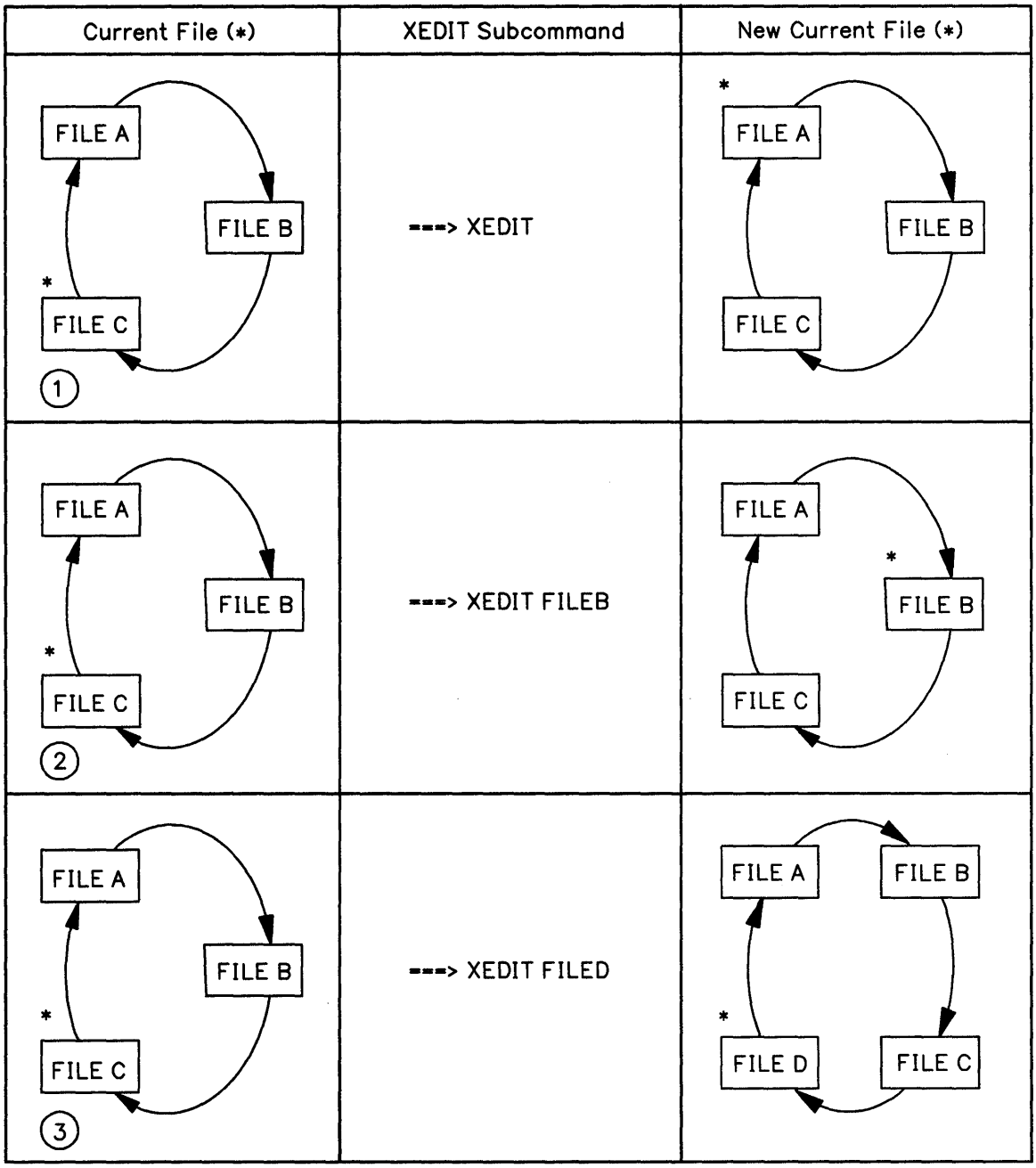


Figure 29. Editing Files in the Ring

SET SCREEN Subcommand

Entering the command, SET SCR 2, will split the screen horizontally into two screens, one on top of the other.

The command SET SCR 2 V, will split the screen vertically into two screens, one beside the other.

Using the SIZE option with the SET SCREEN subcommand allows you to create horizontal screens with the number of lines that you specify. For example, SET SCR SIZE 14 10, will create two screens, one with 14 lines, and another with 10 lines.

Likewise, the WIDTH option is used to specify the number of columns each vertical screen will contain. If SET SCR WID 25 25 30 is entered, 3 vertical screens are created, the first with 25 columns, the second with 25 columns, and the third with 30 columns.

The initial setting of the SCREEN option is SCREEN SIZE *n*, where *n* is the screen size.

To return to the initial setting, enter the following subcommand:

```
====> set screen 1
```

For more information about this command, see the *VM/SP System Product Editor Command and Macro Reference*.

Multiple Views of the Same File

If only one file is in virtual storage and you enter a SET SCREEN subcommand, identical views of the file appear on the screen.

Figure 30 on page 105 is a before-and-after example of a SET SCREEN subcommand that creates two 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. You can type related prefix subcommands in different views of a file, even when different parts of the file are displayed. For example, you can type a "C" (copy) prefix subcommand in one view, and a "P" (preceding) prefix subcommand in another view. 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 entered. 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.

Multiple Views of Different Files

When multiple files are being edited and you enter a SET SCREEN subcommand that increases the number of logical screens, the additional screens are immediately filled with files selected from the ring.

Figure 31 on page 107 illustrates how additional logical screens are filled with files from the ring. The ring of files contains files named FILE1 and FILE2; the current file is FILE1. The SET SCREEN subcommand shown in the top screen causes another file to be displayed.

If a SET SCREEN subcommand decreases the number of logical screens, files are displayed as long as logical screens are available. Those files for which logical screens are not available are removed from the display.

Entering an XEDIT subcommand from one of multiple screens is just like entering it when there is only one screen. It does not affect the other logical screens. In all cases, the file is displayed only on the screen from which the XEDIT subcommand was entered.

The status area of all the screens displays the number of files in virtual storage, not the number of screens.

```
 NASH     SCRIPT   A1  V 132  Trunc=132 Size=6 Line=6 Col=1 Alt=0



===== * * * 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 Col=1 Alt=0

===== 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 Col=1 Alt=0

===== 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 30. Multiple Horizontal Views of the Same File

Order of Processing

You can type over the data, type subcommands on the command line, and type prefix subcommands and macros in the prefix area of all views of a file(s) before pressing a key (like the ENTER key) that effects the changes.

The editor processes requests typed on different views in the following order:

1. Changes typed over the data in all the views are made first. Changes are processed in the order that the data lines appear on the physical screen, from the top, moving left to right, to the bottom.
2. Prefix subcommands and macros are executed next, as follows:

   Prefix subcommands and macros are also scanned in the order that they appear on the virtual screen. As they are scanned, they are placed in a "pending list." Once the scanning is complete, the pending list is executed. Only one pending list is maintained for each file, regardless of the number of views of that file. All views of the file will be updated to reflect the changes.

   The pending list is executed from the first view of each file or from the view that contains the cursor, if any view does. This means that all messages from prefix subcommands and macros will be displayed in the screen from which the pending list was executed. Cursor positioning for prefix subcommands and macros is determined by what lines are displayed in the screen with the cursor. Note that when multiple files are displayed, one pending list is executed for each file, and all views reflect the changes. See other CURSOR considerations below.

   For more information on the pending list, see Chapter 7 in this book.
3. Subcommands typed on the command lines are executed last and in the following order:

   With multiple horizontal screens, the command lines are processed from the top view to the bottom view. With multiple vertical screens, the command lines are processed left to right. With a combination of horizontal and vertical screens, the command lines are processed in the same order that the screens were defined in the SET SCREEN DEFINE subcommand.

Cursor Considerations

The cursor remains in the view that contained it when the ENTER key (or PA/PF key) was pressed. This is true even if a CURSOR subcommand is entered in another view. If no view of a file contained the cursor (for example, if part of the screen was left undefined and your cursor was positioned there), then the cursor is placed in the first logical screen on the screen (the top-most screen for horizontal views, the left-most screen for vertical views, or the first view defined with SET SCREEN DEFINE).

You can move the cursor from one logical screen to another by entering SOS TABCMDF or SOS TABCMDB.

For more information on the SET SCREEN subcommand, see the *VM/SP System Product Editor Command and Macro Reference*.

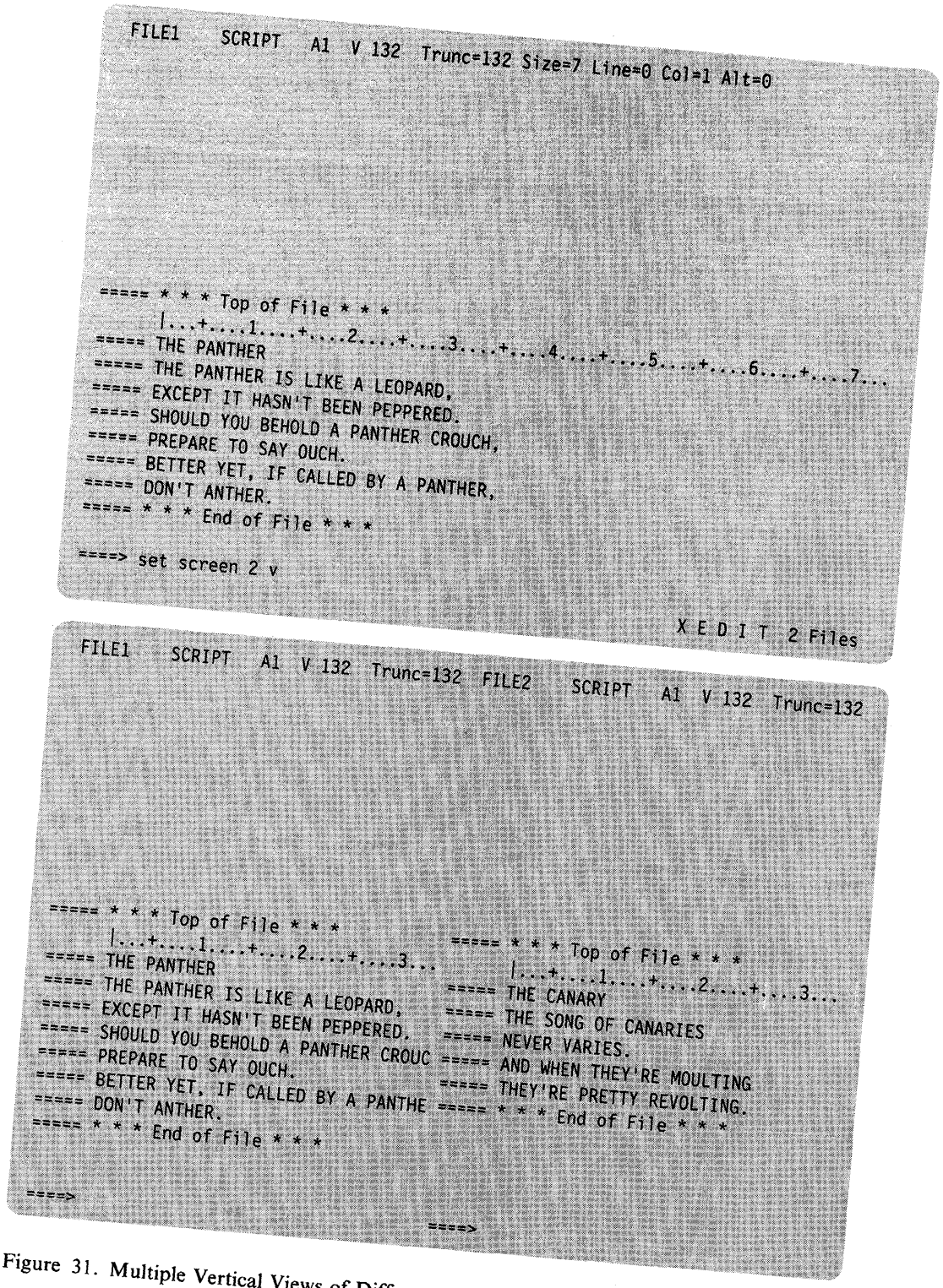


Figure 31. Multiple Vertical Views of Different Files

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 MSGLINE
SET CURLINE
SET SCALE
SET TABLINE
SET COLOR
SET NUMBER.

For a complete description of these options, see 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 or you can set NULLS in the prefix area. Initially, the prefix area is displayed on the left.

Command Line

Use the SET CMDLINE subcommand to move the command line to the top (the second line of the screen), to the bottom (the last line of the screen), or to remove the command line from the screen. Initially, the command line is the last two lines of the screen. If you move the command line to the top, bottom, or off, the status area is not displayed.

With SET CMDLINE TOP (command line on line 2) and the default SET MSGLINE setting (line 2), a message overlays the command line, including the arrow. You must press the ENTER or CLEAR key to recover the command line. To avoid this situation, assign the message line to line 1 or line 3 when using CMDLINE TOP.

Message Line

Use the SET MSGLINE subcommand to define the location of the message line on the screen, and the number of lines the message may expand to, to avoid clearing the screen to display the message. It may also be used to override the blank line that is normally displayed on the screen for messages.

Current Line

Use the SET CURLINE subcommand to establish the position of the current line on the screen. Initially, the current line is in the middle of the screen.

Remember that the editor uses the first line of the screen for the file identification line. Therefore, if you want the current line to be the first available screen line, use the subcommand SET CURLINE ON 2.

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.

Color

Depending on the features supported by your terminal, you can use the SET COLOR subcommand to associate specific colors, highlighting, extended highlightings, and programmed symbol set features with various physical locations on the screen. The physical locations include the arrow, current line, file area, prefix area, command line, scale line, tab line, file identification line, pending message display area, shadow line, status area, top of file and end of file lines, and the message line. Colors associated with those areas can be: blue, red, pink, green, turquoise, yellow, white, or your default terminal display color. You can accentuate this capability by using programmed symbol sets or extended highlighting features such as blinking, reverse video, and underlining.

For a complete explanation of this function refer to the publication, *VM/SP System Product Editor Command and Macro Reference.*

Number

Use the SET NUMBER subcommand to specify whether the prefix area should contain line numbers. Initially, equal signs are used.

Figure 32 on page 111 through Figure 37 on page 116 illustrate how some of 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 Col=1 Alt=0

===== * * * 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 Col=1 Alt=0

* * * 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 32. SET PREFIX Subcommand — "Before" and "After"

