z/VM
7.4

XEDIT User's Guide

SC24-6338-74

Note:

Before you use this information and the product it supports, read the information in “Notices” on page 129.

This edition applies to version 7, release 4 of IBM® z/VM® (product number 5741-A09) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2024-09-18

Contents

Figures vii

Tables ix

About This Document xi
- Intended Audience xi
- Syntax, Message, and Response Conventions xi
- Where to Find More Information xiv
  - Links to Other Documents and Websites xiv

How to provide feedback to IBM xv

Summary of Changes for z/VM: XEDIT User's Guide xvii
- SC24-6338-74, z/VM 7.4 (September 2024) xvii
- SC24-6338-73, z/VM 7.3 (December 2023) xvii
- SC24-6338-73, z/VM 7.3 (September 2022) xvii
- SC24-6338-01, z/VM 7.2 (September 2020) xvii

Chapter 1. An XEDIT Subset: Full-Screen Text Processing 1
- Editing a File 1
  - XEDIT Command 1
  - Screen Layout 2
- XEDIT and Full-Screen CMS 4
- Entering Data 4
  - INPUT Subcommand 4
  - POWERINP Subcommand 6
- Using Program Function (PF) Keys 8
  - Splitting and Joining Lines 9
  - Scrolling Backward and Forward 10
  - Redisplaying a Subcommand 11
  - Reexecuting a Subcommand 11
  - Inserting Words Using the Insert Mode Key and a NULL Key (PA2) 11
- Using Prefix Subcommands 11
  - Adding Lines Using the A Prefix Subcommand 12
  - Deleting Lines Using the D Prefix Subcommand 12
  - Adding Indented Lines Using the SI Prefix Subcommand 14
  - Duplicating Lines Using the " Prefix Subcommand 17
  - Moving Lines Using the M, MM, F, and P Prefix Subcommands 17
  - Copying Lines Using the C, CC, F, and P Prefix Subcommands 18
  - Setting the Current Line (/) 19
  - Canceling Prefix Subcommands 19
- Moving through a File 20
  - BACKWARD and FORWARD Subcommands 20
  - TOP and BOTTOM Subcommands 20
  - DOWN and UP Subcommands 20
- Making Changes in a File 21
  - CLOCATE Subcommand 22
  - CHANGE Subcommand 22
  - Making a Selective Change 23

Making a Global Change....24
CINSERT Subcommand....25
CFIRST Subcommand....27
Setting Tabs....27
Ending an Editing Session....28
FILE Subcommand....28
QUIT Subcommand....28
SET AUTOSAVE Subcommand....29
Inserting Data from Another File....29
Inserting a Whole File....29
Inserting Part of Another File....31
Getting Help....34
Learning More about the Editor....34
Summary of XEDIT Subset....35

Chapter 2. Practice Exercises....39
Exercise 1. Creating a File....39
Exercise 2. Using Power Typing....40
Exercise 3. Using Prefix Subcommands....42
Exercise 4. Making Changes....43
Exercise 5. Getting It All Together....44

Chapter 3. Using the Editor on a Typewriter Terminal....47
Editing a File....47
XEDIT Command....47
Entering Data....47
INPUT Subcommand....48
Column Pointer....48
QUERY LRECL....49
Moving through a File....49
Line Pointer....49
TYPE Subcommand....49
UP and DOWN Subcommands....49
TOP and BOTTOM Subcommands....50
Making Changes in a File....50
CLOCATE Subcommand....50
CFIRST Subcommand....50
CINSERT Subcommand....51
CDELETE Subcommand....51
CAPPEND Subcommand....51
CHANGE Subcommand....52
Inserting and Deleting Lines....53
Inserting a Line....53
Deleting Lines....54
Recovering Deleted Lines....54
Replacing a Line....55
Moving and Copying Lines....55
MOVE Subcommand....55
COPY Subcommand....56
LPREFIX Subcommand....56
Ending an Editing Session....56
FILE Subcommand....56
QUIT Subcommand....57
SET AUTOSAVE Subcommand....57
Inserting Data from Another File....57
Inserting a Whole File....58
Inserting Part of Another File....58

Using Special Characters....60
SET IMAGE Subcommand....60
Disconnect Mode Restrictions....62
Summary of XEDIT Subset....62

Chapter 4. Using Targets....65
What Is a Target?....65
Using a Target to Change Which Line Is Current....66
A Target as the Operand of a LOCATE Subcommand....67
A Target Preceding a Subcommand....67
Using a Target as a Subcommand Operand....68
Types of Targets....69
A Target as an Absolute Line Number....69
A Target as a Relative Displacement from the Current Line....70
A Target as a Line Name....72
A Target as a Simple String Expression....74
A Target as a Complex String Expression....78
Using Column-Targets....81

Chapter 5. Editing Multiple Files....85
The XEDIT Subcommand....85
Creating a Ring of Files in Storage....85
Editing the Files in the Ring....86
Ending an Editing Session....86
Multiple Logical Screens....87
SET SCREEN Subcommand....88
Multiple Views of the Same File....88
Making Changes from Multiple Views of the Same File....88
Multiple Views of Different Files....88
Order of Processing....89
Cursor Considerations....90

Chapter 6. Tailoring the Screen....93
Tailoring using SET subcommand options....93
Prefix Area....93
Command Line....93
Message Line....93
Current Line....93
Scale....94
Tab Line....94
Color....94
Number....94

Chapter 7. The Macrolanguage....103
What Is an XEDIT Macro?....103
Creating a Macro File....103
Using XEDIT Subcommands in a Macro....104
Communicating between the Editor and the Interpreter....104
Saving and Restoring Editing Variables....107
Entering CMS and CP Commands....107
Avoiding Name Conflicts....107
Walking through an XEDIT Macro....108
A Profile Macro for Editing....111
Executing a Profile Macro....111
Writing a Profile Macro....112
An Example of a Profile Macro....112
Writing Prefix Macros....113

Creating a Sample Prefix Macro ... 113
What Information Is Passed to the Macro? ... 114
Current Line Positioning ... 114
Creating a Second Prefix Macro ... 114
Examining the Source String ... 115
Using the Information That Is Passed ... 115
Handling Blocks ... 115
Assigning a Synonym for a Prefix Macro ... 116
Using the Pending List ... 116
Examining the Argument String ... 117
Positioning the Cursor ... 118
Decoding the Prefix Area ... 118
Using the XEDIT Subcommand ... 119
Additional Examples ... 119
The L Prefix Macro ... 120

Appendix A. Summary of XEDIT Subcommands and Macros ... 121

Notices ... 129
Trademarks ... 130
Terms and Conditions for Product Documentation ... 130
IBM Online Privacy Statement ... 131
Acknowledgements ... 131

Bibliography ... 133
Where to Get z/VM Information ... 133
z/VM Base Library ... 133
z/VM Facilities and Features ... 134
Prerequisite Products ... 136
Related Products ... 136

Index ... 137

Figures

1. Screen Layout 2

2. Input Mode — Typing the Data 5

3. Input Mode — Continue Typing 5

4. Input Mode — Data Entered in the File 6

5. Power Typing 7

6. Power Typing — Data Entered in the File 8

7. Prefix Subcommands A and D — Before and After 13

8. RECOVER Subcommand — Recovering Two Deleted Lines 14

9. Prefix Subcommand SI — Adding the First New Line 15

10. Prefix Subcommand SI — Continuing to Add New Lines 16

11. Prefix Subcommand SI — After 17

12. Prefix Subcommands M and F — Before and After 19

13. The DOWN Subcommand — Before and After 21

14. Using PF5 and PF6 to Make a Selective Change 24

15. Using the PF4 Key for Tabbing 28

16. Inserting a Whole File 30

17. Inserting Part of a File — Call Out the Second File 33

18. Inserting Part of a File — Put Lines to Be Inserted, Then QUIT 33

19. Inserting Part of a File — GET 34

20. Inserting Part of a File — The Lines Are Inserted 34

21. Using a Target to Move the Line Pointer 67

22. Using a Target as a Subcommand Operand 69

23. A Target as an Absolute Line Number 70

24. A Target as a Relative Displacement..... 72

25. A Target as a Line Name.....74

26. A Target as a Simple String Expression..... 79

27. A Target as a Complex String Expression..... 81

28. A Ring of Files in Storage..... 86

29. Editing Files in the Ring.....87

30. Multiple Horizontal Views of the Same File.....89

31. Multiple Vertical Views of Different Files.....91

32. SET PREFIX Subcommand — Before and After..... 95

33. SET CMDLINE Subcommand — Before and After.....96

34. SET CURLINE Subcommand — Before and After..... 97

35. SET SCALE Subcommand — Before and After.....98

36. SET TABLINE Subcommand — Before and After.....99

37. SET MSGLINE on Multiple Lines with Overlay (Part 1 of 2)..... 100

38. SET MSGLINE on Multiple Lines with Overlay (Part 2 of 2)..... 101

39. Sample Macro..... 109

40. A PROFILE XEDIT Macro.....113

41. Sample Prefix Macro.....120

Tables

1. Examples of Syntax Diagram Conventions.....xii

2. XEDIT Subcommand Summary..... 35

3. Prefix Subcommands..... 36

4. PF Key Initial Settings..... 36

5. XEDIT Subcommand Summary..... 62

6. XEDIT Subcommand Summary..... 121

7. Prefix Subcommand Summary..... 126

About This Document

This document is designed to give you a working knowledge of the IBM z/VM editor, XEDIT. XEDIT 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 document 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 z/VM HELP Facility) during an editing session.
- XEDIT support of Double-Byte Character Set (DBCS) strings (KANJI, for example).

Intended Audience

This document was written for the person who has limited data processing experience.

Before reading this document, you should be familiar with the terminal keyboard and you should have some knowledge of CMS. Corequisite publications are the *z/VM: XEDIT Commands and Macros Reference* and the *z/VM: CMS Primer*.

Syntax, Message, and Response Conventions

The following topics provide information on the conventions used in syntax diagrams and in examples of messages and responses.

How to Read Syntax Diagrams

Special diagrams (often called *railroad tracks*) are used to show the syntax of external interfaces.

To read a syntax diagram, follow the path of the line. Read from left to right and top to bottom.

- The ►►—— symbol indicates the beginning of the syntax diagram.
- The ——► symbol, at the end of a line, indicates that the syntax diagram is continued on the next line.
- The ►—— symbol, at the beginning of a line, indicates that the syntax diagram is continued from the previous line.
- The ——►◄ symbol indicates the end of the syntax diagram.

Within the syntax diagram, items on the line are required, items below the line are optional, and items above the line are defaults. See the examples in Table 1 on page xii.

Table 1. Examples of Syntax Diagram Conventions

Syntax Diagram Convention	Example
Keywords and Constants A keyword or constant appears in uppercase letters. In this example, you must specify the item KEYWORD as shown. In most cases, you can specify a keyword or constant in uppercase letters, lowercase letters, or any combination. However, some applications may have additional conventions for using all-uppercase or all-lowercase.	
Abbreviations Uppercase letters denote the shortest acceptable abbreviation of an item, and lowercase letters denote the part that can be omitted. If an item appears entirely in uppercase letters, it cannot be abbreviated. In this example, you can specify KEYWO, KEYWOR, or KEYWORD.	
Symbols You must specify these symbols exactly as they appear in the syntax diagram.	***** Asterisk **:** Colon **,** Comma **=** Equal Sign **-** Hyphen **()** Parentheses **.** Period
Variables A variable appears in highlighted lowercase, usually italics. In this example, *var_name* represents a variable that you must specify following KEYWORD.	

Table 1. Examples of Syntax Diagram Conventions (continued)

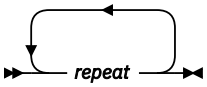
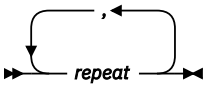
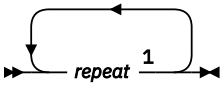
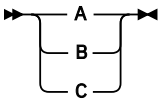
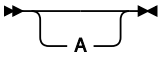
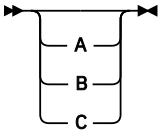
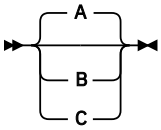
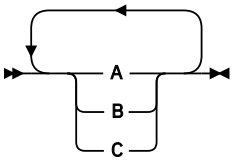
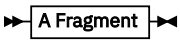
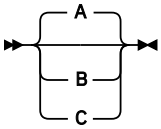
Syntax Diagram Convention	Example
Repetitions An arrow returning to the left means that the item can be repeated. A character within the arrow means that you must separate each repetition of the item with that character. A number (1) by the arrow references a syntax note at the bottom of the diagram. The syntax note tells you how many times the item can be repeated. Syntax notes may also be used to explain other special aspects of the syntax.	   Notes: [1] Specify *repeat* up to 5 times.
Required Item or Choice When an item is on the line, it is required. In this example, you must specify A. When two or more items are in a stack and one of them is on the line, you must specify one item. In this example, you must choose A, B, or C.	 
Optional Item or Choice When an item is below the line, it is optional. In this example, you can choose A or nothing at all. When two or more items are in a stack below the line, all of them are optional. In this example, you can choose A, B, C, or nothing at all.	 
Defaults When an item is above the line, it is the default. The system will use the default unless you override it. You can override the default by specifying an option from the stack below the line. In this example, A is the default. You can override A by choosing B or C.	
Repeatable Choice A stack of items followed by an arrow returning to the left means that you can select more than one item or, in some cases, repeat a single item. In this example, you can choose any combination of A, B, or C.	

Table 1. Examples of Syntax Diagram Conventions (continued)	
Syntax Diagram Convention	**Example**
Syntax Fragment Some diagrams, because of their length, must fragment the syntax. The fragment name appears between vertical bars in the diagram. The expanded fragment appears in the diagram after a heading with the same fragment name. In this example, the fragment is named "A Fragment."	 **A Fragment** 

Examples of Messages and Responses

Although most examples of messages and responses are shown exactly as they would appear, some content might depend on the specific situation. The following notation is used to show variable, optional, or alternative content:

xxx
: Highlighted text (usually italics) indicates a variable that represents the data that will be displayed.

[ ]
: Brackets enclose optional text that might be displayed.