```
 TAILOR   SCRIPT   A1  V 132  Trunc=132 Size=28 Line=9 Col=1 Alt=0

* * * 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 Col=1 Alt=0
====>
* * * 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 33. SET CMDLINE Subcommand — "Before" and "After"

```
 TAILOR   SCRIPT   A1  V 132  Trunc=132 Size=28 Line=9 Col=1 Alt=0
====> 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 Col=1 Alt=0
====>                                                            =====
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 34. SET CURLINE Subcommand — "Before" and "After"

```
 TAILOR   SCRIPT   A1  V 132  Trunc=132 Size=28 Line=9 Col=1 Alt=0
====> 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 Col=1 Alt=0
====>
                                                                      =====
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 35. SET SCALE Subcommand — "Before" and "After"

```
 TAILOR   SCRIPT   A1  V 132  Trunc=132 Size=28 Line=9 Col=1 Alt=0
====> 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 Col=1 Alt=0
====>
                                                                =====
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 36. SET TABLINE Subcommand — "Before" and "After"

```
 TAILOR   SCRIPT   A1  V 132  Trunc=132 Size=28 Line=9 Col=1 Alt=0
====> SET MSGLINE ON 3 15 OVERLAY
                                                                =====
T  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 * * *                                         =====
```

```
 TAILOR   SCRIPT   A1  V 132  Trunc=132 Size=28 Line=9 Col=1 Alt=0
====> QUERY COLOR *
                                                                =====
T  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 37 (Part 1 of 2). SET MSGLINE on Multiple Lines with Overlay

```
 TAILOR   SCRIPT   A1  V 132  Trunc=132 Size=28 Line=9 Col=1 Alt=0
====>
COLOR    ARROW    DEFAULT   NONE      HIGH   PS0
COLOR    CMDLINE  DEFAULT   NONE      NOHIGH PS0
COLOR    CURLINE  DEFAULT   NONE      HIGH   PS0
COLOR    FILEAREA DEFAULT   NONE      NOHIGH PS0
COLOR    IDLINE   DEFAULT   NONE      HIGH   PS0
COLOR    MSGLINE  RED       NONE      HIGH   PS0
COLOR    PENDING  DEFAULT   NONE      HIGH   PS0
COLOR    PREFIX   DEFAULT   NONE      NOHIGH PS0
COLOR    SCALE    DEFAULT   NONE      HIGH   PS0
COLOR    SHADOW   DEFAULT   NONE      NOHIGH PS0
COLOR    STATAREA DEFAULT   NONE      HIGH   PS0
COLOR    TABLINE  DEFAULT   NONE      HIGH   PS0
COLOR    TOFEOF   DEFAULT   NONE      NOHIGH PS0
                                                                =====
THE CANARY                                                      =====
                                                                =====
THE SONG OF CANARIES                                            =====
NEVER VARIES.                                                   =====
AND WHEN THEY'RE MOULTING                                       =====
THEY'RE PRETTY REVOLTING.                                       =====
                                                                =====
THE GIRAFFE
```

Figure 37 (Part 2 of 2). SET MSGLINE on Multiple Lines with Overlay

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
- Expand the prefix 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, explains a profile macro, and explains how to write prefix macros. You should be familiar with the Restructured Extended Executor (REXX) language, which is described in the *VM/SP System Product Interpreter User's Guide* and the *VM/SP System Product Interpreter Reference* before you read this chapter.

What Is an XEDIT Macro?

An XEDIT macro is a REXX file that is invoked from the XEDIT environment.

(A macro may also be written using the EXEC 2 language; see the *VM/SP EXEC 2 Reference*, SC24-5219, for more information. All examples in this chapter are based on the REXX language.)

You execute a macro the same way you execute XEDIT subcommands; type the macro name on the command line (or the prefix area) and press the ENTER key. A macro may be executed by entering only its name (or synonym), or its execution may also depend on arguments you enter when the macro is invoked.

A macro file can contain:

- XEDIT subcommands
- REXX instructions
- 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 file name, file type, and file mode. The file identifier for a macro file must follow certain rules:

- For macros entered from the command line, the file name is a string of one to eight alphameric characters. This name is used to invoke the macro. For example, if the file name is SEND, entering "SEND" during an editing session causes the macro to be executed. (For information on the search order and

handling file names that contain numbers, see "Avoiding Name Conflicts" later in this chapter.)

Prefix macro file names may be one to eight characters, but they may not contain numbers. (Because the prefix area is only five positions long, you can define a synonym for a prefix macro file name that is longer than five characters. For more information on defining synonyms for prefix macros, see "Writing Prefix Macros," later in this chapter, and the SET PREFIX subcommand description in the *VM/SP System Product Editor Command and Macro Reference*.)

- The file type must be XEDIT.
- The file mode can specify any of your accessed disks or directories, for example, A1.

Using XEDIT Subcommands in a Macro

A macro can contain any XEDIT subcommand, with the following exceptions: prefix macros cannot contain READ, QUIT, FILE, SET RANGE, SORT, and LPREFIX. However, some subcommands perform functions that are meaningful only in the context of a macro, for example, one that passes information to the System Product Interpreter.

The following list summarizes these subcommands; some are then discussed according to function. For detailed information on all of these subcommands, see the *VM/SP System Product Editor Command and Macro Reference*.

CMS	RESTORE
CMSG	SET CTLCHAR
COMMAND	SET COLOR
CP	SET DISPLAY
CURSOR	SET MSGLINE
EMSG	SET MSGMODE
EXTRACT	SET PENDING
MACRO	SET RESERVED
MSG	SET SCOPE
PRESERVE	SET SELECT
READ	STACK

Communicating between the Editor and the Interpreter

The following subcommands are discussed in this section:

READ
EXTRACT.

Both READ and EXTRACT can supply a macro with information.

The READ subcommand is used to find out what the user has entered on the screen. It places fields that have been changed on the screen *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 REXX PULL instruction is used to take information out of the console stack and assign it to program variables, which can then be examined by the macro.

The EXTRACT subcommand can supply a macro with information about internal XEDIT variables or about file data. The information is returned in one or more variables, which can then be examined or used by the macro.

The following sections provide examples of using READ and EXTRACT.

READ Subcommand

When a READ subcommand is issued from a macro, the editor displays "Macro-read" in the status area of the user's screen and waits for the user to enter data and/or press a key. (The file image remains on the screen.) After a key is pressed, the data is placed in the console stack.

Operands of the READ subcommand can be used to specify how much information is placed in the console stack. The READ subcommand can be used to place either the command line or all changed lines in the console stack. In addition, you can request that a tag identifying the origin of the line(s) be inserted at the beginning of each line stacked.

Note: If the console stack already contains a line when a READ subcommand is issued, the READ is a no-op (no operation takes place).

A subsequent REXX PULL instruction assigns the data to program variable(s), and the macro continues executing.

The READ subcommand has the following format:

```
READ        Cmdline   Tag | Notag
            All  Number  Tag | Notag
            Nochange  Number  Tag | Notag
```

Where:

Cmdline
only the command line is stacked.

All
anything changed on the screen is stacked.

Nochange
same as ALL, but the copy of the file in storage is not updated.

Number
changed file lines are prefixed by their line numbers.

Tag
tags that identify the origin of changed lines precede lines placed in the stack.

Notag
no tags are stacked.

Normally, a macro displays a message requesting that you enter data on the command line before it issues a READ.

For example:

MSG ENTER FILENAME FILETYPE FILEMODE
("ENTER FILENAME FILETYPE FILEMODE" is displayed.)

READ CMDLINE
(User enters MYFILE SCRIPT A in the command line and READ puts it in the console stack.)

PULL FN FT FM
(Takes the file ID out of the stack and assigns MYFILE, SCRIPT, and A to FN, FT, and FM, respectively.)

The EXTRACT Subcommand

The EXTRACT subcommand returns information about editing options (options defined by the SET subcommand) as well as other file data that is not explicitly "set." The information is returned as one or more variables in the form "name.n", where "name" is the same as the variable requested and "n" is a subscript that distinguishes the different values returned for each option requested.

For example, if a macro wants information about the case setting it can issue:

```
extract /CASE/
```

This returns information about the contents of the case setting in the following variables:

```
CASE.0    number of variables returned
CASE.1    MIXED|UPPER
CASE.2    RESPECT|IGNORE
```

The macro could use this information as follows:

```
msg "The current case setting is" case.1  case.2
```

Displaying Data on the Editor's Screen

The following subcommands are discussed in this section:

MSG
EMSG
CMSG
SET MSGMODE
SET RESERVED
SET CTLCHAR
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.

Note: REXX also provides an instruction, SAY, that displays one line of data at the terminal. However, the SAY instruction 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 SAY in a macro.

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 redisplay input that the user has entered incorrectly, so that it can be corrected and reentered.

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 displayed in various ways for emphasis. For example, depending on the features supported by your terminal, the line can be displayed highlighted, using a programmed symbol set, in various colors, or with extended highlighting features (blinking, reverse video, or underlining).

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.

Another example of SET RESERVED is shown with SET CTLCHAR, discussed below.

SET CTLCHAR Subcommand

The SET CTLCHAR subcommand is used to specify attributes for fields within a reserved line. Depending on the features supported by your terminal, these fields may be displayed highlighted, protected, invisible, in various colors, using different programmed symbol sets, or with extended highlighting features (blinking, reverse video, or underlining).

In the following example, note how SET RESERVED and SET CTLCHAR are used to control exactly how the reserved lines are displayed.

```
/* This XEDIT macro will show examples of using SET CTLCHAR  */
'SET CTLCHAR % ESCAPE'
'SET CTLCHAR + PROTECT    BLUE     REVVIDEO   NOHIGH'
'SET CTLCHAR J NOPROTECT  GREEN    UNDERLINE  NOHIGH'
'SET RESERVED 3  YEL HIGH This is Yellow%+And this is Blue and Reversed.'
'SET RESERVED 5  RED BLINK NOH Red and Blinking%JGreen and Underlined.'
```

CURSOR Subcommand

The CURSOR subcommand can be used to move the cursor to a specified position on the screen, and optionally, to assign a priority to that position. 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. For an example of using the CURSOR subcommand, see "Positioning the Cursor," later in this chapter.

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 example, you might want to preserve a setting so that you can change it for the duration of the macro, and restore it before the macro finishes executing. 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*.

Entering CMS and CP Commands

As you have seen, an XEDIT macro can contain XEDIT subcommands, REXX instructions, 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 REXX instructions, ADDRESS CMS or ADDRESS COMMAND.)

Note: If, from the XEDIT environment, you invoke a CMS exec that then uses the ADDRESS XEDIT instruction to call other CMS execs from the XEDIT environment, your routine may terminate abnormally due to a lack of storage. To avoid this problem when these circumstances arise, use the ADDRESS XEDIT instruction only to invoke XEDIT macros (that is, files with file types of XEDIT), not to invoke CMS execs.

Avoiding Name Conflicts

The following subcommands are discussed in this section:

COMMAND
MACRO
SET MACRO
SET SYNONYM

Use the COMMAND subcommand to cause the editor to execute a specified subcommand without first checking to see if a synonym or macro with the same name exists. This subcommand overrides SET SYNONYM ON or SET MACRO ON (discussed below).

For example:

```
COMMAND PRESERVE
```

executes the PRESERVE subcommand, even if a synonym or macro with the same name exists.

Similarly, 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. (Of course, this cannot be used for prefix macros.)

The MACRO subcommand can also be used to avoid name conflicts, in the following manner. 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.

In addition, SET SYNONYM can be used to specify whether or not the editor looks for synonyms.

Walking through an XEDIT Macro

The following XEDIT macro, (Figure 38 on page 127) 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 file ID is GLOBCHG XEDIT A1. After the listing, each line in the macro is explained. For more information on the System Product Interpreter statements used in the macro, see the publication *VM/SP System Product Interpreter Reference*.

```
00001 /* Do a global change on imbedded Script files                    */
00002 /* Input to this macro is the CHANGE command to be executed on    */
00003 /* the file currently being xedited and on any files it imbeds.   */
00004 parse arg operand                        /* Get passed CHANGE cmd  */
00005 if operand = '' then do                  /* If omitted, then error */
00006   emsg 'EXE545E Missing operand(s)'      /* Give error message     */
00007   parse source . . me .                  /* Get this macros name   */
00008   cmsg me                                /* Put it on command line */
00009   exit                                   /* Leave this macro       */
00010   end                                    /* End of DO group        */
00011 preserve                                 /* Save current status    */
00012 set wrap off                             /* Set wrap off           */
00013 set msgmode on                           /* Set message mode on    */
00014 set case mixed ignore                    /* Set proper case        */
00015 top                                      /* Go to TOP of file      */
00016 find .im                                 /* Find first imbed file  */
00017 if rc ¬= 0 then do                       /* If none found, give msg */
00018   restore                                /* Restore previous status */
00019   emsg 'No IMBED found.'                 /* Give message           */
00020   exit                                   /* Leave this macro       */
00021   end                                    /* End of DO group        */
00022 do while rc=0                            /* Imbed found, process it */
00023   extract '/curline/'                    /* Get current line       */
00024   parse upper var curline.3 . fname .    /* Separate out file name */
00025   address command state fname 'SCRIPT *'/* Does this file exist?  */
00026   if rc ¬= 0 then do                     /* If not, issue message  */
00027     msg 'IMBEDed file' fname 'SCRIPT does not exist, bypassed.'
00028     find .im                             /* Search for next imbed  */
00029     iterate                              /* Cause next loop iterat'n*/
00030     end                                  /* End of DO group   */
00031   xedit fname 'SCRIPT (NOPROFILE'        /* File exists, XEDIT it  */
00032   extract '/fname/ftype/fmode/'          /* Get name, type, mode   */
00033   msg 'Processing file' fname.1 ftype.1 fmode.1 /* Issue message  */
00034   change operand '* *'                   /* Issue CHANGE command   */
00035   file                                   /* Save the file & quit   */
00036   find .im                               /* Find the next imbed    */
00037 end                                      /* End of DO loop   */
00038 restore                                  /* Loop ends, restore     */
00039 msg 'No more .imbeds found, global change completed.' /* Give msg  */
00040 exit                                     /* All done, leave macro  */
```

Figure 38. Sample Macro

Now, let's walk through the macro, a line at a time.

00001-00003 /* Do a global change on imbedded Script files */
System Product Interpreter comment lines. The first line of any REXX macro must be a comment line to tell the Interpreter this is a REXX file.

00004 parse arg operand
Place the passed arguments into the variable called OPERAND.

00005 if operand = '' then do
If no arguments were entered when the macro was invoked, execute the following statements until an END is reached (DO group). (OPERAND was set to a null in line 4.)

00006 emsg 'EXE545E Missing operand(s)'
Display this message.

00007 parse source . . me .
Look at the source string and place the name of this macro into the variable ME.

00008 cmsg me
The macro name (in the variable ME) is displayed on the command line.

00009 exit
Return control to the editor.

00010 end
This statement signals the end of the DO group that began in line 5.

00011 preserve
This subcommand saves the editor settings until a subsequent RESTORE subcommand is issued (line 38).

00012 set wrap off
Wrapping during the target search is turned off. When the end of the master file is reached the macro will end, rather than wrapping around, searching for ".im," and getting caught in a loop.

00013 set msgmode on
Messages will be displayed. By turning the message mode on and off, you can select which messages you want displayed.

00014 set case mixed ignore
In target searches, uppercase and lowercase representations of the same letter will match.

00015 top
Move the line pointer to the top of the master file, which is the file from which the macro was invoked.

00016 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.

00017 if rc ¬= 0 then do
If there is a non-zero return code from the FIND subcommand, no "im." was found, (previous statement), then do the following statements up to the END (another DO group).

00018 restore
Restore the settings of XEDIT variables to the values they had when the PRESERVE subcommand was issued (line 11).

00019 emsg 'No IMBED found.'
Display this message.

00020 exit
Return control to the editor.

00021 end
This statement signals the end of the DO group that was started in line 17.

00022 do while rc=0
Repeat the following statements (up to the END in line 37), as long as the return code (RC) is 0. The initial value for RC is set by the FIND subcommand in line 15; we reach this point only if RC was set to 0, which means an imbedded file was found. The last statement in this loop is also a FIND subcommand, and RC will be reset to the return code for that FIND subcommand just before we return to this point to execute the statements again. When the return code is not 0, this macro will continue with the statement following the END (line 37).

00023 extract '/curline/'
Return information about the current line in macro variables, in the form "curline.n," where the subscript distinguishes among the variables.

00024 parse upper var curline.3 . fname .
CURLINE.3 contains the contents of the current line (as returned by the EXTRACT subcommand above). In this case the current line is the .im statement that was found via a "find .im". This statement takes the second blank delimited word from the variable CURLINE.3 and puts it into the variable FNAME.

00025 address command state fname '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 statement exists. The quotes are needed around the asterisk to avoid confusion with the REXX multiplication operator. Enclosing the word SCRIPT and the asterisk in quotation marks makes it a literal string.

00026 if rc ¬=0 then do
If the return code from the STATE command is not zero, then do the following statements up to the END statement in line 30 (DO group).

00027 msg 'IMBEDed file' fname 'SCRIPT does not exist, bypassed.'
Display this message. REXX will substitute the value of "fname" in the message before it is displayed.