{ }
: Braces enclose alternative versions of text, one of which will be displayed.

|
: The vertical bar separates items within brackets or braces.

...
: The ellipsis indicates that the preceding item might be repeated. A vertical ellipsis indicates that the preceding line, or a variation of that line, might be repeated.

Where to Find More Information

For further information, see the documents listed under “Bibliography” on page 133.

Links to Other Documents and Websites

The PDF version of this document contains links to other documents and websites. A link from this document to another document works only when both documents are in the same directory or database, and a link to a website works only if you have access to the Internet. A document link is to a specific edition. If a new edition of a linked document has been published since the publication of this document, the linked document might not be the latest edition.

How to provide feedback to IBM

We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information. See How to send feedback to IBM for additional information.

Summary of Changes for z/VM: XEDIT User's Guide

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line (**|**) to the left of the change.

SC24-6338-74, z/VM 7.4 (September 2024)

This edition supports the general availability of z/VM 7.4. Note that the publication number suffix (-74) indicates the z/VM release to which this edition applies.

SC24-6338-73, z/VM 7.3 (December 2023)

This edition includes terminology, maintenance, and editorial changes.

SC24-6338-73, z/VM 7.3 (September 2022)

This edition supports the general availability of z/VM 7.3. Note that the publication number suffix (-73) indicates the z/VM release to which this edition applies.

SC24-6338-01, z/VM 7.2 (September 2020)

This edition supports the general availability of z/VM 7.2.

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 CMS experience is assumed. For example, you must know how to log on and enter the CMS environment. You should also be familiar with a CMS file.

This chapter provides a working knowledge of the editor. The subcommands presented here are a subset of XEDIT subcommands selected for text processing on a display terminal; with them you can create a file, enter data, manipulate the screen, change a file, and transfer data between files. (If you have a typewriter terminal, see Chapter 3, "Using the Editor on a Typewriter Terminal," on page 47.) The editor has many additional capabilities, which are described later in this book and in *z/VM: XEDIT Commands and Macros Reference*.

Editing a File

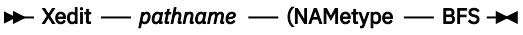
Editing is changing, adding, or deleting data in a CMS file. You make these changes interactively: you instruct the editor to make a change, the editor makes it, and then you request another change.

You also use XEDIT to create a file.

XEDIT Command

After you log on and enter the CMS environment, you can enter the edit environment and begin creating a file. Invoke the editor with the CMS command XEDIT. Its format is:

You can also create a different type of file called a byte file system (BFS) file by entering:



In Figure 1 on page 2, the following command invoked the editor:

```
XEDIT inventor script
```

Before entering data in a file, look at the screen layout illustrated in Figure 1 on page 2.

Note: If your screen layout differs from Figure 1 on page 2 or if some of the commands or PF keys work differently from the way this book says they do, a PROFILE XEDIT macro may be tailoring your editing session. To keep the PROFILE XEDIT macro from executing, add the NOPROFILE option:

XEDIT *filename filetype* (NOPROFILE
or
XEDIT *pathname* (NAMETYPE BFS NOPROFILE

See "A Profile Macro for Editing" on page 111 for more information on the PROFILE XEDIT macro.

If these differences continue while using NOPROFILE, be sure this book was written for your system's release level. See the front cover of this book for its release level and use QUERY CMSREL (see *z/VM: CMS Commands and Utilities Reference*) to find the release level of your system. If they are different, use the guide that matches the release level of your system.

Screen Layout

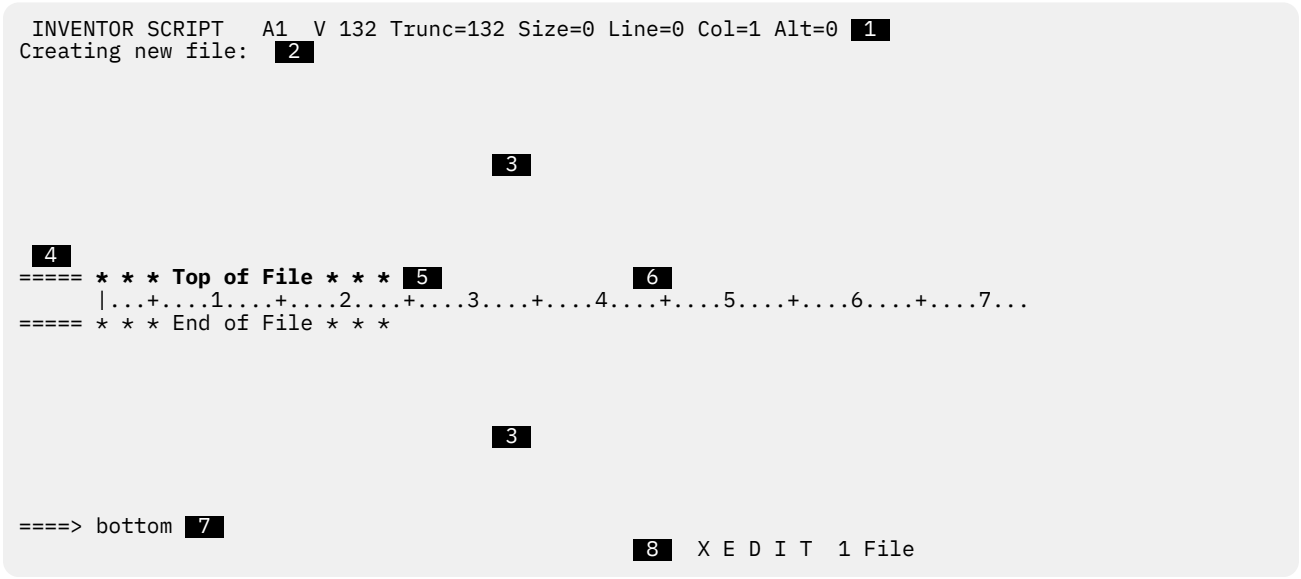


Figure 1. Screen Layout

1

File Identification Line

The first line on the screen identifies the file you are editing. The following information is displayed:

- file name, file type, file mode
- or BFS path name

  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.

- 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.

- truncation column (Trunc=)

  Notice the truncation column is the same as the record length (132). Because a file line can be only 132 characters long, any data you enter beyond 132 characters (in total) can be truncated.

- current number of lines in the file (Size=)

  (Because data has not yet been entered in the file, the number of lines is zero.)

- file line number of the current line (Line=)

  (See number 5, following.)

- position of the column pointer (Col=)

  (See number 6, following.)

- 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 you are creating a new file.

3

File Area

- This part 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 Enter. When you press Enter, the changes are made to the copy of the file 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. Scrolling the screen is like turning the pages of a book.

4

Prefix Area

- The prefix area is the five leftmost columns on the screen, and it displays five equal signs (=====). Each line in the file has a prefix area.

  You can perform various editing tasks such as deleting a line by entering short commands, called prefix subcommands, in the prefix area of a line.

5

The Current Line

- The current line is the file line in the middle of the screen (above the scale). It is *highlighted*, appearing brighter than the other file lines.

  In Figure 1 on page 2, the current line is the Top of File line; the file contains no data yet.

  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, 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.

6

Scale

- The scale appears under the current line to help you edit. It is like the margin scale on a typewriter.

  The vertical bar (|) 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 current column is the column under which the column pointer is positioned.

7

Command Line

- The large arrow (====>) at the bottom of the screen points to the command input area. One way you communicate with the editor is to enter XEDIT subcommands on this line. You can type subcommands in uppercase or lowercase or a combination of both, and many can be abbreviated. For example, BOTTOM, Bottom, and b are all valid ways to type the BOTTOM subcommand.

  After typing a subcommand on the command line, press Enter to execute the subcommand. Figure 1 on page 2 shows the subcommand BOTTOM typed on the command line. (To move the cursor from any place on the screen to the command line, just press Enter or PF12.)

8

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 one file is being edited.

XEDIT and Full-Screen CMS

If you invoke XEDIT from full-screen CMS, the way you see messages other users send you is not the same as when full-screen CMS is off. While full-screen CMS is off, the message appears on a cleared screen with a HOLDING status at the bottom. You can press Clear to get the XEDIT screen back.

If full-screen CMS is on, any message you receive appears 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 Enter. 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 display, 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 Figure 1 on page 2 shows. 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.

```
 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)

ONE OF THE WORLD'S FIRST ELECTRONIC COMPUTERS WAS CALLED ENIAC,
ELECTRONIC NUMERIC 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

In Figure 2 on page 5, the entire input zone has been filled. To stay in input mode and type more data, press Enter once. The lines 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 for you to type more data, as shown in Figure 3 on page 5.

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

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

ONE OF THE WORLD'S FIRST ELECTRONIC COMPUTERS WAS CALLED ENIAC,
ELECTRONIC NUMERIC 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

If you have no more data to type, pressing Enter 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 you press Enter twice. The display is restored to the edit mode screen layout described in Figure 1 on page 2, and the file contains the data.

```
 INVENTOR SCRIPT   A1  V 132  Trunc=132 Size=10 Line=10 Col=1 Alt=10
XEDIT:
===== THE ELECTRONIC COMPUTER (1946)
===== .sp
===== ONE OF THE WORLD'S FIRST ELECTRONIC COMPUTERS WAS CALLED ENIAC,
===== ELECTRONIC NUMERIC 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

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 you can use input mode to insert lines between any two existing lines in the file.

POWERINP Subcommand

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

```
====>  power
```

The advantage of power typing is 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 Enter 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 done typing, press Enter twice to exit from power typing and reenter 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 you enter with 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. 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 you type the following data in power typing mode:

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

The data is 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 is like inserting a box car in a train. Press Reset when you are done inserting characters.

Example of Power Typing

Figure 5 on page 7 shows the same file, INVENTOR SCRIPT, but the data was typed in power typing mode, after entering the POWERINP subcommand. 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.

```
INVENTOR  SCRIPT  A1  * * * P o w e r  T y p i n g * * *       Alt=0
* * * Top of File * * *
ONE OF THE WORLD'S FIRST ELECTRONIC COMPUTERS WAS CALLED ENIAC, ELECTRONIC NUMER
IC INTEGRATOR AND COMPUTER.  IT WAS BUILT BY A GROUP OF RESEARCHERS LED BY AMERI
CAN PHYSICIST JOHN MAUCHLY AT THE UNIVERSITY OF PENNSYLVANIA.  UNLIKE EARLIER CO
MPUTERS, IT RAN ON RADIO TUBES - 18,000 OF THEM IN TOTAL.  IT FILLED A ROOM 30 F
EET BY 50 FEET AND COST $400,000.  USING TEN-DIGIT NUMBERS, IT COULD DO 5,000 AD
DITIONS A SECOND.#.sp#A GERMAN PHYSICIST, ROENTGEN, DISCOVERED THE XRAY BY ACCID
ENT.  HE WAS DOING EXPERIMENTS WITH A CROOKES TUBE, WHICH PRODUCED STREAMS OF EL
ECTRONS CALLED CATHODE RAYS.  ONE DAY HE LEFT AN ACTIVATED CROOKES TUBE ON A BOO
K BEFORE LEAVING THE LABORATORY.  HE DID NOT REALIZE THAT A KEY AND SOME PHOTOGR
APHIC FILM WERE SANDWICHED IN THE BOOK.  LATER, WHEN HE DEVELOPED THE FILM, HE S
AW THE IMAGE OF THE KEY.  THUS WAS THE FIRST XRAY ACCIDENTALLY TAKEN.
```

Figure 5. Power Typing

Notice the pound signs (#) in the eighth line (from the top of the screen). When you enter a pound sign, it causes the data that follows it to begin on a new line. The pound sign itself does not appear in the file.

To leave power typing mode and return to the XEDIT environment, press Enter 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 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 Enter. Figure 6 on page 8 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 Enter.

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

===== ONE OF THE WORLD'S FIRST ELECTRONIC COMPUTERS WAS CALLED ENIAC, ELECTRONIC
===== NUMERIC 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, 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 that is executed when you press the key. Using the PF key saves you the time it takes to type that subcommand on the command line and press Enter.

You can display the PF key settings with the following subcommand:

```
====>  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 18
PF18	ONLY ?

PF19	BEFORE BACKWARD
PF20	BEFORE FORWARD
PF21	ONLY =
PF22	BEFORE RGTLEFT
PF23	BEFORE SPLTJOIN
PF24	BEFORE CURSOR HOME

These are the subcommands the editor assigns to the PF keys. Note 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, see the SET PF subcommand in *z/VM: XEDIT Commands and Macros 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 Enter. 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.

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 definition assigned to one (or more) of the PF keys, you can use the SET PF subcommand, whose format is:

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

For example,

```
====>  set pf1 input
```

assigns the INPUT subcommand to the PF1 key. Pressing PF1 immediately places 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 previously 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, move the cursor under the character where you want the line to be split, and press PF11.

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 PF11 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 under the I in IN:

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

Pressing PF11 splits the line 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 PF11 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 after the end of the data. Pressing PF11 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 PF7, which is set to the BACKWARD subcommand, scrolls the screen backward, toward the top of the file, for one screen display.

Pressing PF8, 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 you have 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 PF6, 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 Enter. If you entered the subcommand incorrectly, you can correct the error by typing over the subcommand displayed in the command line and then pressing Enter.

Reexecuting 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 you use the PF6 key (which is set to the ? subcommand).

Each time you press PF9, the subcommand is executed, saving you the time it takes to retype the subcommand.

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

One way to insert letters, spaces, or words in a line is to press PA2 (or its equivalent) and then use the insert mode key. The PA2 key is initially set to NULLKEY. For information about how to change the initial PA key settings (SET PA*n*), see *z/VM: XEDIT Commands and Macros 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 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, press PA2 again.

Remember to press Reset 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 PA2 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 PA2.)

Using Prefix Subcommands

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

Enter prefix subcommands by typing over any position of the five-character prefix area on one or more lines. When you press Enter, all of the prefix subcommands that have been typed on the screen are executed. Prefix subcommands are not case specific.

Adding Lines Using the A Prefix Subcommand

To add a line, type the single character A in the prefix area. When you press Enter, a blank line is immediately inserted following the line containing the A. A number can precede or follow the A to indicate adding more than one line. For example, A5 adds five blank lines.

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.
```

You can then type information 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.