00028 find .im
This locates the next imbed control word in the file.

00029 iterate
This statement tells REXX to go to the END statement and complete the processing for this iteration of the DO loop.

00030 end
This statement signals the end of the DO group that was started in line 26 above.

00031 xedit fname 'SCRIPT (NOPROFILE'
This statement will invoke the editor (XEDIT) for the file specified. REXX will substitute the value of "fname" in this line before passing it to XEDIT.

00032 extract '/fname/ftype/fmode/'
Returns the file name, file type, and file mode in macro variables.

00033 msg 'Processing file' fname.1 ftype.1 fmode.1
Displays the message, with the file identification as returned by EXTRACT.

00034 change operand '* *'
The global change is executed. OPERAND contains the arguments entered when the macro was invoked (see line 4).

00035 file
The changed file is written to disk or directory.

00036 find .im
The editor resumes editing the master file, searching for the next ".im" statement.

00037 end
This statement signals the end of the DO loop that was started in line 22.

00038 restore
Restore the settings of XEDIT variables to the values they had when the PRESERVE subcommand was issued (line 11).

00039 msg 'No more .imbeds found, global change completed.'
Display this message.

00040 exit
Return control to the editor. You can then issue a QUIT subcommand for the master file.

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 file type of an XEDIT profile macro must be "XEDIT". If the file ID 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 file type of a profile macro must be "XEDIT," the file name does not have to be "PROFILE". If your profile macro has a name other than "PROFILE," you must indicate its file name in the PROFILE option of the XEDIT command.

For example, if the file ID 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 System Product Interpreter 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. (System Product Interpreter 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 file type.

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 39 on page 132.

```
00001 /* Sample XEDIT profile                                    */
00002 parse arg fn ft '(' options       /* put arguments into variables */
00003 if ft= '' then ft= 'SCRIPT'       /* if no file type, use SCRIPT  */
00004 load fn ft '(' options           /* issue LOAD statement        */
00005 set tabline on 22                /* put tab line on line 22     */
00006 set scale on 22                  /* put scale on line 22        */
00007 set fullread on                  /* full-screen read on         */
00008 set nulls on                     /* end of line nulls on        */
00009 set number on                    /* line numbers to be used     */
00010 set pf10 save                    /* PF10 to SAVE                */
00011 set synonym fiel 4 file          /* when my fingers don't work  */
00012 exit                            /* exit this macro             */
```

Figure 39. A PROFILE XEDIT Macro

00001 /* Sample XEDIT profile */
Identifies the macro as a System Product Interpreter file.

00002 parse arg fn ft '(' options
Puts argument into variables.

00003 if ft= '' then ft= 'SCRIPT'
If there is no file type assigned, this will assign a file type of SCRIPT.

00004 load fn ft '(' options
Loads the file.

00005 set tabline on 22
Sets the tabline on line 22 of the screen.

00006 set scale on 22
Superimposes the scale on line 22 of the screen.

00007 set fullread on
Sets the full-screen read on to allow recognition by XEDIT of 3270 null characters in the middle of the screen lines.

00008 set nulls on
Sets NULLS ON to replace all trailing blanks with nulls.

00009 set number on
Sets NUMBER ON to assign a line number to each line in the file.

00010 set pf10 save
Sets PF10 to save the file.

00011 set synonym fiel 4 file
Sets a synonym "fiel" for the subcommand "file".

00012 exit
Exit the macro.

Writing Prefix Macros

You can write prefix macros for a variety of purposes, from performing a function from the prefix area that is normally accomplished by entering a subcommand on the command line, to creating an entirely new function.

You must be familiar with the REXX language before reading this section. More information on REXX can be found in the publications cited at the beginning of this chapter.

Creating a Sample Prefix Macro

The U prefix macro gives the user the ability to translate one or more lines in a file to uppercase, which normally is accomplished by issuing the UPPERCAS subcommand in the command line. When U is entered in the prefix area of a line, that line is translated to uppercase. A number may be specified before or after the U to translate more than one line; for example, 3U= = = or =U5= =.

The file is created with the XEDIT command:

```
XEDIT U XEDIT
```

The U prefix macro looks like this:

```
00001 /* This macro translates a line(s) to uppercase. */
00002 arg . . pline op .
00003 If op =  '' then op = 1
00004 'COMMAND :'pline 'UPPERCAS' op
00005 Exit 0
```

What Information Is Passed to the Macro?

An argument string is automatically passed to a prefix macro when it is invoked. It can supply a macro with information critical to its execution, like the line number of the prefix area in which the macro was entered.

Line 2 (above) is a REXX statement that parses (splits up) the string, according to the template shown. (The argument string is described in greater detail later in this chapter.) *Pline* represents the line number of the prefix area, and *op* represents the optional operand. These variable names provide the macro with answers to the following questions:

On which line was the macro entered?
How many lines are to be translated to uppercase?

Line 3 determines if an operand was entered. If the operand is null, a default of 1 is assumed.

Line 4 makes the line in which the prefix macro was entered (pline) the new current line and then issues the UPPERCAS subcommand, with the operand.

Current Line Positioning

Note that in line 4, *:pline* is an absolute line number target. It is used to make the prefix line (pline) current for the UPPERCAS subcommand, which operates on the current line.

After the pending list is finished executing, the current line is returned automatically to the line that was current when it began execution. Therefore, even though *pline* is made current for the UPPERCAS subcommand, the macro need not restore the current line.

For information on overriding this automatic current line return, see the SET PENDING subcommand in the *VM/SP System Product Editor Command and Macro Reference*.

Creating a Second Prefix Macro

Let's create another prefix macro called L, which gives the user the ability to translate one or more lines in a file to lowercase, which normally is accomplished by issuing the LOWERCAS subcommand in the command line. This macro is similar in function to the U macro described above; however, we will give the user the additional ability of specifying a block of lines to be translated, by entering LL on both the first and last lines of the block.

This macro is presented in segments, to illustrate various concepts. The entire macro is shown at the end of this chapter.

Examining the Source String

You have already seen that an argument string is passed to a prefix macro when it is invoked. A source string is also passed.

```
00007 parse source . . . . . name .
00008 arg pref func pline op extra
```

Line 7 parses the source string according to the template shown. In this example, we're using the source string to get the name of the prefix macro as the user entered it (without operands). Later, you will see how the macro uses *name* to determine if it was invoked in its simple form (L) or block form (LL).

The source string is described in detail in the *VM/SP System Product Interpreter Reference*, under "Parse Source."

In line 8, the argument string is parsed. For now, note that *pline* is the line number of the prefix area, and *op* is the optional operand.

The rest of the argument string is described later in this chapter, under "Examining the Argument String."

In this example, if *L8* were entered in the prefix area of line 3 of a file, *name* would be *L*, *pline* would be *3*, and *op* would be *8*.

Using the Information That Is Passed

The following part of the macro shows how some of the information derived from the strings is used.

```
00007 parse source . . . . . name .
00008 arg pref func pline op extra
      .
      .
      .
00019     when length(name)=1 then do
00020         If op = '' then op = 1
00021         If datatype(op,'W') then,
00022             'COMMAND :'pline 'LOWERCAS' op
00023         else call error "Invalid operand :" op
00024         end
```

In lines 19 through 24, you can see that the source and argument strings supply the answers to these questions:

What name was used to invoke the macro?
On which line was it entered?
How many lines are to be changed to lowercase?

Using the variable names assigned in the templates, lines 19 through 24 perform the following functions:

1. See if the macro was entered in its simple form (L). When the length of *name* is one, the macro was entered in its simple form.
2. If no operand was entered, assign a default of 1 or determine if the operand (if any) is a valid whole number (lines 20 and 21). Otherwise, go to an error routine (line 23).
3. Make the line in which the prefix macro was entered (pline) current and issue the LOWERCAS subcommand, with the operand (line 22).

Handling Blocks

A block is a group of consecutive lines. Several XEDIT prefix subcommands and macros (for example, D and >) allow you to specify blocks by doubling the name and entering it on both the first and last lines of the block (for example, DD entered on the first and last lines of a block deletes the entire block of lines). Let's expand the L prefix macro to accept blocks (specified by entering LL on the first and last lines of the block).

This section explains the following:

- How to assign a synonym for a prefix macro

- How to examine the *pending list* of prefix subcommands and macros for a matching block entry
- How to display a pending notice in the status area of the screen.

```
00018 select
00019     when length(name)=1 then do
00020         If op = '' then op = 1
00021         If datatype(op,'W') then,
00022             'COMMAND :'pline 'LOWERCAS' op
00023         else call error "Invalid operand :" op
00024         end
00025
00026     when length(name)=2 then do
00027         If op ¬= '' then call error,
00028             'Invalid operand :' op
00029         'COMMAND EXTRACT /PENDING BLOCK' name ':0 :'pline '/'
00030         if pending.0¬=0 then do
00031            'COMMAND :'pending.1 'SET PENDING OFF'
00032            'COMMAND :'pending.1 'LOWERCAS :'pline+1
00033            end
00034         else 'COMMAND :'pline 'COMMAND SET PENDING BLOCK' name
00035         End
```

Assigning a Synonym for a Prefix Macro

The user must issue the following subcommand in order to be able to specify the block form of the L macro. You can enter this subcommand in the PROFILE XEDIT file:

```
SET PREFIX SYNONYM LL L
```

Now, the user can invoke the L prefix macro by entering either L (with an optional numeric operand) or LL. In line 19, the macro checks for its simple form (when the length of *name* is 1). In line 26, the macro checks for its block form (when the length of *name* is 2).

Synonyms can also be assigned for other reasons. For example:

- A prefix macro file name can be up to eight alphabetic characters long, but the prefix area is only five positions long. You can use SET PREFIX SYNONYM to assign a synonym that is up to five characters long.
- The synonym can be a special character that is not permitted as part of a CMS file name. For example, the file name for the XEDIT prefix macro > is PRFSHIFT.
- A macro can perform different functions, depending on how it is entered. Different synonyms can signify different functions to the macro. For example, the XEDIT prefix macro PRFSHIFT shifts the screen right if > is entered and left if < is entered. The synonyms assigned to this macro are:

  SET PREFIX SYNONYM > PRFSHIFT
  SET PREFIX SYNONYM < PRFSHIFT
  SET PREFIX SYNONYM > > PRFSHIFT
  SET PREFIX SYNONYM < < PRFSHIFT

- Prefix macros can also use the names of prefix subcommands such as F (following) or P (preceding). To use a prefix subcommand in a prefix macro, you should either define a synonym (see SET PREFIX in the *VM/SP System Product Editor Command and Macro Reference*) or override the prefix subcommand by using SET MACRO ON.

To determine what prefix macro synonyms are in effect, use the QUERY PREFIX SYNONYM subcommand, which is described in detail in the *VM/SP System Product Editor Command and Macro Reference*.

Using the "Pending List"

You have seen that the source and argument strings are two sources of information upon which a prefix macro can base decisions. Another is the "pending list."

The "pending list" is a list of prefix subcommands and macros that have not yet been executed. Every time the editor reads the screen, the pending list is updated (automatically) with any new prefix subcommands and macros that have been entered, each of which causes an entry to be added to the list. Each entry is associated with a specific line in the file.

The pending list is executed when it is changed. If a prefix macro returns a non-zero return code, execution of the pending list stops and all entries not executed remain pending, until the user presses the ENTER (or PF/or PA) key.

An entry is deleted from the pending list when it is executed or overtyped on the user's screen with a new prefix subcommand, prefix macro, or blanks. For example, when the L prefix macro is invoked, it is removed from the pending list.

A prefix macro can control its execution and display or remove the pending notice from the status area of the screen by examining information in the pending list (EXTRACT/QUERY PENDING) and by adding or deleting entries in it (SET PENDING). Refer to the *VM/SP System Product Editor Command and Macro Reference* for detailed information on these subcommands.

The pending notice is displayed in the status area as follows:

```
'value' pending...
```

where "value" is the name of the prefix subcommand or macro that was entered in the prefix area, as derived from the source string (see line 34). (If multiple prefix subcommands or macros are pending, the first one, starting from the top of file, is displayed in the pending notice.)

In our example, suppose that the user entered the block form (LL), which is determined by line 26. First, the macro needs to know if another LL has been entered, that is, if the pending list contains a matching block entry.

To determine this, the macro examines the pending list by issuing the EXTRACT subcommand shown in line 29. This subcommand searches the pending list for a matching block entry, which must be located in the file within the range specified by the targets, that is, between the top of file (:0) and the prefix line (:pline), inclusive. If no matching entry is found, the screen is placed in a pending status (line 34).

If a second LL was entered, the pending status of the screen will not be seen because the macro is automatically invoked again as the pending list is executed. This time, the EXTRACT subcommand (line 29) finds the matching block entry, the pending notice is removed (line 31) and the LOWERCAS subcommand is executed for the block of lines (line 32).

Examining the Argument String

The argument string is as follows:

```
PREFIX SET|SHADOW|CLEAR pline [op1[op2[op3]]]
```

Where:

PREFIX
indicates that this is a prefix call.

SET
indicates that the prefix macro was entered on some line in the file displayed.

SHADOW
indicates that a prefix macro was entered on a shadow line (see SET SHADOW in the *VM/SP System Product Editor Command and Macro Reference*).

CLEAR
indicates that a new prefix subcommand or macro or new blank area replaces a previously pending prefix subcommand or macro on the same line, or the RESET subcommand was entered. In this case, this macro is invoked with "PREFIX CLEAR pline".

pline
is the line number on which the prefix macro was entered.

op1 op2 op3
are the optional operands of the macro, entered either to its left or right (for example, 5M or M5). (Operands are recognized according to the rules explained in "Section 4: Prefix Subcommands and Macros" in the *VM/SP System Product Editor Command and Macro Reference*.)

Let's see how this macro uses the argument string for validity checking.

```
00008 arg pref func pline op extra
00009 If pref ¬= 'PREFIX' then call error1,
00010   'This macro must be invoked from the PREFIX area.'
00011 If func = 'CLEAR' then exit
00012 If func = 'SHADOW' then call error1,
00013   'Invalid on shadow line.'
00014 If func ¬= 'SET' then call error1,
00015   'This macro must be invoked from the PREFIX area.'
00016 If extra ¬= '' then call error,
00017   'Extraneous parameter:' extra
          .
          .
          .
00042 /* error routines */
00043 error: 'COMMAND :'pline 'SET PENDING ERROR' name||op
00044 error1: parse arg t
00045        'COMMAND EMSG' t
00046        Exit
```

Lines 9 through 17 verify that the macro is a prefix call and was entered on a valid prefix line, that is, not on a shadow line. Lines 42 through 45 are the associated error routines.

Line 43 is a form of the SET PENDING subcommand used to notify the user the macro was entered incorrectly. In this case, if an extra operand was entered (determined in line 16), the incorrect macro is displayed highlighted in the prefix area, prefixed by a question mark. For example, if the user entered L3 4, the prefix area displays ?L3 and the user gets the message 'Extraneous parameter: 4'.

SET PENDING ERROR does not cause a pending notice to be displayed. When the user presses the ENTER key again, the prefix area is reset. This prevents subsequent attempts to execute an incorrectly-entered macro.

Positioning the Cursor

The cursor is positioned in the line in which the prefix macro was entered by using the following subcommand:

```
00039 'COMMAND CURSOR FILE' pline 'PRIORITY 30'
```

By using the CURSOR subcommand, user-written prefix macros can specify a priority that is associated with cursor positioning. The cursor is positioned at the location specified that has the highest priority when all pending prefix subcommands and any macros are executed.

For more information on the CURSOR subcommand and various priorities associated with prefix subcommands and macros, see the *VM/SP System Product Editor Command and Macro Reference*, the CURSOR subcommand and "Section 4: Prefix Subcommands and Macros."

The rest of this chapter presents additional information which may be useful in writing prefix macros or tells you where the information can be found.

Decoding the Prefix Area

See the *VM/SP System Product Editor Command and Macro Reference*, "Section 4: Prefix Subcommands and Macros" for a description of how the editor interprets what is entered in the prefix area.

Using the XEDIT Subcommand

A prefix macro can issue the XEDIT subcommand to edit a different file in the ring. However, when the macro finishes executing, control automatically returns to the file from which it was invoked.

Additional Examples

For additional examples of prefix macros, you can examine the IBM-supplied prefix macros, which are as follows:

Macro synonym(s)	File Identifier
X, XX	PREFIXX XEDIT
S	PRFSHOW XEDIT
<, >, > >, < <	PRFSHIFT XEDIT
. . . . .	SI XEDIT

The L Prefix Macro

```
00001 /* Use this macro to translate a line or lines in a file */
00002 /* to lowercase.                                          */
00003 /* You may specify nL, Ln, L-n or L to lowercase a line. */
00004 /* If you add the following prefix synonym to your       */
00005 /* profile, you may also use LL for specifying blocks:   */
00006 /*         SET PREFIX SYNONYM LL L                       */
00007 parse source . . . . . name .
00008 arg pref func pline op extra
00009 If pref ¬= 'PREFIX' then call error1,
00010   'This macro must be invoked from the PREFIX area.'
00011 If func = 'CLEAR' then exit
00012 If func = 'SHADOW' then call error1,
00013   'Invalid on shadow line.'
00014 If func ¬= 'SET' then call error1,
00015   'This macro must be invoked from the PREFIX area.'
00016 If extra ¬= '' then call error,
00017   'Extraneous parameter:' extra
00018 select
00019     when length(name)=1 then do
00020         If op = '' then op = 1
00021         If datatype(op,'W') then,
00022             'COMMAND :'pline 'LOWERCAS' op
00023         else call error "Invalid operand :" op
00024         end
00025
00026     when length(name)=2 then do
00027         If op ¬= '' then call error,
00028             'Invalid operand :' op
00029         'COMMAND EXTRACT /PENDING BLOCK' name ':0 :'pline '/'
00030         if pending.0¬=0 then do
00031            'COMMAND :'pending.1 'SET PENDING OFF'
00032            'COMMAND :' pending.1 'LOWERCAS :'pline+1
00033            end
00034         else 'COMMAND :'pline 'COMMAND SET PENDING BLOCK' name
00035         End
00036
00037     Otherwise call error "Invalid macro synonym."
00038 End
00039 'COMMAND CURSOR FILE' pline 'PRIORITY 30'
00040 Exit
00041
00042 /* error routines */
00043 error: 'COMMAND :'pline 'SET PENDING ERROR' name||op
00044 error1: parse arg t
00045        'COMMAND EMSG' t
00046        Exit
```

Figure 40. Sample Prefix Macro

Appendix A. Summary of XEDIT Subcommands and Macros

These subcommands are described in detail in the publication *VM/SP System Product Editor Command and Macro Reference.*

Subcommand	Purpose
Add	Adds n line(s) after current line.
ALL	Selects a collection of lines for display/editing.
ALter	Changes a single character to another (character or hex).
BAckward	Scrolls backward n screen displays.
Bottom	Goes to last line of file.
CANCEL	Terminates the editing session for all files.
CAppend	Adds text to end of current line.
CDelete	Deletes characters, starting at column pointer.
CFirst	Moves column pointer to beginning of line (zone).
Change	Changes one string to another.
CInsert	Inserts text starting at the column pointer of the current line.
CLAst	Moves the column pointer to the end of the line (zone).
CLocate	Locates a string; moves the column pointer and the line pointer.
CMS	Passes a command to CMS, or enters CMS subset mode.
CMSG	Displays message in command line of user's screen.
COMMAND	Executes a subcommand without checking for synonym or macro.
COMPress	Prepares line(s) for realignment by replacing blanks with tab characters.
COpy	Copies line(s) at specified location.
COUnt	Displays the number of times a string appears.
COVerlay	Replaces characters, starting at column pointer.
CP	Passes command to VM/SP control program.
CReplace	Replaces characters, starting at the column pointer.
CURsor	Moves the cursor to specified position on the screen, and optionally assigns a priority for this position.
DELete	Deletes line(s) beginning with the current line.
Down	Moves line pointer n lines toward end of file (same as NEXT).
DUPlicat	Duplicates line(s).
EMSG	Displays a message and sounds the alarm.

Subcommand	Purpose
EXPand	Repositions data according to new tab settings.
EXTract	Returns information about internal XEDIT variables and file data.
FILE	Writes file to disk or directory.
Find	Searches for line that starts with specified text.
FINDUp	Searches for a line that starts with specified text; searches in a backward direction.
FOrward	Scrolls forward n screen displays.
GET	Inserts lines from another file.
Help	Requests online display of XEDIT subcommands and macros; invokes the CMS HELP facility.
HEXType	Displays line(s) in hexadecimal and EBCDIC.
Input	Inserts a single line, or enters input mode.
Join	Combines two or more lines into one line.
LEft	Views data to the left of column one.
LOAD	Reads file into storage; use in profile macro only.
Locate	Moves line pointer to specified target.
LOWercas	Changes uppercase letters to lowercase.
LPrefix	Simulates writing in the prefix area of the current line. Used on typewriter terminals.
MACRO	Executes macro without checking for subcommand or synonym.
MErge	Combines two sets of lines.
MODify	Displays a subcommand and its current values in the command line, so it can be overtyped and reentered.
MOve	Moves line(s) to another place in the file.
MSG	Displays message in message line.
Next	Moves line pointer n lines toward end of file (same as DOWN).
NFind	Searches forward for first line that does not start with the specified text.
NFINDUp	Searches backward for first line that does not start with the specified text.
Overlay	Replaces characters in current line.
PARSE	Scans a line of a macro to check the format of its operands.
POWerinp	Enters input mode for continuous typing.
PREServe	Saves settings of XEDIT variables until RESTORE is entered.
PURge	Removes macro from virtual storage.