Deleting Lines Using the D Prefix Subcommand

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

A number can precede or follow the D to indicate deleting more than one line.

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 you press Enter, 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 you have typed one DD and pressed Enter, 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 you press Enter, the entire block of lines is deleted.

Figure 7 on page 13 is a before-and-after example of the A and D prefix subcommands.

```
 DESSERT  RECIPES  A1  F 80  Trunc=80 Size=15 Line=9 Col=1 Alt=0

===== * * * Top of File * * *
==A== CHOCOLATE SAUCE
=====       12    OUNCES SEMI-SWEET CHOCOLATE
=====        2    OUNCES UNSWEETENED CHOCOLATE
D====             DOLLOP MUSTARD
=====        1    CUP HEAVY CREAM
===2A        2    OUNCES COGNAC
=DD== VINAIGRETTE SAUCE
=====        1/2  CUP OLIVE OIL
=====        1/2  CUP VINEGAR
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
=====             PINCH OF SALT
DD===             DASH OF PEPPER
===== APRICOT GLAZE
=====
=====        1    JAR APRICOT PRESERVES (1 POUND)
=====        2    TABLESPOONS KIRSCH
===== * * * End of File * * *



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

```
 DESSERT  RECIPES  A1  F 80  Trunc=80 Size=12 Line=9 Col=1 Alt=1

===== * * * Top of File * * *
===== CHOCOLATE SAUCE
=====
=====       12    OUNCES SEMI-SWEET CHOCOLATE
=====        2    OUNCES UNSWEETENED CHOCOLATE
=====        1    CUP HEAVY CREAM
=====        2    OUNCES COGNAC
=====
=====
===== APRICOT GLAZE
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
=====
=====        1    JAR APRICOT PRESERVES (1 POUND)
=====        2    TABLESPOONS KIRSCH
===== * * * End of File * * *







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

Figure 7. Prefix Subcommands A and D — Before and After

Recovering Deleted Lines

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:

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 put them back where they belong (by using the M prefix subcommand, discussed later.)

To recover *all* lines you deleted during an editing session, enter:

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

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

```
====>  recover 2
```

results in:

```
 DESSERT  RECIPES  A1  F 80  Trunc=80 Size=14 Line=9 Col=1 Alt=1
2 line(s) recovered
===== * * * Top of File * * *
===== CHOCOLATE SAUCE
=====
=====       12    OUNCES SEMI-SWEET CHOCOLATE
=====        2    OUNCES UNSWEETENED CHOCOLATE
=====        1    CUP HEAVY CREAM
=====        2    OUNCES COGNAC
=====
=====
=====             PINCH OF SALT
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
=====             DASH OF PEPPER
===== APRICOT GLAZE
=====
=====        1    JAR APRICOT PRESERVES (1 POUND)
=====        2    TABLESPOONS KIRSCH
===== * * * End of File * * *



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

Figure 8. RECOVER Subcommand — Recovering Two Deleted Lines

Adding Indented Lines Using the SI Prefix Subcommand

To continuously add lines of indented text, type the characters SI in the prefix area. When you press Enter, 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 15 through Figure 11 on page 17 shows how new indented lines are added.

Type the prefix subcommand SI 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
=====
=====           1/2    Pound of butter
SI===           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 you press Enter, 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

Enter text 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 you press Enter, a new line is automatically added following the one you just typed on. Each time you type on the new line and press Enter, another new line is 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, press Enter 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 * * *









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

Figure 11. Prefix Subcommand SI — After

To add a blank line in a file while using SI, make at least one change (such as pressing the spacebar once) on the line that contains `. . . . .` in the prefix area. Just using the cursor position keys to move the cursor 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 Enter while the cursor is away from the new line, another new line is added following the last line that was added. SI is canceled only if you press Enter and have typed no text on the new line.

Duplicating Lines Using the " Prefix Subcommand

To duplicate a line, enter the character " (double quotation mark) 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.
===== They will appear before this line.
```

When you press Enter, 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.
===== They will appear before this line.
```

To duplicate a block of lines either one time or a specified number of times, you can type "" (two double quotation marks) 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 you type one "" and press Enter, the status area of the screen displays **"" pending...**. This lets you scroll the screen before completing the block and pressing Enter.

Moving Lines Using the M, MM, F, and P Prefix Subcommands

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

When you press Enter, 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 moving more than one line, for example, 5M or M5 in the prefix area.

The line to move and the destination line can be on different screens. When you have entered an M or F (or P), 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 Enter.

Figure 12 on page 19 is a before-and-after example of the M prefix subcommand.

Copying Lines Using the C, CC, F, and P Prefix Subcommands

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 is indicated by F or P.

```
 BOOKS    LIST     A1  F 80  Trunc=80 Size=17 Line=8 Col=1 Alt=0

===== * * * Top of File * * *
===== AUSTEN DEPICTS SOCIETY FROM THE DRAWING ROOM IN:
=====    PRIDE AND PREJUDICE
=====    SENSE AND SENSIBILITY
===== COLLINS, A CONTEMPORARY OF DICKENS, WROTE SENSATION NOVELS:
=====    ARMADALE
=====    THE MOONSTONE
=====    THE WOMAN IN WHITE
=mm== ELIOT'S MASSIVE NOVELS DEPICTED SOCIETAL FLAWS:
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
=====    DANIEL DERONDA
=====    FELIX HOLT
=mm==    MIDDLEMARCH
===== DOSTOEVSKY'S CHARACTERS ARGUE PHILOSOPHY IN DIALECTIC NOVELS:
=====    CRIME AND PUNISHMENT
=====    THE BROTHERS KARAMAZOV
==f==    THE POSSESSED
===== FAULKNER WROTE IN TRAIN OF CONSCIOUSNESS IN:
=====    THE SOUND AND THE FURY

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

```
 BOOKS    LIST     A1  F 80  Trunc=80 Size=17 Line=8 Col=1 Alt=1

===== * * * Top of File * * *
===== AUSTEN DEPICTS SOCIETY FROM THE DRAWING ROOM IN:
=====    PRIDE AND PREJUDICE
=====    SENSE AND SENSIBILITY
===== COLLINS, A CONTEMPORARY OF DICKENS, WROTE SENSATION NOVELS:
=====    ARMADALE
=====    THE MOONSTONE
=====    THE WOMAN IN WHITE
===== DOSTOEVSKY'S CHARACTERS ARGUE PHILOSOPHY IN DIALECTIC NOVELS:
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
=====    CRIME AND PUNISHMENT
=====    THE BROTHERS KARAMAZOV
=====    THE POSSESSED
===== ELIOT'S MASSIVE NOVELS DEPICTED SOCIETAL FLAWS:
=====    DANIEL DERONDA
=====    FELIX HOLT
=====    MIDDLEMARCH
===== FAULKNER WROTE IN TRAIN OF CONSCIOUSNESS IN:
=====    THE SOUND AND THE FURY

====>
                                                          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.

You can type the / (diagonal) prefix subcommand in the prefix area of any line on the screen. When you press Enter, 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 *on the command line:*

```
====>  reset
```

When you press Enter, 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 Enter, you can press Clear to remove them.

Moving through a File

XEDIT lets you move backward, forward, to the top and bottom, and up and down in a file.

BACKWARD and FORWARD Subcommands

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. You can also enter the BACKWARD and FORWARD subcommands in the command line.

The format of these subcommands is:

where *n* is the number of screen displays you want to scroll backward or forward. (This is like pressing PF7 or PF8 *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 can 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 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 you omit the number, 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 you omit the number, 1 is assumed.

Figure 13 on page 21 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 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.

CLOCATE Subcommand

The CLOCATE subcommand searches a file, beginning with the column after the current column in the current line, for a character string 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:

Enclose the string in delimiters. In the examples in this book, the delimiter is a diagonal (/), but you can use any character that does not appear in the string itself except for a plus (+), minus (–), not (¬), or period (.). 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 CLOCATE/be/ (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 you enter CLOCATE/be/ 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 you enter CLOCATE/be/, 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 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/
```

searches backward 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:

For example:

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

====>  change/rose/daisy/
```

```
===== 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 PF5 to locate each occurrence of the old word, examine it, and then either change it (by pressing PF6) or go on to the next occurrence (by pressing PF5).

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 Enter*.
3. Press PF5. The cursor moves under the first occurrence of *oldword*, and the line that contains it is highlighted.
4. If you want to change the word, press PF6. If not, press PF5 again, and step number 3 will be repeated.

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

Figure 14 on page 24 shows 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 Enter was not pressed:

```
====>  change/rose/daisy/
```

This subcommand is executed when the PF6 key is pressed.

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

In the bottom screen, PF5 was successively pressed until the last occurrence of "rose". Then PF6 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 PF5, the cursor moves under the next occurrence of the string and the line is highlighted. Pressing PF6 has no effect.

For more information on making a selective change, see SCHANGE in *z/VM: XEDIT Commands and Macros 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.
====>
                                                          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

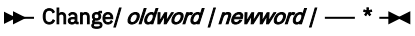
If you want to make a global change, that is, change *every occurrence* of a word *throughout the file,* first make the first line of the file the current line and then use the following form of the CHANGE subcommand:

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 changes only the first occurrence in each line of a word throughout the file:



CINSERT Subcommand

Often, you need to insert words in a line. You have already seen how to use the PA2, 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,

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 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 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 wrong.
```

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

Notice 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 you press the PF4 key, the cursor is positioned under the next tab column, where you can enter data.

The editor defines initial tab settings according to file type; you can display them with 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 you press PF4, the cursor moves to column 10 on the screen. The second time, it moves to column 20, and so forth.

You can use PF4 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
```

This displays the current SET TABS subcommand in the command line; you can type over the numbers and press Enter to define new tabs.

Figure 15 on page 28 is an example of data entered using PF4 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

You can end an editing session using FILE or QUIT. The SET AUTOSAVE subcommand automatically saves your file for you while you are editing it.

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 brought into virtual storage (when the XEDIT command is entered). However, virtual storage is *temporary.* To write a new or modified file to disk, SFS directory, or byte file system for *permanent* storage, 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. You must use FFILE to file an empty file. It will only be saved if the file is located on a directory in a file pool managed by a server at the z/VM release 1.1 or later level.

QUIT Subcommand

The QUIT subcommand ends an editing session and leaves the permanent copy of the file intact on the 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
```

Use the QUIT subcommand instead of the FILE subcommand when you edit a file just 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 the file is new and you have not input any data, the file is not written to disk or directory. Otherwise, if a file is new or has been changed, the editor gives you a warning message to prevent your inadvertently using QUIT instead of FILE. The message is:

```
File has been changed; type 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 SFS directories are not affected if the system malfunctions. However, a new file you are creating or the changes you are making to an existing file might be lost if the system fails. To minimize the risk of losing your data, use the SET AUTOSAVE subcommand, which causes your file to be automatically written to disk or SFS directory (or *saved*) after you have typed in or changed a certain number of lines. Its format is:

where *n* is the number of typed in or changed lines.

For example, to write the file to disk or SFS directory every time you have changed 10 lines, enter:

```
====>  set autosave 10
```

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

You can enter the SET AUTOSAVE subcommand at any time during an editing session, but it is a good idea to enter it right after you enter an XEDIT command to create a new file or to call an existing file from disk or SFS directory.

When a file is automatically saved, it is written into a new file whose file name is a number and whose 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. To replace a BFS file with the AUTOSAVE file, use the OPENVM PUTBFS 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 or revising an existing file, and then enter a QUIT subcommand, the new or revised file is not saved, but the AUTOSAVE file is available from disk or SFS directory.

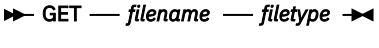
Inserting Data from Another File

The GET subcommand inserts all or part of another file into the file you are editing after the current line.

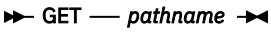
Before entering the GET subcommand, make the current line the line *after which* you want to insert data. For example, to insert another file at the *end* of a file, you use the BOTTOM subcommand. To insert another file in the *middle* of a file, use the / prefix subcommand to make the desired line current.

Inserting a Whole File

Suppose you are 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:



or



XEDIT determines whether the file ID entered is a CMS file (*filename filetype*) or a BFS file (*pathname*) from the NAMETYPE setting. The default type of file ID is CMS, but NAMETYPE BFS can be entered on the XEDIT command line to change the file ID to a BFS file.

Figure 16 on page 30 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 is entered:

```
====>  get cookies cookbook
```

In the bottom screen, the message EOF `reached` indicates 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, specify in the GET subcommand the line number of the first line and the number of lines to insert. The following GET subcommand inserts the first 10 lines of a second file:

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

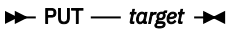
If you do not know the line numbers, you can edit a second file without ending your current editing session, put the lines you want to insert into a temporary file, and insert this into your current file.

Note: In the byte file system, you cannot GET part of a BFS 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, make the line after which you want to insert data the current line. Then, without ending your current editing session, enter an XEDIT subcommand to edit the second file. The second file will appear on the screen.
2. The PUT subcommand stores lines in a temporary holding area, *starting with the current line*, up to an ending, or target line. Make the first line you want to insert the current line. Then enter the PUT subcommand; its format is:

   where *target* identifies the end of the group of lines to insert. You can specify a target in various ways. For more information, see Chapter 4, “Using Targets,” on page 65. Three ways are described here. They are all equivalent; you can choose the method you prefer.

   Starting with the current line, you can count the number of lines you want to insert and specify this number as the target. 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
   ```

   Or you can specify the target as a character string; the editor stores lines, from the current line, up to, but not including, the line containing the string.

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

   ```
   ====>  put/television/
   ```

   You can also specify a target as the file line number. To display the line numbers in the prefix area, enter:

   ```
   ====>  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 to return to your original file.
4. Then enter:

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

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

Figure 17 on page 33 through Figure 20 on page 34 shows how the PUT and GET subcommands are used to insert part of a file into another file.

The file DESSERT COOKBOOK contains a recipe for cream puffs. Recipes for sauces are in a separate file, SAUCES COOKBOOK. To insert the recipe for chocolate sauce after the recipe for cream puffs:

1. XEDIT DESSERT COOKBOOK and make the current line the line after which you want to insert the sauce recipe (use the / prefix subcommand). The current line should be the last line of the cream puffs recipe (Figure 17 on page 33).