Subcommand	Purpose
PUT	Inserts lines into another file (new or existing), or into a buffer (to be retrieved by GET from another file).
PUTD	Same as PUT, but deletes original lines.
Query	Displays the current value of editing options.
QUIT	Ends an editing session without saving changes.
READ	Places information from the terminal in the console stack.
RECover	Replaces removed lines.
REFRESH	Issued from a macro, it updates the display on the screen.
RENum	Renumbers VSBASIC or FREEFORT files.
REPEat	Advances line pointer and reexecutes last subcommand.
Replace	Replaces current line, or deletes current line and enters input mode.
RESet	Removes prefix subcommands or macros when screen is in "pending" status.
RESTore	Restores settings of XEDIT variables to values they had when PRESERVE was issued.
RGTLEFT	Shifts display to the right or left; reissue to shift back to original display.
RIght	Views data to the right of the last (right-most) column.
SAVE	Writes file to disk or directory and remains in edit mode.
SCHANGE	Locates string and makes a selective change, using PF keys.
SET ALT	Changes the number of alterations that have been made to the file since the last AUTOSAVE and/or since the last SAVE.
SET APL	Informs the editor and CMS if APL keys are used.
SET ARBchar	Defines an arbitrary character to be used in a target definition.
SET AUtosave	Automatically issues a SAVE subcommand at specified intervals.
SET BRKkey	Specifies whether or not CP should break in when the "BRKKEY " (defined by CP TERMINAL BRKKEY) is pressed.
SET CASE	Uppercase or lowercase control; specifies if case is significant in target searches.
SET CMDline	Moves the position of the command line.
SET COLOR	Associates specific colors and attributes with various fields on the XEDIT screen.
SET COLPtr	Specifies if column pointer is displayed (typewriter terminals only).

Subcommand	Purpose
SET CTLchar	Defines a control character(s), which associate parts of a reserved line with highlighting, protection, visibility, various colors, extended highlighting, and Programmed Symbol Sets.
SET CURLine	Defines the position of the current line on the screen.
SET DISPlay	Indicates which selection levels of lines will be displayed on the screen.
SET ENTer	Defines a meaning for the ENTER key.
SET ESCape	Defines a character that allows you to enter a subcommand while in input mode (typewriter terminals only).
SET ETARBCH	Defines an arbitrary character within a file containing Double-Byte Character Set (DBCS) characters to be used in target definitions.
SET ETMODE	Informs the editor that there are Double-Byte Character Set strings in the file.
SET FILler	Defines a character that is used when a line is expanded.
SET FMode	Changes the file mode of the current file.
SET FName	Changes the file name of the current file.
SET FType	Changes the file type of the current file.
SET FULLread	Determines whether or not the editor and CMS recognize null characters in the middle of screen lines.
SET HEX	Allows string operands and targets to be specified in hexadecimal.
SET IMage	Controls how tabs and backspaces are handled when a line is entered.
SET IMPcmscp	Controls whether subcommands not recognized by the editor are transmitted to CMS and CP.
SET LASTLorc	Defines the contents of the last locate or change buffer.
SET LINENd	Defines a line end character.
SET LRecl	Defines a new logical record length.
SET MACRO	Controls the order in which the editor searches for subcommands and macros.
SET MASK	Defines a new mask, which is the contents of added lines and the input zone.
SET MSGLine	Defines position of message line and the number of lines a message may expand to.
SET MSGMode	Controls the message display.
SET NONDisp	Defines a character to XEDIT and CMS that is used in place of non-displayable characters.
SET NULls	Specifies whether trailing blanks are replaced with nulls to allow character insertion.

Subcommand	Purpose
SET NUMber	Specifies whether file line numbers are displayed in the prefix area.
SET PAn	Defines a meaning for a PA key.
SET PACK	Specifies if the file is to be written to disk or directory in packed format.
SET PENDing	Controls the execution of a prefix macro and the status of the screen while the macro is being executed.
SET PFn	Defines a meaning for a PF key.
SET Point	Defines a symbolic name for the current line.
SET PREfix	Controls the display of the prefix area or defines a synonym for a prefix subcommand.
SET RANge	Defines a new "top" and "bottom" for the file.
SET RECFm	Defines the record format.
SET REMote	Controls the way data transmission is handled in XEDIT and CMS.
SET RESERved	Reserves a line, which cannot be used by the editor.
SET SCALe	Controls the display of the scale line.
SET SCOPE	Specifies whether the editor operates on the entire file or on only those lines displayed.
SET SCReen	Divides the screen into logical screens, for multiple views of the same or of different files.
SET SELect	Assigns a selection level to a line or group of lines in a file.
SET SERial	Controls file serialization.
SET SHADow	Specifies whether the file is to be displayed with or without shadow lines indicating where lines have been excluded from the display.
SET SIDcode	Specifies a character string that is to be inserted into every line of an update file.
SET SPAN	Allows a string target to span a number of lines.
SET SPILL	Controls whether or not truncation will occur for certain subcommands.
SET STAY	Specifies for certain subcommands whether the line pointer moves when searching for a string.
SET STReam	Specifies whether the editor searches only the current line or the whole file for a column-target.
SET SYNonym	Specifies whether the editor looks for synonyms; assigns a synonym.
SET TABLine	Controls the display of the tab line.
SET TABS	Defines the logical tab stops.
SET TERMinal	Specifies whether a terminal is used in line mode or full-screen mode.

Subcommand	Purpose
SET TEXT	Informs the editor and CMS if TEXT keys are used.
SET TOFEOF	Controls the display of TOF/EOF lines.
SET TRANSLat	Controls user-defined uppercase translation.
SET TRunc	Defines the truncation column.
SET VARblank	Specifies whether the number of blanks between two words is significant in a target search.
SET Verify	Controls whether lines changed by subcommands are displayed; defines the columns displayed and whether displayed in EBCDIC, hexadecimal or both.
SET WRap	Controls whether the editor wraps around the file if EOF (or TOF for backwards searches) is reached during a search.
SET Zone	Defines new limits within each line for target searches.
SET =	Inserts string into the equal buffer.
SHift	Moves data right or left (data loss possible).
SI	Continuously adds lines and positions cursor for indented text.
SORT	Sorts all or part of a file, in ascending or descending order.
SOS	Specifies functions for screen operation simulation.
SPlit	Splits a line into two or more lines.
SPLTJOIN	Splits a line or joins two lines at the cursor.
STAck	Places line(s) from the file into the console stack.
STATus	Displays SET subcommand current settings; creates a macro that contains these settings.
TOP	Moves line pointer to null TOP OF FILE line.
TRAnsfer	Places editing variable(s) in the console stack, for use by a macro.
Type	Displays lines.
Up	Moves line pointer n lines toward top of file.
UPPercas	Translates all lowercase characters to uppercase.
Xedit	Edits multiple files.
&	Use before a subcommand to redisplay the command.
=	Reexecutes the last subcommand, macro, or CP/CMS command.
?	Displays the last subcommand, macro, or CP/CMS command executed.

Prefix	Subcommands
A	Adds line(s).
C	Copies line(s).
D	Deletes line(s).
E	Extends a line.
F	Moves or copies following this line.
I	Inserts line(s).
M	Moves line(s).
P	Moves or copies preceding this line.
SI	Continuously adds lines and positions cursor for indented text.
"	Duplicates line(s).
/	Makes this line the current line.
SCALE	Displays the scale on this line.
TABL	Displays the tab line on this line.
.xxxx	Assigns symbolic name to this line.
X	Excludes line(s) from display.
S	Shows excluded line(s).
<	Shifts line(s) to the left.
>	Shifts line(s) to the right.

Summary of Changes

Previous editions of this book may be ordered using the pseudo-number found in the *VM/SP Release 6.0 Library Guide and Master Index.*

Summary of Changes
for SC24-5220-04
VM/SP Release 6

This edition reflects minor technical and editorial changes.

Summary of Changes
for SC24-5220-03
VM/SP Release 5

This edition reflects minor technical changes and editorial corrections.

Summary of Changes
for SC24-5220-02
VM/SP Release 4

The XEDIT enhancements described in this document provide new or improved support in the following areas:

- Structured Input

  SI Prefix Macro (to continuously add new lines of indented text)

- Usability

  (all messages issued by the editor are in mixed case).

Glossary of Terms and Abbreviations

A

alphameric. Synonym for *alphanumeric*.

alphanumeric. Pertaining to a character set that contains letters, digits, and usually other characters, such as punctuation marks. Synonymous with *alphameric*.

B

border. A boundary around a window. The user can enter one-letter BORDER commands from the corners of the border. For example, the letter *P* entered from a border corner pops the window. The border corners are indicated by a + (plus) sign.

buffer. An area of storage, temporarily reserved for performing input or output, into which data is read, or from which data is written.

C

CMS. Conversational Monitor System.

CMS EXEC. An EXEC procedure or EDIT macro written in the CMS EXEC language and processed by the CMS EXEC processor. Synonymous with *CMS program*.

CMS EXEC **language**. A general-purpose, high-level programming language, particularly suitable for EXEC procedures and EDIT macros. The CMS EXEC processor executes procedures and macros (programs) written in this language. Contrast with *EXEC 2 language* and *Restructured Extended Executor (REXX) language*.

CMS **program**. Synonym for *CMS EXEC*.

command. A request from a user at a terminal for the execution of a particular CP, CMS, IPCS, GCS, TSAF, or AVS function. A CMS command can also be the name of a CMS file with a file type of EXEC or MODULE. See *subcommand* and *user-written CMS command*.

command abbreviation. A short form of the command name, operand, or option that is not a truncation of the word. For example, MSG instead of MESSAGE, RDR instead of READER. Contrast with *truncation*.

command line. The line at the bottom of display panels that lets a user enter commands or panel selections. It is prefixed by an arrow (= = = >).

control program. A computer program that schedules and supervises the program execution in a computer system. See *Control Program (CP)*.

Control Program (CP). A component of VM/SP that manages the resources of a single computer so multiple computing systems appear to exist. Each virtual machine is the functional equivalent of an IBM System/370.