   Then enter the following subcommand:

   ```
   ====>  xedit sauces cookbook
   ```

2. This file appears on the screen. The status area (lower right corner) indicates 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 33). Now enter the subcommand to store the chocolate sauce recipe:

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

3. The stored lines begin with CHOCOLATE SAUCE and end with the line *preceding* VINAIGRETTE. You could also have entered the PUT subcommand 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 you enter the PUT subcommand, quit this file by entering:

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

4. The original file comes back on the screen (Figure 19 on page 34). To insert the lines that were stored, enter:

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

   The sauce recipe is inserted, as shown in Figure 20 on page 34. 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, press PF1, which is set to the HELP MENU subcommand.

When you press PF1, a list of all subcommands and macros available with the editor appears on the screen. Move the cursor to the desired subcommand and press Enter. The subcommand description appears on the screen, replacing the HELP Menu. Pressing PF3 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 *z/VM: XEDIT Commands and Macros Reference*.

ALL
Displays a collection of lines for editing, excluding others from display.

ALTER
Changes a character to one that is not available on your keyboard, like a backspace character.

COMPRESS, EXPAND
Repositions data in new tab columns without retyping.

LEFT, RIGHT
Displays columns of data that extend to the left or right of the screen display.

LOWERCAS, UPPERCAS
Translates alphabetic characters to all lowercase or all uppercase.

MERGE
Combines two sets of lines.

SET ARBCHAR
Controls whether you can specify only the beginning and end of a long string to be located or changed.

SET CASE
Controls whether data you type is entered in the file the same way you type it or translated into uppercase.

SET POINT
Assigns name(s) to any line; you can reference the name(s) in XEDIT subcommands.

SET SCREEN
Controls dividing the screen so you can edit multiple files or multiple views of the same file.

SET VERIFY
Controls whether changed lines are displayed in the message area and what columns of data are displayed, in character or hexadecimal or both.

SORT
Arranges the file lines in alphabetic order.

< (SHIFT LEFT) MACRO
This prefix macro shifts one line or a block of lines one or more columns to the left.

> (SHIFT RIGHT) MACRO
This prefix macro shifts one line or a block of lines one or more columns to the right.

Summary of XEDIT Subset

The following table summarizes the subcommands in this chapter. Minimum abbreviations of subcommands are in uppercase letters.

Table 2. XEDIT Subcommand Summary

Function	**Subcommand/PF Key**
To create or edit a file	**X**edit
To enter data	**I**nput **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

Table 2. XEDIT Subcommand Summary (continued)	
Function	**Subcommand/PF Key**
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 with a subsequent GET	**PUT**
To imbed a complete or a partial copy of one file in another	**GET**
To cancel pending prefix subcommands	**RES**et

Table 3. Prefix Subcommands	
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	**/**

Table 4. PF Key Initial Settings	
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**

Table 4. PF Key Initial Settings (continued)	
PF Keys, Initial Settings:	
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 you press the key 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 from the command line to the screen.	**PF12, PF24**

Chapter 2. Practice Exercises

This chapter will give you practice in using some of the XEDIT subcommands discussed in Chapter 1, “An XEDIT Subset: Full-Screen Text Processing,” on page 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 are asked to type contains errors so you can use subcommands to correct them.

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

Exercise 1. Creating a File

This part of the exercise covers the following subcommands: SET AUTOSAVE, QUERY TABS, SET TABS, INPUT, 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 Enter. 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 an SFS directory at periodic intervals, enter the following subcommand:

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

You will enter data in the file using PF4 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 PF4, and the cursor moves to the column (10) you specified in the SET TABS subcommand. Type:

```
          Telescope
```

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

```
                              1608
```

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

```
          Hot air balloon
```

Press PF4 and then type:

```
                          1783
```

Using PF4 to move the cursor, type the following:

```
          Margarine           1869
          Tranquilizer        1952
```

Now press Enter. The status area (lower right corner) shows 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 Enter 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 for 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 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 ENTER 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. (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 PF11, 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 PA2, 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 Reset 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 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 Enter, that line becomes the new current line.

When your file is too big to fit on one screen, you can use PF7 and PF8 (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 Enter twice to reenter 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 Enter. The sentences have been reversed.

Type an A in the prefix area of the first sentence in the file and press Enter. In the blank line you just added, type:

```
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 PF2, 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 Enter. 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 Enter.

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 Enter 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 Enter.*

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

Now press PF5. 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 PF6. This "bitter" is fine, so press PF5 again.

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

Press PF5.
Press PF6.

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 E0F `reached` tells you the entire file has been inserted. The new current line is the last line 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 the status area indicates 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 written for the person with limited data processing experience, but some CMS experience is assumed. For example, you must know how to log on and enter the CMS environment. You should also be familiar with the concept of a CMS file.

This chapter provides a working knowledge of the editor. The subcommands presented here are a subset of XEDIT subcommands for text processing on a typewriter terminal. You can use them to create a file, enter data, change the file, and transfer data between files. Many of the subcommands presented in this chapter work similarly in disconnect mode. For more information, see "Disconnect Mode Restrictions" on page 62. The editor has many additional capabilities, described in the rest of this book and in *z/VM: XEDIT Commands and Macros Reference*.

Editing a File

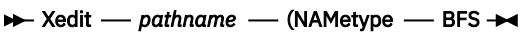
Editing is changing, adding, or deleting data in a CMS file. You make these changes interactively: you tell the editor to make a change, the editor makes it, and then you request another change.

When you edit a file that does not exist, you are creating a file.

XEDIT Command

After you log on and enter the CMS environment, you can enter the edit environment. Invoke the editor with the CMS command XEDIT, whose format is:

You can also create a different type of file called a byte file system (BFS) file by entering:



If the file does not already exist, the editor creates it in virtual storage.

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 change lines in that file. Enter XEDIT subcommands by typing the subcommand and then pressing the Return key. (You can enter XEDIT subcommands in uppercase or lowercase, or a combination of both.)

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. In this chapter, anything the editor displays is enclosed in a box. Subcommands or data you enter are not.

Now let us create a simple file called POEM1 SCRIPT. Enter:

```
xedit poem1 script
```

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

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

Entering Data

You enter data with the INPUT subcommand. The QUERY LRECL subcommand lets you find the logical record length of your file.

INPUT Subcommand

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

To enter data in the file, you must be in input mode. To enter input mode, type the following subcommand and press the Return key:

```
input
```

The editor displays the following message:

```
  Input mode:
```

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

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 done typing data and want to return to edit mode (to make changes to the file or to end the editing session), press the Return key on a null line.

The editor responds:

```
  XEDIT:
```

While you are editing, you can enter input mode at any time to insert new lines of data in the file. 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 insert lines between any two existing file lines.

The following two subcommands (discussed later) display the data you entered:

```
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 the underscore below the first letter in each line. The underscoring represents the *column pointer*.

Some subcommands start at the column pointer to perform their editing functions; XEDIT subcommands that are discussed later move the column pointer.

QUERY LRECL

No line can be longer than the logical record length of the file, which varies with file type. To find out the logical record length of any file, enter:

```
query lrecl
```

In the preceding example, 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.

Moving through a File

You can move through a file with the TYPE, UP, DOWN, TOP, and BOTTOM subcommands.

Line Pointer

The line pointer points to the current line. The current line changes as you move up and down in a file editing various lines. When the line that is current changes, the line pointer moves.

TYPE Subcommand

Many XEDIT subcommands perform their functions starting with the current line. Therefore, you often need to know which line is current so you can move the line pointer, if necessary.

To display the current line, enter:

```
type
```

To display more than one line, enter the TYPE subcommand with the number of lines. For example, to display five lines, beginning with the current line, enter:

```
type 5
```

To display the entire file, first position the line pointer at the top of the file:

```
top
```

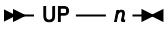
Then, to display all the lines in the file, enter:

```
type *
```

After the TYPE subcommand is executed, the line pointer is at the last line that was displayed.

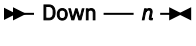
UP and DOWN Subcommands

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:



where *n* is the number of lines to move the line pointer. If you omit the number, 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:



where *n* is the number of lines to move the line pointer. If you omit the number, 1 is assumed.

To insert new lines of data after any existing file line:

1. Enter the UP or DOWN subcommand to move the line pointer to the line after which you want data inserted.
2. 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 before the first line of the file, enter:

```
top
```

To move the line pointer to the last file line, enter:

```
bottom
```

To begin entering new lines either at the beginning or the end of a file, enter:

```
top (or bottom)
input
```

Making Changes in a File

Subcommands that insert, delete, or change characters operate from the column pointer, represented as an underscore character (_).

CLOCATE Subcommand

The CLOCATE subcommand searches a file, *starting with the column after the column pointer in the current line*, for a character string you specify. One format of the CLOCATE subcommand is:

Enclose the string in delimiters, such as the diagonal (/). You can use any character except plus (+), minus (–), not (¬), or period (.) that is not in the character string (for example: CLOCATE?VM/CMS?).

If the string is found, the line containing the string becomes the new current line (and is displayed); the column pointer moves under the first character of the string.

For example, for the file POEM1 SCRIPT, the subcommands:

```
top
```

```
clocate/legs/
```

display the following line:

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

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 by entering the CFIRST subcommand; enter:

```
cfirst
```

In the previous example, with the column pointer under the “l” in “legs”, entering CFIRST results in:

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

CINSERT Subcommand

The CINSERT subcommand inserts characters immediately before the column pointer.

To insert the phrase "exactly 29,000 feet" before the word "high" in the following sentence

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

first move the column pointer to the first character in "high" by entering:

```
clocate/high/
```

Then insert the phrase:

```
cinsert exactly 29,000 feet
```

(Press the spacebar once after the word "feet" so a blank separates "feet" and “high”.) The resulting line is:

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

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 with one too many “or not to be”:

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

Because deletion starts at the column pointer, first move the column pointer under the first “or not to be” by entering:

```
clocate/or/
```

Then, count the number of characters to delete, starting at the column pointer, and use the CDELETE subcommand:

```
cdelete 13
```

The resulting line is displayed:

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

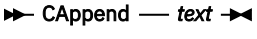
To avoid counting the number of characters, you can specify the operand of the CDELETE subcommand as a character string:

```
cdelete/or/
```

This form of the CDELETE subcommand means, "delete characters *from* the column pointer *to* the first character of the string specified in the operand." The result is the same: the extra "or not to be" is removed.

CAPPEND Subcommand

The CAPPEND subcommand adds data to the end of the current line. Its format is:



where *text* is the data to add to the end of the line.

For example, a file contains the following line:

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

To add “and he stoppeth one of three.” at the end of the line, enter:

```
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.
```

CHANGE Subcommand

The CHANGE subcommand replaces one word with another. Use the following form to change the first occurrence of a word in the current line:

For example, the current line is:

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

```
change/rose/daisy/
```

results in:

```
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, when you replace a word with a shorter word, the editor removes extra blanks.

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

To change *every occurrence* of a word, first move the line pointer to the line where you want the change to begin, and then use the following form:

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.)

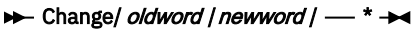
```
  A rose is a rose is a rose.
  A rose is a rose is a rose.
  A rose is a rose is a rose.
  A rose is a rose is a rose.
```

```
change/rose/daisy/ * *
```

produces the following changes (the editor displays only changed lines):

```
  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 changes only the first occurrence in each line of a word throughout the file:



Making a Selective Change

Suppose you want to change one word to another only some of the time. You can repeatedly execute the CLOCATE subcommand to scan the file, entering a CHANGE subcommand only when you want to make the change.

= Subcommand

Or, instead of typing the 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, type an equal sign (=) and press the Return key.

Inserting and Deleting Lines

You can insert and delete lines with the INPUT, DELETE, RECOVER, and REPLACE subcommands.

Inserting a Line

To insert a single line of data between existing lines enter 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 after the current line, without leaving edit mode. (To insert more than one line, enter the INPUT subcommand with no operand to enter input mode.)

To insert a blank line, 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, 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. To insert a title line, enter:

```
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, enter:

```
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, beginning with the current line.

To delete only the current line, enter:

```
delete
```

To delete more than one line, specify the number of lines in the operand; for example,

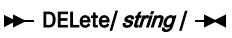
```
delete 5
```

deletes five lines, including the current line.

To delete the rest of the file, enter:

```
delete *
```

To delete a number of lines without counting how many, you can use the form:



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

For example, 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
```

```
delete/bread/
```

deletes all lines from the current line up to, but not including, the line containing “bread”. All that remains is:

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

Recovering Deleted Lines

If you delete one or more lines and change your mind, you can recover the lines anytime during an editing session with the RECOVER subcommand:

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

The last lines deleted are the first lines recovered. For instance, in our previous example of deleting lines, if you enter:

```
recover 2
```

you get the radio and frisbee back:

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

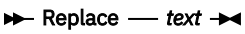
Recovered lines are inserted at the current line. If the lines belong elsewhere in the file, you put them back with the MOVE subcommand, discussed later.

To recover *all* lines you deleted during an editing session, enter:

```
recover *
```

Replacing a Line

The REPLACE subcommand deletes the current line and replaces it with the text you specify. Its format is:



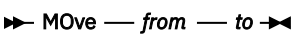
If you enter the REPLACE subcommand with no *text*, the editor deletes the current line and places you in input mode.

Moving and Copying Lines

You can move and copy lines with the MOVE, COPY, and LPREFIX subcommands.

MOVE Subcommand

The MOVE subcommand removes one or more lines, starting with the current line, from their current line and inserts them elsewhere in the file. Its format is:



The first operand is the number of lines to move, starting with the current line. The second operand tells the destination; moved lines are deleted from their original location and inserted *after* the destination.

For example, to move the current line three lines down in the file, enter:

```
move 1 3
```

To move the current line and the two lines after it three lines down in the file, enter:

```
move 3 3
```

To move a line *backward* in the file, specify a minus (–) sign in front of the destination operand. For example,

```
move 1 -3
```

moves the current line up two lines in the file.

To avoid counting lines, you can specify the destination operand as a character string. For example,

```
move 1 /string/
```

moves the current line after the line containing the string.

To move a line backward in the file, specify a minus (–) sign before the string. For example:

```
move 1 -/string/
```

In the following example, the top line is the current line:

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

Either of the following subcommands moves the line containing "filberts" (the current line) after the line containing “chestnuts”.

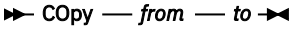
```
move 1 3 or move 1 /chestnuts/
```

The resulting file looks like this:

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

COPY Subcommand

The COPY subcommand duplicates one or more lines, starting with the current line, and places them after the destination line; the original line(s) remain in place. The format of COPY is:



LPREFIX Subcommand

The LPREFIX subcommand simulates writing in the prefix area of the current line, even though no prefix area is available on typewriter terminals. LPREFIX performs some of the functions (such as moving or copying lines) that prefix subcommands and macros provide on display terminals. For a description of the LPREFIX subcommand, see *z/VM: XEDIT Commands and Macros Reference*.

Ending an Editing Session

You can end an editing session using FILE or QUIT. The SET AUTOSAVE subcommand automatically saves your file for you while you are editing it.

FILE Subcommand

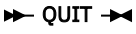
New files are created and changes are made to existing files in virtual storage. Virtual storage is *temporary.* To write a new or modified file to disk, SFS directory, or byte file system for *permanent* storage, enter:

```
file
```

This writes the file disk or directory and returns control to CMS. You must use FFILE to file an empty file, and it will only be saved if the file is located on a directory in a file pool managed by a server at the z/VM release 1.1 or later level.

QUIT Subcommand

The QUIT subcommand ends an editing session and leaves the permanent copy of the file intact on the disk or directory. Its format is:



Use QUIT instead of FILE if you edit a file just to examine, but not change, its contents, or if you have made errors in changing a file and do not want them recorded.

When you enter QUIT, a new file with no input is not written to disk or directory. Otherwise, the editor gives you a warning message to prevent your inadvertently using QUIT instead of FILE:

```
File has been changed; type QQUIT to quit anyway
```

If you really do not want to save the file, enter QQUIT (abbreviated QQ). To save the changes, enter FILE.

SET AUTOSAVE Subcommand

Files on disks or SFS directories are not affected if the system malfunctions. But a new file 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 with the SET AUTOSAVE subcommand. It causes your file to be written to disk or SFS directory (or *saved*) after you have typed in or changed a certain number of lines. Its format is:

where *n* is the number of typed in or changed lines.

For example, to write the file to disk or SFS directory, or *save* it every time you have changed 10 lines, enter:

```
set autosave 10
```

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

You can enter the SET AUTOSAVE subcommand any time during an editing session, but it is a good idea to enter it right after you enter an XEDIT command to create a new file or edit an existing file.

When a file is automatically saved, it is written into a new file whose file name is a number and whose 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 so, replace the original file with the AUTOSAVE file by using the CMS COPYFILE command with the REPLACE option. To replace a BFS file with the AUTOSAVE file, use the OPENVM PUTBFS command with the REPLACE option. Then, erase the AUTOSAVE file and resume editing.

If you enter a SET AUTOSAVE subcommand and then enter a QUIT subcommand, the new or revised file is not saved, but the AUTOSAVE file is still available from disk or SFS directory.

Inserting Data from Another File

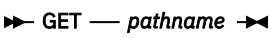
The GET subcommand inserts all or part of another file into the file you are editing after the current line. (A file you *get* is not destroyed; a copy of that file is inserted.)

Before entering the GET subcommand, make the current line the line *after which* you want to insert data. To insert another file at the *end* of your file, use the BOTTOM subcommand to make the last line current. To insert another file somewhere in the *middle* of your file, use the UP or DOWN subcommand to make the desired line current.

Inserting a Whole File

To insert all of another file into the file you are editing, use the format:


```
           or
```



XEDIT determines whether the file ID entered is a CMS file (*filename filetype*) or a BFS file (*pathname*) from the NAMETYPE setting. The default type of file ID is CMS, but NAMETYPE BFS can be entered on the XEDIT command line to change the file ID to a BFS file.

For example, to insert all of POEM2 at the end of POEM1, while you are editing POEM1, enter:

```
bottom
```

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

```
get poem2 script
```

(insert the whole file)

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

```
EOF reached
```

Inserting Part of Another File

To insert part of another file, specify in the GET subcommand the line number of the first line and the number of lines to insert. For example, the following GET subcommand inserts the first 10 lines of a second file:

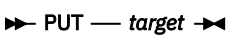
```
get file2 data 1 10
```

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 that into your current file.

Note: In the byte file system, you cannot GET part of a BFS file.

Follow these steps:

1. In the first file, make the line after which you want to insert data the current line. Then, without ending your current editing session, enter an XEDIT subcommand to call out the second file.
2. Make the first line you want to insert the current line. Use the PUT subcommand to store the lines you want to insert into a temporary holding area. Its format is:

   where *target* identifies the end of a group of lines to insert. You can specify a target in various ways, described in Chapter 4, “Using Targets,” on page 65. Two ways are described here.

   Starting with the current line, you can count the number of lines you want to insert and specify this number as the target. 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
```

Or you can specify the target as a character string. The editor stores lines from the current line up to, but not including, the line containing the string. For example, the following subcommand stores the first three lines, but not the line containing “a portable television”.

```
put/television/
```

3. Enter a QUIT subcommand to return to your original file.
4. Then enter:

```
get
```

No operands are required. The lines stored with the PUT subcommand are inserted. The last line inserted becomes the new current line.

The following example illustrates these steps. A file, called DESSERT COOKBOOK, contains a recipe for cream puffs. Recipes for sauces are in a separate file, SAUCES COOKBOOK. To insert the recipe for chocolate sauce after the recipe for cream puffs:

1. XEDIT DESSERT COOKBOOK and make the current line the line after which you want to insert the sauce recipe (the blank line between HEAVY CREAM and ALMOND COOKIES):

```
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
```

Then call out the second file, SAUCES COOKBOOK:

```
xedit sauces cookbook
```

2. Make the first line you want to insert (CHOCOLATE SAUCE) the current line.

```
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
```

Enter a PUT subcommand:

```
put/VINAIGRETTE/ or put 7
```

3. Return to the DESSERT COOKBOOK file:

```
quit
```

The original file is now being edited.

4. To insert the lines stored that were stored, enter:

```
get
```

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

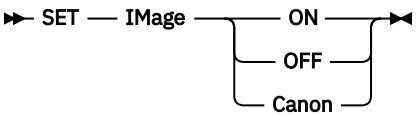
The SET IMAGE subcommand lets you use special characters.

SET IMAGE Subcommand

The SET IMAGE subcommand controls how special characters entered on an input line are represented in a file. SET IMAGE affects these special characters:

- Tab characters (X'05')
- Backspace characters (X'16')

The format of the SET IMAGE subcommand is:



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

Tab Characters

You set physical tab settings manually on the typewriter. Logical tab settings indicate the column positions where fields within a record begin. The SET TABS subcommand defines them. Its format is:

where *n* represents the column numbers for the logical tab settings.

The SET IMAGE setting determines how the data is entered in the file when you press the TAB key. With SET IMAGE ON, the editor replaces tab characters with blanks, from the column where you pressed the TAB key, to the last column before the next logical tab setting. The next character you enter 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 beginning with a tab character, the first data character after the tab is written into the file in column 15, regardless of the physical tab stop on the terminal.

With SET IMAGE OFF in effect, the editor inserts a tab character in the record, just as it inserts any other data character. It inserts no blanks.

To insert a tab character (X'05') into a record if SET IMAGE ON is in effect, enter SET IMAGE OFF before entering the line. Then use the Tab key as a character key; pressing the Tab key inserts a tab character in a line.

Setting Tabs

When you create a file, default logical tab settings are in effect, so you do not need to set tabs. To display the default tab settings, enter:

```
query tabs
```

To change the default tab settings, you can use the SET TABS subcommand. Then, regardless of the physical tab stops set up on your terminal, pressing the Tab key with SET IMAGE ON in effect spaces the data you enter to the columns you defined.

Note: If you enter one line with the INPUT subcommand and SET IMAGE ON is in effect, the line is placed 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 the first number specified in the SET TABS subcommand is the column where 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 OFF means backspace characters (and 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. For example, if you want an input line to look like this:

```
A̲B̲C̲
```

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:

```
A̲B̲C̲
```

To modify a line that has backspaces without typing all the characters again, change all the backspaces to some other character with the ALTER subcommand. The following sequence shows how to delete all the backspace characters in a line:

```
AAAAA
alter 16 + 1 *
```

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

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

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

```
AAAAA
```

Disconnect Mode Restrictions

The following is a list of some of the XEDIT subcommands, prefix subcommands, or macros not supported in disconnect mode.

Name	Type
BAckward	Subcommand
CURsor	Subcommand
FOrward	Subcommand
POWerinp	Subcommand
REFRESH	Subcommand
SOS	Subcommand
SI	Prefix macro
MODify	Macro
RGTLEFT	Macro
SCHANGE	Macro
SI	Macro
SPLTJOIN	Macro

Summary of XEDIT Subset

This table summarizes the subcommands in this chapter. Minimum abbreviations of subcommands are in uppercase letters.

Table 5. XEDIT Subcommand Summary

Function	Subcommand
To create or edit a file	**X**edit
To enter data	**I**nput
To display file lines	**T**ype
To move the line pointer	**D**own **U**p **TOP** **B**ottom
To move the column pointer	**CL**ocate **CF**irst
To locate data	**CL**ocate

Table 5. XEDIT Subcommand Summary (continued)	
Function	**Subcommand**
To make changes to the file	**C**hange **CI**nsert **CD**elete **CA**ppend
To recover deleted data	**REC**over
To delete lines	**DEL**ete
To replace a line	**R**eplace
To move lines	**MO**ve
To copy lines	**CO**py
To repeat a subcommand	**=**
To control special characters	**SET IM**age
To define logical tabs	**SET TABS**
To display tab settings	**Q**uery **TABS**
To display the logical record length	**Q**uery **LR**ecl
To alter special character	**AL**ter
To end an editing session without saving the changes	**QUIT**
To save automatically after changing a specified number of lines	**SET AU**tosave
To save the changed file when you have finished working on it	**FILE**
To store lines in temporary file for subsequent embed in another	**PUT**
To embed 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

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

What Is a Target?

Very simply, a target is a way you identify a line to the editor. You use targets 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

You can enter a target in the following ways:

- By itself
- As the operand of the LOCATE subcommand
- Before any XEDIT subcommand
- As the operand(s) in many other XEDIT subcommands

When you enter a target 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 you enter a target 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 *z/VM: XEDIT Commands and Macros Reference* for a complete description of the subcommand formats.

You can express a target 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

Look at Figure 21 on page 67. The purpose of Figure 21 on page 67 is to show you there are various ways to identify any given line to the editor. 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 the following targets are equivalent:

```
====>  :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 in the file identification line at the top of the screen, the "Line=" indicator shows 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 itself.
====> /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 itself.
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

You can specify the preceding targets 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

You can enter a target 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:

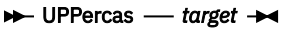
```
====>  :10 add 5
```

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 Enter, entering the subcommand, and pressing Enter. Typing both the target and the subcommand in the command line and pressing Enter only once saves you time.

Using a Target as a Subcommand Operand

When a subcommand format shows you can specify an operand as a target, the target usually tells 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, the format of the UPPERCAS subcommand is:



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

Figure 22 on page 69 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't say howdy,
00016 For I am a sufferer
00017 Magna cum laude.
00018 * * * End of File * * *




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

Figure 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 70 is a before-and-after example of a COUNT subcommand whose target operand is 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 you press Enter (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. A plus or minus sign preceding the number indicates

a forward (+) or backward (–) displacement from the current line. If the sign is omitted, a plus (+) is assumed.

A relative displacement can also be an asterisk (*), which means the Top of File (–*) or End of File (+* or *) line. When you specify an asterisk 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 72 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

You can assign any line in a file a name of one to eight characters preceded by a period (.), for example, .PART2.

You can use either the SET POINT subcommand or the *.xxxx* prefix subcommand to define a name for a line. The SET POINT subcommand defines a name of one to eight characters, preceded by a period, for the current line. The *.xxxx* prefix subcommand defines a name for any line; just enter a period followed by a one- to four-character name in the prefix area of the line.

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 you add or delete lines 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; you can then reference the line 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 you originally assigned it to the line; the editor always pays attention to uppercase and lowercase characters when looking for a line name.

Examples:

You can use 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)

You can also use the *.xxxx* prefix subcommand to name a line: type a name preceded by a period in the prefix area of any line on the screen:

```
=====   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 you press Enter, 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.

To use 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 74 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".

```
 TARGET5  SCRIPT   A1  V 132  Trunc=132 Size=13 Line=1 Col=1 Alt=0









===== * * * Top of File * * *
===== CELERY
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
=====
===== CELERY, RAW,
===== DEVELOPS THE JAW,
===== BUT CELERY, STEWED,
===== IS MORE QUIETLY CHEWED.
=====
===== THE PARSNIP
=====
===== THE PARSNIP, CHILDREN, I REPEAT,
====> DELETE .STOP
                                                          X E D I T  1 File
```

```
 TARGET5  SCRIPT   A1  V 132  Trunc=132 Size=6 Line=1 Col=1 Alt=1
7 line(s) deleted









===== * * * Top of File * * *
===== THE PARSNIP
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
=====
===== THE PARSNIP, CHILDREN, I REPEAT,
===== IS SIMPLY AN ANEMIC BEET.
===== SOME PEOPLE CALL THE PARSNIP EDIBLE;
===== MYSELF, I FIND THIS CLAIM INCREDIBLE.
===== * * * End of File * * *



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

Figure 25. A Target as a Line Name

A Target as a Simple String Expression

You can specify a target 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 you specify the string target 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 under “Specifying a Search Direction” on page 75.

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 previously, is one or more characters surrounded by delimiters. You can also:

- Determine the direction of the search
- Search for a line that does *not* contain a given string
- Search for any of several strings

Specifying a Search Direction

Type a plus (+) or minus (–) sign before a string target to 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 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 searching *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."

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 the target is a line that does *not* . contain the specified string. For example:

```
====>  ¬/Part Number/
```

(beginning with the line following the current line, locate a line that does not contain "Part Number" and make it the new current line)

```
====>  move 1 ¬/Part Number/
```

(move the current line after the first line that does not contain "Part Number")

Using an OR Symbol (|)

A string target can comprise 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 subcommand:

```
====>  locate /oranges/|/pears/|/peaches/
```

makes the following line current:

```
===== peaches
```

Using an AND Symbol (&)

You can use an *AND* symbol in the same way you use the *OR* symbol. The editor searches the file one line at a time and the first line that contains all 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/
```

makes the following line current:

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

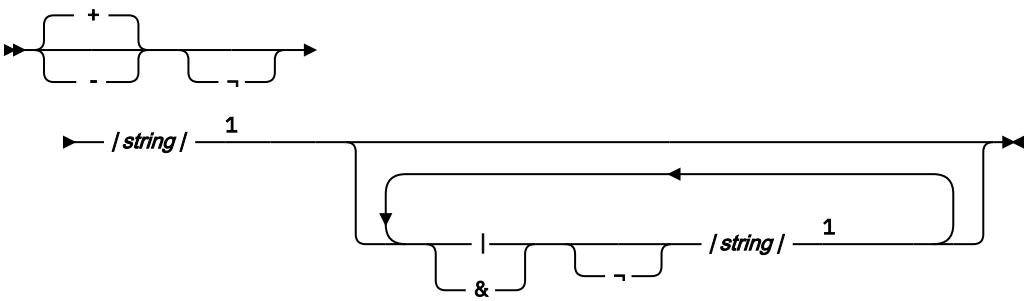
A Summary of Simple String Targets

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 further define a string.