Conversational Monitor System (CMS). A virtual machine operating system and component of VM/SP that provides general interactive time sharing, problem solving, program development capabilities, and operates only under the control of the VM Control Program (CP).

CP. Control Program.

D

DBCS. Double-byte character set.

delimiter. A character that groups or separates words or values in a line of input. Usually one or more blank characters separate the command name and each operand or option in the command line. In certain cases, a tab, left parenthesis, or backspace character can also act as a delimiter.

disk. A magnetic disk unit in the user's CMS virtual machine configuration. Also called a virtual disk.

display mode. A type of editing at a display terminal in which an entire screen of data is displayed at once and in which the user can access data through commands or by using a cursor. Contrast with *line mode*.

display terminal. A terminal with a component that can display information on a viewing surface such as a CRT or gas panel.

double-byte character set (DBCS). A character set that requires 2 bytes to uniquely define each character. This contrasts with EBCDIC, in which each printed character is represented by 1 byte.

E

edit. A function that makes changes, additions, or deletions to a file on a disk. These changes are interactively made. The edit function also generates information in a file that did not previously exist.

edit mode. The environment in which CMS EDIT

subcommands and System Product Editor (XEDIT) subcommands can be entered by the user to insert, change, delete, or rearrange the contents of a CMS file. Contrast with *input mode*.

EOF. End of file.

EXEC 2 language. A general-purpose, high-level programming language, particularly suitable for EXEC procedures and XEDIT macros. The EXEC 2 processor runs procedures and XEDIT macros (programs) written in this language. Contrast with *CMS EXEC language* and *Restructured Extended Executor (REXX) language*.

F

file access mode. A file mode number that designates whether the file can be used as a read-only or read/write file by a user. See *file mode*.

file ID. A CMS file identifier that consists of a file name, file type, and file mode. The file ID is associated with a particular file when the file is created, defined, or renamed under CMS. See *file name, file type,* and *file mode*.

file mode. A two-character CMS file identifier field comprised of the file mode letter (A through Z) followed by the file mode number (0 through 6). The file mode letter indicates the minidisk or SFS directory on which the file resides. The file mode number indicates the access mode of the file. See *file access mode*.

file name. A one-to-eight character alphanumeric field, comprised of A through Z, 0 through 9, and special characters $ # @ + - (hyphen) : (colon) _ (underscore), that is part of the CMS file identifier and serves to identify the file for the user.

file type. A one-to-eight character alphanumeric field, comprised of A through Z, 0 through 9, and special characters $ # @ + - (hyphen) : (colon) _ (underscore), that is used as a descriptor or as a qualifier of the file name field in the CMS file identifier. See *reserved file types*.

full-screen CMS. When a user enters the command SET FULLSCREEN ON, CMS is in a window and can take advantage of 3270-type architecture and windowing support, and various classes of output are routed to a set of default windows. Also, users can type commands anywhere on the physical screen and scroll through commands and responses previously displayed. See *windowing*.

I

input line. For typewriter terminals, information keyed in by a user between the time the typing element of the terminal comes to rest following a carriage return until another carriage return is typed. For display terminals, the data keyed into the user input area of the screen. See *user input area*.

input mode. In the CMS Editor or System Product Editor (XEDIT), the environment that lets the user key in new lines of data. Contrast with *edit mode*.

invoke. To start a command, procedure, or program.

L

line mode. The mode of operation of a display terminal that is equivalent to using a typewriter-like terminal. Contrast with *display mode*.

line number. A number located at either the beginning or the end of a record (line) that can be used during editing to refer to that line. See *prompting*.

load. In reference to installation and service, to move files from tape to disk, auxiliary storage to main storage, or minidisks to virtual storage within a virtual machine.

logical line. A command or data line that can be separated from one or more additional command or data lines on the same input line by a logical line end symbol.

logical record. A formatted record that consists of a 2-byte logical record length and a data field of variable length.

M

module. A unit of a software product that is discretely and separately identifiable with respect to modifying, compiling, and merging with other units, or with respect to loading and execution. For example, the input to, or output from, a compiler, the assembler, the linkage editor, or an exec routine.

N

null line. A logical line with a length of zero that usually signals the CMS Editor to end input mode and enter edit mode. In VM/SP, a null line for typewriter terminals is a terminal input line consisting of a return character as the first and only information, or a logical line end symbol as the last character in the data line. For display devices, a null line is indicated by the cursor positioned at the beginning of the user input area or the

data in the user input area ending with a logical line end symbol.

O

operand. Information entered with a command name to define the data on which a command processor operates and to control the execution of the command processor.

P

parameter. A variable that is given a constant value for a specified application and that may denote the application.

PF key. Program function key.

physical screen. Synonym for *screen.*

prefix area. The five left-most positions on the System Product Editor's full-screen display, in which prefix subcommands or prefix macros can be entered. See *prefix macros* and *prefix subcommands.*

prefix macros. System Product Editor macros entered in the prefix area of any line on a full-screen display. See *prefix area.*

prefix subcommands. System Product Editor subcommands entered in the prefix area of any line on a full-screen display. See *prefix area.*

PROFILE EXEC. A special EXEC procedure with a file name of PROFILE that a user can create. The procedure is usually executed immediately after CMS is loaded into a virtual machine (also known as IPL CMS).

program function (PF) key. On a terminal, a key that can do various functions selected by the user or determined by an application program.

prompt. A displayed message that describes required input or gives operational information.

prompting. An interactive technique that lets the program guide the user in supplying information to a program. The program types or displays a request, question, message, or number, and the user enters the desired response. The process is repeated until all the necessary information is supplied.

R

receive. Bringing into the specified buffer data sent to the user's virtual machine from another virtual machine or from the user's own virtual machine.

reserved file types. File types recognized by the CMS editors (EDIT and XEDIT) as having specific default attributes that include: record size, tab settings, truncation column, and uppercase or lowercase characters associated with that particular file type. The CMS Editor creates a file according to these attributes.

Restructured Extended Executor (REXX) language. A general-purpose programming language, particularly suitable for EXEC procedures, XEDIT macros, or programs for personal computing. Procedures, XEDIT macros, and programs written in this language can be interpreted by the System Product Interpreter. Contrast with *CMS EXEC language* and *EXEC 2 language.*

REXX language. Restructured Extended Executor language.

ring of files. The arrangement of files in virtual storage when multiple files are being edited by the System Product Editor.

S

scale. A line on the System Product Editor's (XEDIT) full-screen display, used for column reference.

screen. An illuminated display surface; for example, the display surface of a CRT. Synonymous with *physical screen.*

scrolling. (1) Moving a display image vertically or horizontally in order to view data not otherwise visible within the boundaries of the display screen.
(2) Performing a scroll up, scroll down, scroll right, or scroll left operation.

SFS. Shared file system.

shared file system (SFS). A part of CMS that lets users organize their files into groups known as *directories* and to selectively share those files and directories with other users.

source update file. A file containing a single change to a statement in a source file. The file can also include requisite information for applying the change. Synonymous with *update file.*

subcommand. The commands of processors such as EDIT or System Product Editor (XEDIT) that run under CMS.

System Product Editor. The CMS facility, comprising the XEDIT command and XEDIT subcommands and macros, that lets a user create, change, and manipulate CMS files.

System Product Interpreter. The language processor of the VM/SP operating system that processes procedures, XEDIT macros, and programs written in the REXX language.

T

target. One of many ways to identify a line to be searched for by the System Product Editor. A target can be specified as an absolute line number, a relative displacement from the current line, a line name, or a string expression.

terminal. A device, usually equipped with a keyboard and a display, capable of sending and receiving information.

terminal session. The period of time from logon to logoff when a user and the virtual machine can use the facilities of VM/SP or the operating system or both. This also includes any period of time that the virtual machine is running in disconnect mode.

truncation. A valid shortened form of CP, CMS, GCS, IPCS, RSCS, TSAF (Query only) command names, operands, and options that can be keyed in. When the shortened form is used, the number of key strokes is reduced. For example, the ACCESS command has a minimum allowable truncation of two, so AC, ACC, ACCE, ACCES, and ACCESS are all recognized by CMS as the ACCESS command. Contrast with *command abbreviation.*

typewriter terminal. Printer-keyboard devices that produce hardcopy output only, such as: the IBM 2741 Communication Terminal; the IBM 3215 Console Printer-Keyboard; the IBM 3767 Communication Terminal, Model 1 or 2, operating as a 2741. This term also refers to the IBM 3101 Display Terminal operating as a 2741.

U

update file. Synonym for *source update file.*

user. Anyone who requests the services of a computing system.

user input area. On a display device, the lines of the screen where the user is required to key in command or data lines. See *display mode*, *input line*, and *line mode.*

user-written CMS command. Any CMS file created by a user that has a file type of MODULE or EXEC. Such a file can be executed as if it were a CMS command by issuing its file name, followed by any operands or options expected by the program or EXEC procedure.

V

virtual machine (VM). A functional equivalent of a real machine.

Virtual Machine/System Product (VM/SP). An IBM licensed program that manages the resources of a single computer so that multiple computing systems appear to exist. Each virtual machine is the functional equivalent of a *real* machine.

virtual screen. A functional simulation of a physical screen. A virtual screen is a *presentation space* where data is maintained. The user can view pieces of the virtual screen through a window on the physical screen.

virtual storage. Storage space that can be regarded as addressable main storage by the user of a computer system in which virtual addresses are mapped into real addresses. The size of virtual storage is limited by the addressing scheme of the computing system and by the amount of auxiliary storage available, and not by the actual number of main storage locations.

VM. Virtual machine.

VM/SP. Virtual Machine/System Product.

W

window. An area on the physical screen where virtual screen data can be displayed. Windowing lets the user do such functions as defining, positioning, and overlaying windows; scrolling backward and forward through data; and writing data into virtual screens.

windowing. A set of functions that lets the user view and manipulate data in user-defined areas of the physical screen called *windows.* Windowing support lets the user define, position, and overlay windows; scroll backward and forward through data; and write data into virtual screens.

X

XEDIT. See *System Product Editor.*

XEDIT macro. (1) A procedure defined by a frequently used command sequence to do a commonly required editing function. A user creates the macro to save repetitious rekeying of the sequence, and invokes the entire procedure by entering a command (that is, the macro file's file name). The procedure can consist of a long sequence of XEDIT commands and subcommands or both, and CMS and CP commands or both, along with REXX or EXEC 2 control statements to control

processing within the procedure. (2) A CMS file with a file type of *XEDIT*.

XEDIT profile macro. A special XEDIT macro with a file name of PROFILE and a file type of XEDIT that a user can create. It is automatically executed when an XEDIT command (or subcommand) is entered.

Bibliography

Related Publications

Virtual Machine/System Product

System Product Editor Command and Macro Reference, SC24-5221

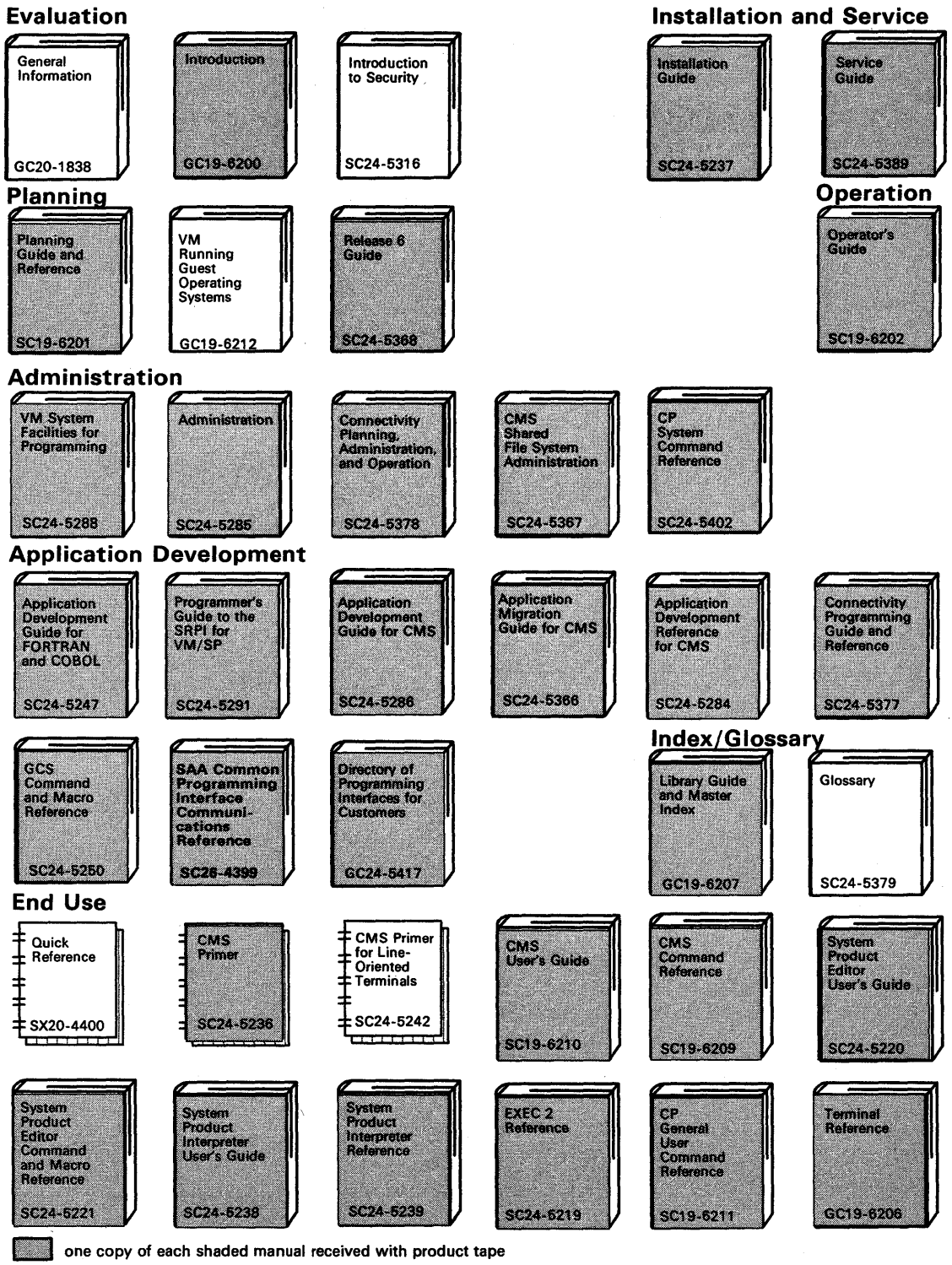
System Product Interpreter User's Guide, SC24-5238

System Product Interpreter Reference, SC24-5239

CMS Primer, SC24-5236

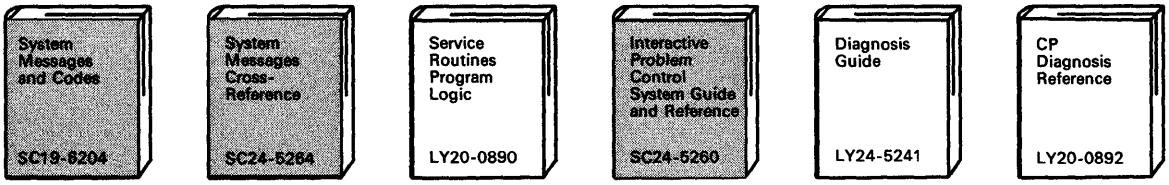
CMS Primer for Line-Oriented Terminals, SC24-5242.

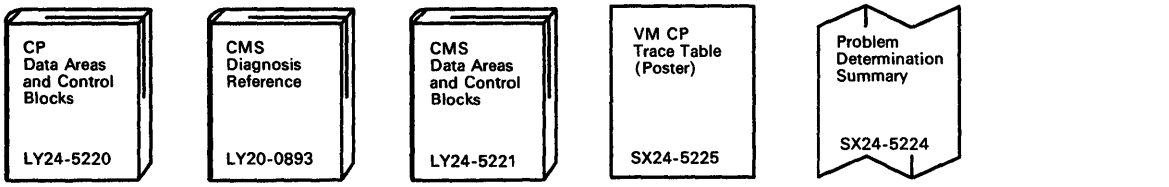
VM/SP RELEASE 6 LIBRARY



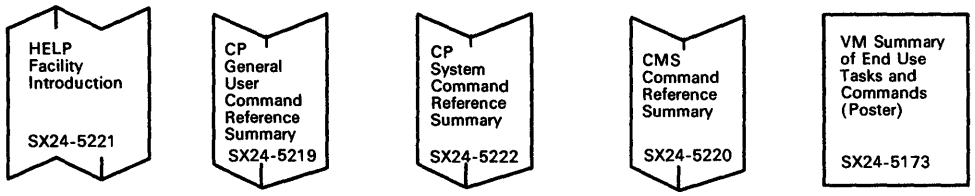
VM/SP RELEASE 6 LIBRARY

Diagnosis

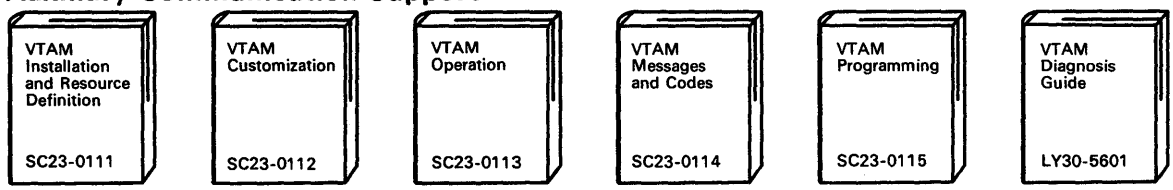


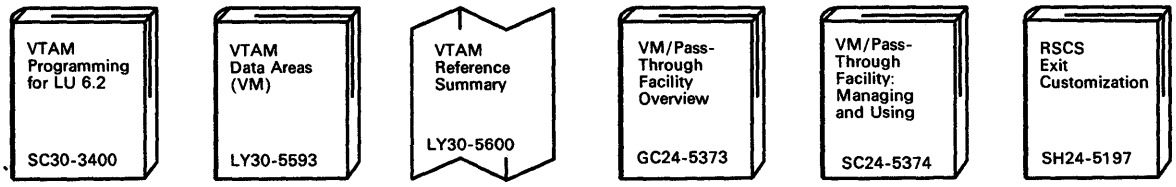


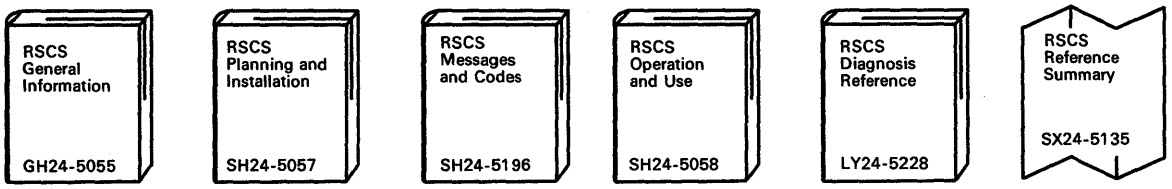
Reference Summaries



Auxiliary Communication Support







Index

D

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Special Characters

Program Number
5664-167

File Number
S370/4300-39

Printed in U.S.A.

VM/SP
System Product Editor User's Guide
Order No. SC24-5220-04

READER'S COMMENT FORM

Is there anything you especially like or dislike about this book? Feel free to comment on specific errors or omissions, accuracy, organization, or completeness of this book.

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you, and all such information will be considered nonconfidential.

Note: Do not use this form to report system problems or to request copies of publications. Instead, contact your IBM representative or the IBM branch office serving you.

Would you like a reply? ___YES ___NO

Please print your name, company name, and address:

_____

_____

_____

_____

IBM Branch Office serving you: _____

Thank you for your cooperation. You can either mail this form directly to us or give this form to an IBM representative who will forward it to us.

SC24-5220-04

Reader's Comment Form

CUT
OR
FOLD
ALONG
LINE

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NY

POSTAGE WILL BE PAID BY ADDRESSEE:

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
DEPARTMENT G60
PO BOX 6
ENDICOTT NY 13760-9987

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

VM/SP
System Product Editor User's Guide
Order No. SC24-5220-04

READER'S COMMENT FORM

Is there anything you especially like or dislike about this book? Feel free to comment on specific errors or omissions, accuracy, organization, or completeness of this book.

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you, and all such information will be considered nonconfidential.

Note: Do not use this form to report system problems or to request copies of publications. Instead, contact your IBM representative or the IBM branch office serving you.

Would you like a reply? ___YES ___NO

Please print your name, company name, and address:

_____

_____

_____

_____

IBM Branch Office serving you: _____

Thank you for your cooperation. You can either mail this form directly to us or give this form to an IBM representative who will forward it to us.

SC24-5220-04

Reader's Comment Form

CUT
OR
FOLD
ALONG
LINE

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NY

POSTAGE WILL BE PAID BY ADDRESSEE:

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
DEPARTMENT G60
PO BOX 6
ENDICOTT NY 13760-9987

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

Program Number
5664-167

File Number
S370/4300-39

SC24-5220-04

Printed in U.S.A.