You can also *combine* all these features to define a single target; that is, a plus or minus sign and a NOT symbol can precede a single string, enclosed in delimiters. OR or AND symbols can separate two or more strings.

Furthermore, if the subcommand SET HEX ON is in effect, you can specify a string in hexadecimal notation, for example, /X'C3D4E2'/.

The following is the format of a simple string expression:



Notes:

[1] The final delimiter (/) is optional after the last *string*. Leading or trailing blanks are considered part of *string*.

+ or -
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.

¬
NOT symbol (Locate something that is not the specified *string*.)

string
Character (or hexadecimal) string. The trailing delimiter may be necessary in certain circumstances. For example, if the first *string* has trailing blanks, use the trailing delimiter to indicate where the string ends.

&
AND symbol (ampersand) (Locate the line containing all the *string*s separated by an &.)

|
OR symbol (vertical bar) (Locate the line containing any of the *string*s, separated by OR symbols, starting with the first *string* specified.)

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 is in effect))

- 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 79 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. You can express a string 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 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 can span a specified number of lines.

- SET VARBLANK

  lets you control whether 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. The editor assigns each of the options an initial setting. You can alter the setting one or more times during an editing session by issuing the appropriate SET subcommand. (See *z/VM: XEDIT Commands and Macros 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.

Example:

```
====>  /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 respects or ignores the difference between uppercase and lowercase representations of alphabetic letters by using the SET CASE subcommand.

The following subcommand tells the editor uppercase and lowercase representations of the same letter do not match:

```
====>  set case mixed respect
```

For example, a file contains the following line:

```
===== The Text Editor
```

The following string target does *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 to be located. You can use the SET SPAN subcommand to specify 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 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 the number of blank characters between two words is significant in a target search.

SET VARBLANK ON means 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 preceding 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 81 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
```

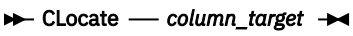
```
 TARGET7  SCRIPT   A1  V 132  Trunc=132 Size=19 Line=14 Col=1 Alt=0

===== 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 moves. However, the column pointer does not move. 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 locate all occurrences of a string throughout a file and move the column pointer. The format of the CLOCATE subcommand is:



where the *column_target* can be 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/
```

(search for "reporter", even if "clerk" occurs first; if "reporter" is not found, then search 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 CLOCATE can successively 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. Use the CLOCATE subcommand first to position the column pointer; then you can use another subcommand that operates based on the position of the column pointer.

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; blanks can replace characters.

These subcommands are discussed in detail in *z/VM: XEDIT Commands and Macros Reference*. Column-targets are discussed in that book in the CLOCATE subcommand section.

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)

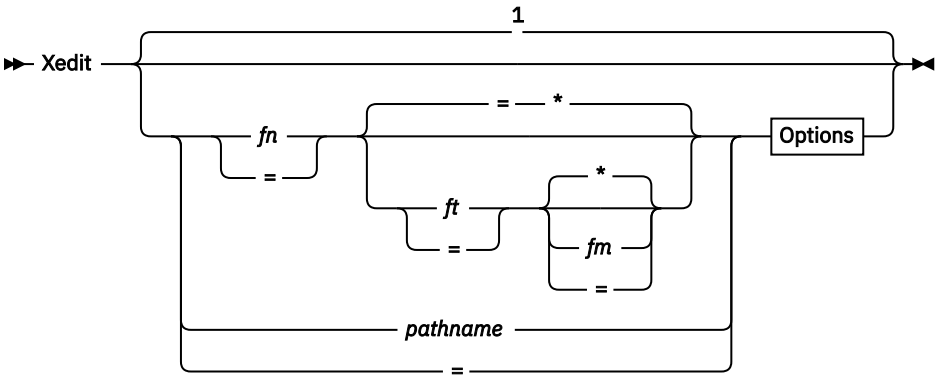
```
===== If anything can go, it will.
      <...+....1....+...|2....+....3....+....4....+....5....+....6....+....7...
```

Chapter 5. Editing Multiple Files

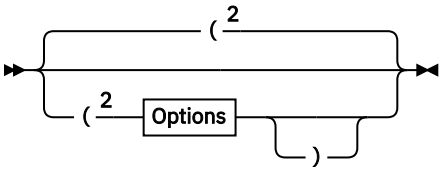
When you enter the CMS command XEDIT, a copy of the specified file is brought into virtual storage, where it remains until you enter a FILE or QUIT subcommand. In other words, the XEDIT *command* brings one file at a time into storage. By entering the XEDIT *subcommand* during an editing session, you can bring more than one file into virtual storage at a time.

The XEDIT Subcommand

The formats of the XEDIT subcommand and the XEDIT command are similar. Following is an example:



Options



Notes:

[1] If operands are not specified, the next file in the ring is chosen.

[2] See the XEDIT command in *z/VM: XEDIT Commands and Macros Reference* for the Options, Update Mode Options, and appropriate defaults.

For a complete description of the XEDIT subcommand operands, see *z/VM: XEDIT Commands and Macros 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, the file that is displayed.

A file remains in the ring until you enter a FILE or QUIT subcommand for that file; then the preceding file in the ring is displayed. Only your virtual storage size limits the number of files you can edit simultaneously.

Figure 28 on page 86 illustrates a ring of files in storage.

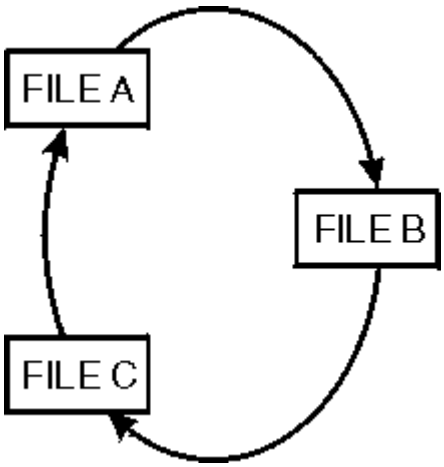


Figure 28. A Ring of Files in Storage

Note: The ring of files can be a mixture of CMS and BFS files.

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 87.) Therefore, entering a series of XEDIT subcommands 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 87.)
- Entering the XEDIT subcommand and specifying the file ID of a new or existing file that is not already in the ring adds that file to the ring just after the current file and displays it. (See Part 3 of Figure 29 on page 87.)

Ending an Editing Session

When you finish 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, as follows:

```
====>  cancel
```

Entering the CANCEL macro is equivalent to entering a QUIT subcommand for each file in the ring. If you modified any of the files, the usual warning message is displayed for each of those files:

```
File has been changed; type QQUIT to quit anyway
```

You can then enter either QQUIT or FILE.

If none of the files being canceled were modified, control returns immediately to CMS.

Multiple Logical Screens

Our discussion, up until now, has been on 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 *z/VM: XEDIT Commands and Macros Reference*.

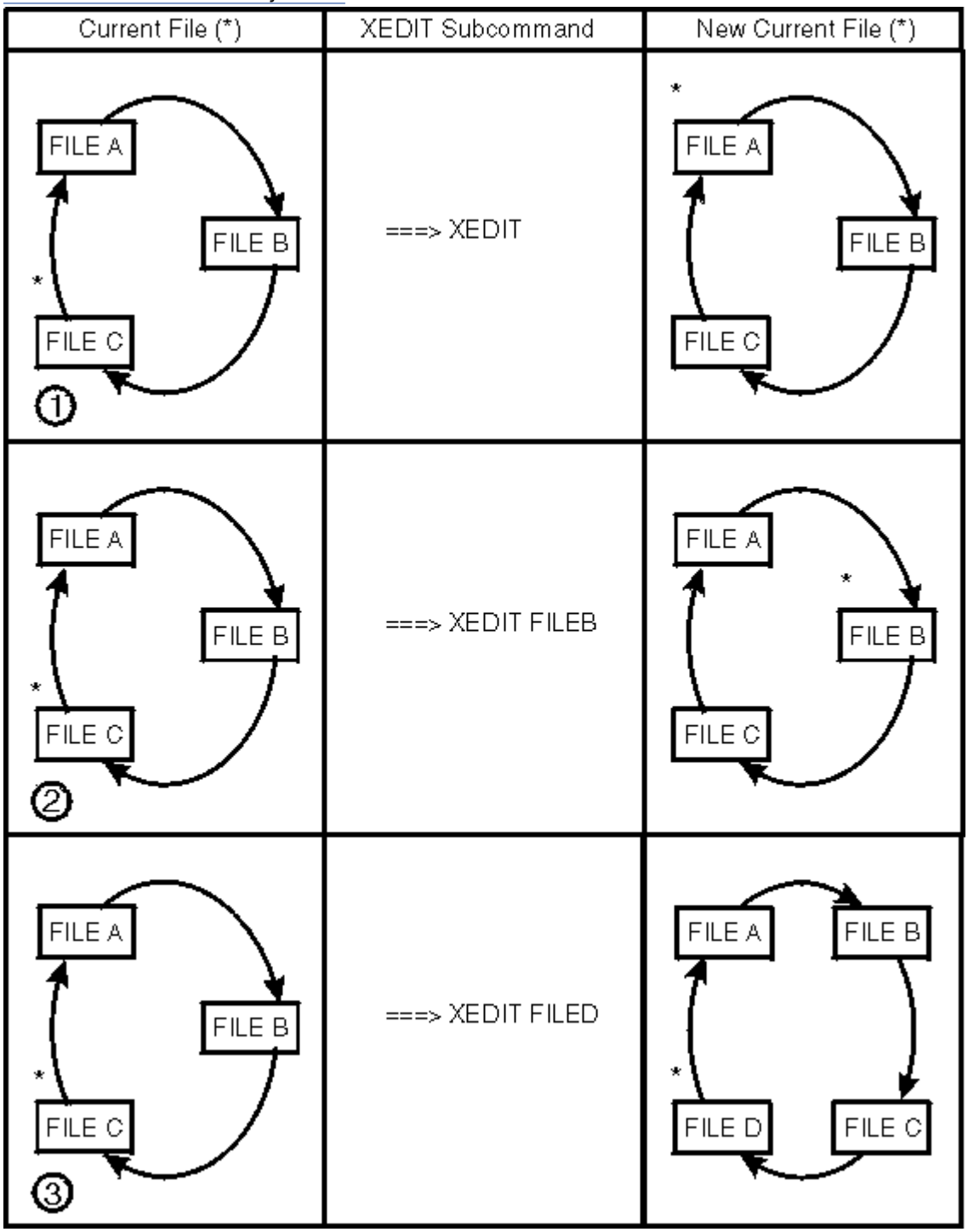


Figure 29. Editing Files in the Ring

SET SCREEN Subcommand

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

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

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

Likewise, the WIDTH option specifies the number of columns each vertical screen will contain. If you enter SET SCR WID 25 25 30, 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 *z/VM: XEDIT Commands and Macros 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 89 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 you press a PF key.

If any of the views are later used to display another file in the ring, the characteristics for that view of the original file are not saved unless it is the last view of the original file.

Multiple Views of Different Files

When you are editing multiple files and you enter a SET SCREEN subcommand that increases the number of logical screens, files from the ring immediately fill the additional screens.

Figure 31 on page 91 illustrates how files from the ring fill additional logical screens. 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 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 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 are 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 all messages from prefix subcommands and macros are displayed in the screen from which the pending list is 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 "Cursor Considerations" on page 90.

For more information on the pending list, see Chapter 7, "The Macrolanguage," on page 103.

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 the screens were defined in the SET SCREEN DEFINE subcommand.

Cursor Considerations

The cursor remains in the view that contained it when you pressed Enter (or a PA or PF key). 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), 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 *z/VM: XEDIT Commands and Macros Reference*.

```
 FILE1    SCRIPT   A1  V 132  Trunc=132 Size=7 Line=0 Col=1 Alt=0









===== * * * Top of File * * *
      |...+....1....+....2....+....3....+....4....+....5....+....6....+....7...
===== THE PANTHER
===== THE PANTHER IS LIKE A LEOPARD,
===== EXCEPT IT HASN'T BEEN PEPPERED.
===== SHOULD YOU BEHOLD A PANTHER CROUCH,
===== PREPARE TO SAY OUCH.
===== BETTER YET, IF CALLED BY A PANTHER,
===== DON'T ANTHER.
===== * * * End of File * * *

====> set screen 2 v
                                                            X E D I T  2 Files
```

```
 FILE1    SCRIPT   A1  V 132  Trunc=132  FILE2    SCRIPT   A1  V 132  Trunc=132









===== * * * Top of File * * *           ===== * * * Top of File * * *
      |...+....1....+....2....+....3...       |...+....1....+....2....+....3...
===== THE PANTHER                       ===== THE CANARY
===== THE PANTHER IS LIKE A LEOPARD,    ===== THE SONG OF CANARIES
===== EXCEPT IT HASN'T BEEN PEPPERED.   ===== NEVER VARIES.
===== SHOULD YOU BEHOLD A PANTHER CROUC ===== AND WHEN THEY'RE MOULTING
===== PREPARE TO SAY OUCH.              ===== THEY'RE PRETTY REVOLTING.
===== BETTER YET, IF CALLED BY A PANTHE ===== * * * End of File * * *
===== DON'T ANTHER.
===== * * * End of File * * *


====>                                   ====>
```

Figure 31. Multiple Vertical Views of Different Files

Chapter 6. Tailoring the Screen

This chapter explains how you can change your screen to suit your editing session.

Tailoring using SET subcommand options

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 *z/VM: XEDIT Commands and Macros 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 Enter or Clear 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 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 with the SET TABS subcommand). Initially, a tab line is not displayed. If you change the tab settings during an editing session, the tab line reflects that change, that is, the Ts are 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. Note this option may be specified as COLOR or COLOUR.

For a complete explanation of this function see *z/VM: XEDIT Commands and Macros 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 95 through Figure 37 on page 100 illustrates how some of the subcommands discussed above 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
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
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. SET MSGLINE on Multiple Lines with Overlay (Part 1 of 2)

```
 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 38. SET MSGLINE on Multiple Lines with Overlay (Part 2 of 2)

Chapter 7. The Macrolanguage

The macrolanguage is one of the most powerful facilities 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 *z/VM: REXX/VM User's Guide* and *z/VM: REXX/VM Reference* before you read this chapter.

What Is an XEDIT Macro?

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

Note: The XEDIT macro cannot be a BFS file.

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 Enter. You can execute a macro by entering only its name (or synonym), or its execution may also depend on arguments you enter when invoking the macro.

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, you can create it in any of the ways CMS provides for file creation. You can even create it dynamically, by using the XEDIT multiple file editing capability (see Chapter 5, “Editing Multiple Files,” on page 85). After you issue FILE or SAVE for the macro file, you can use the macro.

Like any CMS file, a macro file has a file name, file type, and file mode. The file identifier for a macro file must follow certain rules:

- For macros you enter from the command line, the file name is a string of 1-8 alphanumeric characters. This name invokes 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” on page 107.)

  Prefix macro file names can be 1-8 characters, but they cannot 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” on page 113, and the SET PREFIX subcommand description in *z/VM: XEDIT Commands and Macros Reference*.)
- The file type must be XEDIT.
- The file mode can specify any of your accessed disks or SFS 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 meaningful only in the context of a macro, for example, one that passes information to REXX.

The following list summarizes these subcommands; some are then discussed according to function. For detailed information on all these subcommands, see *z/VM: XEDIT Commands and Macros Reference*.

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

Communicating between the Editor and the Interpreter

The READ and EXTRACT subcommands supply a macro with information.

The READ subcommand finds 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, the macro cannot use it until it is taken *out of the console stack.* The REXX PULL instruction takes information out of the console stack and assigns it to program variables, which the macro can then examine.

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 the macro can then examine or use.

The following sections provide examples of using READ and EXTRACT.

READ Subcommand

When a macro issues a READ subcommand, the editor displays

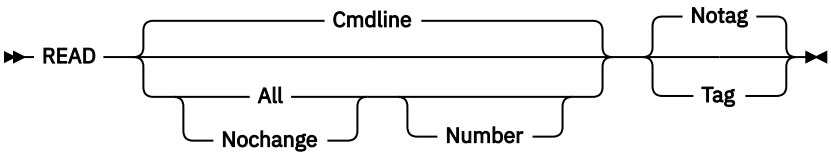
```
Macro-read
```

in the status area of your screen and waits for you to enter data or press a key. (The file image remains on the screen.) After you press a key, the data is placed in the console stack.

Operands of the READ subcommand specify how much information is placed in the console stack. The READ subcommand can place either the command line or all changed lines in the console stack. You can also specify inserting a tag identifying the origin of the line(s) at the beginning of each line stacked.

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

The READ subcommand has the following format:



Cmdline
indicates only the command line is stacked.

All
indicates anything changed on the screen is stacked.

Nochange
is similar to ALL, but the copy of the file in storage is not updated.

Number
indicates line numbers prefix changed file lines. Prefix area changes will be prefixed by their associated file line numbers only if the TAG option is specified.

Notag
indicates no tags are inserted.

Tag
indicates inserting a tag identifying the origin of changed lines at the beginning of each line stacked.

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

For example:

`MSG ENTER FILENAME FILETYPE FILEMODE`
Displays `ENTER FILENAME FILETYPE FILEMODE`.

`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 the SET subcommand defines) 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*; *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, to get information about the case setting, a macro could 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
```

MSG, EMSG, and CMSG Subcommands

A macro can communicate with the user by displaying messages in the message line of the screen. Messages have various purposes, for example, requesting the user to enter data, telling a user an error has occurred during processing, and so forth.

The following two subcommands display a message in the message line:

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 clears the screen before displaying the data.

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:

```
CMSG
```

When issued from a macro, the CMSG subcommand can redisplay input the user has entered incorrectly so the user can correct and reenter it.

SET MSGMODE Subcommand

The SET MSGMODE subcommand controls whether 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 displayed.

SET RESERVED Subcommand

When issued from a macro, the SET RESERVED subcommand reserves a specified line on the screen for the macro's use, 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 your terminal supports, you can display the line 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 specifies attributes for fields within a reserved line. Depending on the features your terminal supports, you can display these fields 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 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.'
```

SUPERSET Subcommand

The SUPERSET subcommand lets you enter multiple SET options on one subcommand to improve performance. You will save time by using SUPERSET with any of the SET options. For example, the following subcommand:

```
superset /scale on/autosave 5/case mixed/cmdline top/
```

quickly sets your scale on, issues the SAVE subcommand after 5 changes to the file, lets you type in mixed case, and moves your command line to the top of the file.

CURSOR Subcommand

The CURSOR subcommand moves the cursor to a specified position on the screen, and, optionally, assigns 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” on page 118.

Saving and Restoring Editing Variables

The PRESERVE subcommand saves the settings of various editing variables until a subsequent RESTORE subcommand. For example, you might want to preserve a setting so 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, see the PRESERVE subcommand description in *z/VM: XEDIT Commands and Macros 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
```

(The REXX instructions ADDRESS CMS and ADDRESS COMMAND can also issue CMS and CP commands.)

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 because of 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 COMMAND subcommand causes 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 in the following).

For example:

```
COMMAND PRESERVE
```

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

Similarly, the MACRO subcommand causes 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.)

You can use the MACRO subcommand 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, the editor interprets both NEXT8 and N8 as 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 named N8 would then be executed.

You can use the SET MACRO subcommand 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 specifies whether the editor looks for synonyms.

Walking through an XEDIT Macro

The following XEDIT macro (Figure 39 on page 109) 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, with 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 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 REXX statements in the macro, see *z/VM: REXX/VM 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 39. Sample Macro

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

00001-00003 /* Do a global change on imbedded Script files */
REXX 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 ends, 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 nonzero 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 with the FIND subcommand in line 16; this point is only reached 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 is reset to the return code for that FIND subcommand just before returning to this point to execute the statements again. When the return code is not 0, this macro continues 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 preceding EXTRACT subcommand). In this case the current line is the .im statement that was found through "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 quotation marks 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 substitutes 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.

00031 xedit fname 'SCRIPT (NOPROFILE'
This statement invokes the editor (XEDIT) for the file specified. REXX substitutes 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 you issue an XEDIT command (or subcommand). You can write a PROFILE XEDIT macro, file it, and forget about it. It is executed before each file is brought into storage.

Note: The XEDIT macro cannot be a BFS file.

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 labeled MYPROF XEDIT is executed, even if a macro labeled 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 REXX 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 *only* a profile macro can issue. 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 (except for PROFILE and NOPROF, which are ignored), can supply editing options not specified in the XEDIT command itself.

Within the profile macro, the LOAD subcommand must be the first XEDIT subcommand. If it is not, the editor automatically issues a LOAD subcommand; its operands are the same as those issued in the XEDIT command. (REXX statements and CMS commands can be issued before the LOAD.)

The profile macro can prompt the user for XEDIT command options or assign values to editing variables before issuing the LOAD subcommand. For example, a SCRIPT user might program his or her profile to issue a LOAD subcommand specifying a default file type.

The options specified in the LOAD subcommand have a lower priority than those specified in an XEDIT command. For example, a NOUPDATE option in the XEDIT command overrides an UPDATE option specified in the LOAD subcommand.

When the LOAD subcommand is executed, the file is brought into virtual storage.

If the LOAD is successful, you will receive a return code of 0 or 3. If LOAD completes with any other return code, all subsequent subcommands in the profile macro are rejected with a unique return code of 6, and the editor automatically issues a quit subcommand.

For detailed information on the LOAD subcommand, see *z/VM: XEDIT Commands and Macros Reference*.

An Example of a Profile Macro

An example of a profile macro is shown in Figure 40 on page 113.

```
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 40. A PROFILE XEDIT Macro

00001 /* Sample XEDIT profile */
Identifies the macro as a REXX file.

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

00003 if ft= '' then ft= 'SCRIPT'
If no file type is assigned, this assigns 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 XEDIT to recognize 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 usually done 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 is usually done 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. You can specify a number 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 (in the preceding macro) 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 makes 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. To set the current line, you must override the automatic current line restoration by issuing the subcommand:

```
set pending on /
```

Note: When the RESET subcommand is executed to clear the pending list, any pending prefix macros are invoked with the CLEAR argument. In this case, the prefix macro may issue subcommands to specify which line is current when it is finished executing, that is, the current line is not automatically returned to the line that was current when the prefix macros on the pending list were invoked.

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. This is usually done 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, the source string is used 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 *z/VM: REXX/VM Reference*.

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. See "Examining the Argument String" on page 117.

In this example, if the user entered *L8* 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).

```
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 up to five characters long.
- The synonym can be a special character 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 *z/VM: XEDIT Commands and Macros 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 *z/VM: XEDIT Commands and Macros Reference*.

Using the Pending List

You have seen 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 nonzero return code, execution of the pending list stops and all entries not executed remain pending, until the user presses Enter (or a 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). See *z/VM: XEDIT Commands and Macros 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 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 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 this is a prefix call.

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

SHADOW
indicates a prefix macro was entered on a shadow line (see SET SHADOW in *z/VM: XEDIT Commands and Macros Reference*).

CLEAR
indicates 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.

op
is the optional operand(s) of the macro, entered either to its left or right (for example, 5M or M5). See *z/VM: XEDIT Commands and Macros Reference* chapter that describes "Prefix Subcommands and Macros" for rules for the recognition of operands.

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 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 Enter 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 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 CURSOR subcommand and *z/VM: XEDIT Commands and Macros Reference* chapter that describes "Prefix Subcommands and Macros".

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

Decoding the Prefix Area

See the *z/VM: XEDIT Commands and Macros 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

Also see Figure 41 on page 120 for a sample prefix macro.

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 41. Sample Prefix Macro

Appendix A. Summary of XEDIT Subcommands and Macros

These subcommands are described in detail in *z/VM: XEDIT Commands and Macros Reference*.

Table 6. XEDIT Subcommand Summary

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 the 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 zone specified.
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 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.
EXPand	Repositions data according to new tab settings.

Table 6. XEDIT Subcommand Summary (continued)	
Subcommand	**Purpose**
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 z/VM 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.
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.

Table 6. XEDIT Subcommand Summary (continued)	
Subcommand	**Purpose**
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 re-executes 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 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 BFSLine	Controls the translation between records and byte stream for BFS files.
SET BRKkey	Specifies whether 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).
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 lets you enter a subcommand while in input mode (typewriter terminals only).

Table 6. XEDIT Subcommand Summary (continued)	
Subcommand	**Purpose**
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	Controls whether the 3270 null character is recognized in the middle of the 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 NAMetype	File IDs are in CMS format (*fn ft fm*) or in BFS format (*pathname*).
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.
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 SFS 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 PName	Changes the BFS path name of the current file.
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.

Table 6. XEDIT Subcommand Summary (continued)	
Subcommand	**Purpose**
SET RANge	Controls limits for line-pointer movement.
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 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.
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 backward searches) is reached during a search.
SET Zone	Defines new limits within each line for target searches.
SET =	Inserts string into the equal buffer.

Table 6. XEDIT Subcommand Summary (continued)	
Subcommand	**Purpose**
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.
SUPerset	Specifies multiple SET options on one subcommand to improve performance.
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.
=	Re-executes the last subcommand, macro, or CP/CMS command.
?	Displays the last subcommand, macro, or CP/CMS command executed.

Table 7. Prefix Subcommand Summary	
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.

Table 7. Prefix Subcommand Summary (continued)	
Prefix	**Subcommands**
.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.

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information may contain sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., in the United States and/or other countries. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on IBM Copyright and trademark information (https://www.ibm.com/legal/copytrade).

Adobe is either a registered trademark or trademark of Adobe Systems Incorporated in the United States, and/or other countries.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Terms and Conditions for Product Documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal Use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial Use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see:

- The section entitled **IBM Websites** at IBM Privacy Statement (https://www.ibm.com/privacy)
- Cookies and Similar Technologies (https://www.ibm.com/privacy#Cookies_and_Similar_Technologies)

Acknowledgements

IBM gratefully acknowledges 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.

Bibliography

This topic lists the publications in the z/VM library. For abstracts of the z/VM publications, see *z/VM: General Information*.

Where to Get z/VM Information

The current z/VM product documentation is available in IBM Documentation - z/VM (https://www.ibm.com/docs/en/zvm).

z/VM Base Library

Overview

- *z/VM: License Information*, GI13-4377
- *z/VM: General Information*, GC24-6286

Installation, Migration, and Service

- *z/VM: Installation Guide*, GC24-6292
- *z/VM: Migration Guide*, GC24-6294
- *z/VM: Service Guide*, GC24-6325
- *z/VM: VMSES/E Introduction and Reference*, GC24-6336

Planning and Administration

- *z/VM: CMS File Pool Planning, Administration, and Operation*, SC24-6261
- *z/VM: CMS Planning and Administration*, SC24-6264
- *z/VM: Connectivity*, SC24-6267
- *z/VM: CP Planning and Administration*, SC24-6271
- *z/VM: Getting Started with Linux on IBM Z*, SC24-6287
- *z/VM: Group Control System*, SC24-6289
- *z/VM: I/O Configuration*, SC24-6291
- *z/VM: Running Guest Operating Systems*, SC24-6321
- *z/VM: Saved Segments Planning and Administration*, SC24-6322
- *z/VM: Secure Configuration Guide*, SC24-6323

Customization and Tuning

- *z/VM: CP Exit Customization*, SC24-6269
- *z/VM: Performance*, SC24-6301

Operation and Use

- *z/VM: CMS Commands and Utilities Reference*, SC24-6260
- *z/VM: CMS Primer*, SC24-6265
- *z/VM: CMS User's Guide*, SC24-6266
- *z/VM: CP Commands and Utilities Reference*, SC24-6268

- *z/VM: System Operation*, SC24-6326
- *z/VM: Virtual Machine Operation*, SC24-6334
- *z/VM: XEDIT Commands and Macros Reference*, SC24-6337
- *z/VM: XEDIT User's Guide*, SC24-6338

Application Programming

- *z/VM: CMS Application Development Guide*, SC24-6256
- *z/VM: CMS Application Development Guide for Assembler*, SC24-6257
- *z/VM: CMS Application Multitasking*, SC24-6258
- *z/VM: CMS Callable Services Reference*, SC24-6259
- *z/VM: CMS Macros and Functions Reference*, SC24-6262
- *z/VM: CMS Pipelines User's Guide and Reference*, SC24-6252
- *z/VM: CP Programming Services*, SC24-6272
- *z/VM: CPI Communications User's Guide*, SC24-6273
- *z/VM: ESA/XC Principles of Operation*, SC24-6285
- *z/VM: Language Environment User's Guide*, SC24-6293
- *z/VM: OpenExtensions Advanced Application Programming Tools*, SC24-6295
- *z/VM: OpenExtensions Callable Services Reference*, SC24-6296
- *z/VM: OpenExtensions Commands Reference*, SC24-6297
- *z/VM: OpenExtensions POSIX Conformance Document*, GC24-6298
- *z/VM: OpenExtensions User's Guide*, SC24-6299
- *z/VM: Program Management Binder for CMS*, SC24-6304
- *z/VM: Reusable Server Kernel Programmer's Guide and Reference*, SC24-6313
- *z/VM: REXX/VM Reference*, SC24-6314
- *z/VM: REXX/VM User's Guide*, SC24-6315
- *z/VM: Systems Management Application Programming*, SC24-6327
- *z/VM: z/Architecture Extended Configuration (z/XC) Principles of Operation*, SC27-4940

Diagnosis

- *z/VM: CMS and REXX/VM Messages and Codes*, GC24-6255
- *z/VM: CP Messages and Codes*, GC24-6270
- *z/VM: Diagnosis Guide*, GC24-6280
- *z/VM: Dump Viewing Facility*, GC24-6284
- *z/VM: Other Components Messages and Codes*, GC24-6300
- *z/VM: VM Dump Tool*, GC24-6335

z/VM Facilities and Features

Data Facility Storage Management Subsystem for z/VM

- *z/VM: DFSMS/VM Customization*, SC24-6274
- *z/VM: DFSMS/VM Diagnosis Guide*, GC24-6275
- *z/VM: DFSMS/VM Messages and Codes*, GC24-6276
- *z/VM: DFSMS/VM Planning Guide*, SC24-6277

- *z/VM: DFSMS/VM Removable Media Services*, SC24-6278
- *z/VM: DFSMS/VM Storage Administration*, SC24-6279

Directory Maintenance Facility for z/VM

- *z/VM: Directory Maintenance Facility Commands Reference*, SC24-6281
- *z/VM: Directory Maintenance Facility Messages*, GC24-6282
- *z/VM: Directory Maintenance Facility Tailoring and Administration Guide*, SC24-6283

Open Systems Adapter

- Open Systems Adapter/Support Facility on the Hardware Management Console (https://www.ibm.com/docs/en/SSLTBW_2.3.0/pdf/SC14-7580-02.pdf), SC14-7580
- Open Systems Adapter-Express ICC 3215 Support (https://www.ibm.com/docs/en/zos/2.3.0?topic=osa-icc-3215-support), SA23-2247
- Open Systems Adapter Integrated Console Controller User's Guide (https://www.ibm.com/docs/en/SSLTBW_2.3.0/pdf/SC27-9003-02.pdf), SC27-9003
- Open Systems Adapter-Express Customer's Guide and Reference (https://www.ibm.com/docs/en/SSLTBW_2.3.0/pdf/ioa2z1f0.pdf), SA22-7935

Performance Toolkit for z/VM

- *z/VM: Performance Toolkit Guide*, SC24-6302
- *z/VM: Performance Toolkit Reference*, SC24-6303

The following publications contain sections that provide information about z/VM Performance Data Pump, which is licensed with Performance Toolkit for z/VM.

- *z/VM: Performance*, SC24-6301. See z/VM Performance Data Pump.
- *z/VM: Other Components Messages and Codes*, GC24-6300. See Data Pump Messages.

RACF® Security Server for z/VM

- *z/VM: RACF Security Server Auditor's Guide*, SC24-6305
- *z/VM: RACF Security Server Command Language Reference*, SC24-6306
- *z/VM: RACF Security Server Diagnosis Guide*, GC24-6307
- *z/VM: RACF Security Server General User's Guide*, SC24-6308
- *z/VM: RACF Security Server Macros and Interfaces*, SC24-6309
- *z/VM: RACF Security Server Messages and Codes*, GC24-6310
- *z/VM: RACF Security Server Security Administrator's Guide*, SC24-6311
- *z/VM: RACF Security Server System Programmer's Guide*, SC24-6312
- *z/VM: Security Server RACROUTE Macro Reference*, SC24-6324

Remote Spooling Communications Subsystem Networking for z/VM

- *z/VM: RSCS Networking Diagnosis*, GC24-6316
- *z/VM: RSCS Networking Exit Customization*, SC24-6317
- *z/VM: RSCS Networking Messages and Codes*, GC24-6318
- *z/VM: RSCS Networking Operation and Use*, SC24-6319
- *z/VM: RSCS Networking Planning and Configuration*, SC24-6320

TCP/IP for z/VM

- *z/VM: TCP/IP Diagnosis Guide*, GC24-6328
- *z/VM: TCP/IP LDAP Administration Guide*, SC24-6329
- *z/VM: TCP/IP Messages and Codes*, GC24-6330
- *z/VM: TCP/IP Planning and Customization*, SC24-6331
- *z/VM: TCP/IP Programmer's Reference*, SC24-6332
- *z/VM: TCP/IP User's Guide*, SC24-6333

Prerequisite Products

Device Support Facilities

- Device Support Facilities (ICKDSF): User's Guide and Reference (https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/ickug00_v2r5.pdf), GC35-0033

Related Products

XL C++ for z/VM

- *XL C/C++ for z/VM: Runtime Library Reference*, SC09-7624
- *XL C/C++ for z/VM: User's Guide*, SC09-7625

z/OS

IBM Documentation - z/OS (https://www.ibm.com/docs/en/zos)

Index

Special Characters

? subcommand 11
..... (pending...) 15
.xxxx (prefix subcommand) 73
" (prefix subcommand) 17
"" (pending...) 17
/ prefix subcommand
    practice exercise using 40
& (symbol used in string target) 76
# (as default line-end character) 6, 7
< SHIFT LEFT prefix macro 35
= subcommand
    in typewriter mode 53
> SHIFT RIGHT prefix macro 35
¬ (symbol)
    used in target string 75
| *symbol used in string target) 76
$ (as arbitrary character) 78

A

A prefix subcommand
    practice exercise using 42
absolute column number 25
add
    lines
        continuously 14
        of indented text 14
        using A (prefix subcommand) 12
        using SI 14
    subcommands 103
    text
        in typewriter mode 51
        to end of line 83
alarm
    sounding 106
ALL macro 34
Alt= 3
alter
    a character 34
    in typewriter mode 61
ALTER macro 34
alteration count 3, 29, 57
AND symbol used in string target 76
append
    in typewriter mode 51
    text to line 83
arbitrary character 78
argument string
    passed to prefix macro 114, 117
assign
    name to a line 72
automatic
    line wrapping 25
    save
        in typewriter mode 57

B

backspace
    characters in typewriter mode 60, 61
backward
    search 22, 75, 77
BACKWARD subcommand 20
BFS (byte file system) 1, 2, 47
blank
    characters
        targets, significance 80
    line
        in typewriter mode 53
        insert 12
block
    lines
        accepted by prefix macros 115
        copying 18
        deleting 12
        duplicating 17
        moving 18
BOTTOM subcommand
    in typewriter mode 50
    practice exercise using 40
break in the data, causing 6
bypassing
    profile macro 111

C

C or CC pending... 18
C prefix subcommand 18
CANCEL macro 86
CAPPEND macro
    in typewriter mode 51
case
    changing 35
    specifying 78
CDELETE subcommand
    in typewriter mode 51
CFIRST subcommand
    in typewriter mode 50
change
    definition of a character
        in typewriter mode 61
    screen layout 93
    tab settings
        in typewriter mode 60
change data
    from multiple views of the same file 88
    globally
        in typewriter mode 52
    position 34
    selectively
        in typewriter mode 53
    using CHANGE
        in typewriter mode 52

change data *(continued)*
using COVERLAY 83
using CREPLACE 83
CHANGE subcommand
in typewriter mode 52
practice exercise using 43
with absolute line number as target 70
character
delete using CDELETE 83
insert using CINSERT 25, 83
overlay using COVERLAY 83
replace using CREPLACE 83
CINSERT subcommand
in typewriter mode 51
CLAST subcommand 83
CLEAR key to remove prefix subcommands 20
CLOCATE subcommand
in typewriter mode 50
CMS
commands issued from a macro 107
CMS subcommand 107
CMSG subcommand 106
Col= 2
color
defining
with SET COLOR 94
with SET CTLCHAR 106
with SET RESERVED 106
column
specified for viewing 35
column pointer
defined 3
displayed in the scale 3
displayed on typewriter terminal 48
indicator in file identification line 2
list of subcommands operating based on 82
moving
in typewriter mode 50
to beginning of line 27, 83
to end of line 83
using CLOCATE 22
resetting
in typewriter mode 50
subcommands based on position 50, 82
column-target 81
combine
files
in typewriter mode 57
SET options 80
command
input area 4
command line
changing location
example 96
defining display features 94
displaying message in 106
location on screen 4
COMMAND subcommand 107
commands issued from a macro 107
complex string expression as target
example 81
COMPRESS subcommand 34
concatenate
files
concatenate *(continued)*
files *(continued)*
in typewriter mode 57
console
stack 104
copy lines
from another file
using GET on display terminal 29
using GET on typewriter terminal 57
in the same file
in typewriter mode, using COPY 56
using C prefix subcommand 18
under original lines, using " 17
COPY subcommand
in typewriter mode 56
COUNT subcommand
example 70
COVERLAY subcommand 83
CP
commands issued from a macro 107
subcommand 107
create
a file
in typewriter mode 47
macro file 103
practice exercise 39
CREPLACE subcommand 83
current column
in typewriter mode 48
current line
append words to
in typewriter mode 51
as starting place for subcommands 3
change
using / 19
using a target 66
using CLOCATE 22
using DOWN 20
using UP 21
changing location on screen
example 97
defining display features 94
displaying in typewriter mode 49
indicator in file identification line 2
location on screen 3
replacing, in typewriter mode 55
using target as displacement from 70
cursor
placement in multiple screens 90
CURSOR subcommand 107
cursor, move
by priority in prefix macros 118
to command line 4
to specified location 107

D

D or DD pending... 12
D prefix subcommand
practice exercise using 42
data, changing
globally
in typewriter mode 52
selectively

data, changing *(continued)*
selectively *(continued)*
in typewriter mode 53
using CHANGE
in typewriter mode 52
using COVERLAY 83
using CREPLACE 83
data, entering
on display terminal 4
on typewriter terminal 48
data, locating
using a target 74
using CLOCATE
in typewriter mode 50
define
screen size 88
delete
characters 83
in typewriter mode 51
lines
block of lines 12
in typewriter mode 54
recovering 13
using D prefix subcommand 12
DELETE subcommand
in typewriter mode 54
delimiters
in typewriter mode 50
use of 22
destination line
F prefix subcommand 17
for copied lines 18
for moved lines 17
P prefix subcommand 17
disconnect mode
restrictions 62
display
data from a macro 105
features 94
help menus 34
in typewriter mode 61
line numbers on screen 31
lines on typewriter terminal 48
messages on editor screen 105
more than one file 88
screen layout 2
tab settings 27
divide
screen 87
DOWN subcommand
in typewriter mode 49
practice exercise using 40
duplicate
lines 17

E

edit
defined 1
environment 1
illustration 87
in typewriter mode 47
mode 4
multiple files 85
edit *(continued)*
one file 1
edit variables
preserving 107
restoring 107
transferring 105
editor
invoke
in typewriter mode 47
EMSG subcommand 106
end an editing session
in typewriter mode 56
enter
data 4
in typewriter mode 47
prefix subcommands 3, 11
subcommands 4
using INPUT 4
using POWERINP 6
XEDIT subcommands
on typewriter terminal 47
error message
display
in typewriter mode 47
example 21
EXEC 2
file used as XEDIT macro 103
execute
subcommand 4, 11
exercises, practice 39
exit the editor
in typewriter mode 56
EXPAND subcommand 34
extended highlighting
define
with SET COLOR 94
with SET CTLCHAR 106
with SET RESERVED 106
EXTRACT subcommand 104, 105

F

F pending... 18
F prefix subcommand 17
file
area on screen 3
identification line 2
insert 29
file mode
XEDIT macro 103
file name
XEDIT macro 103
XEDIT prefix macro 103
FILE subcommand
in typewriter mode 56
practice exercise using 39
file type
XEDIT macro 103
find, data
in typewriter mode 50
using a target 74
using CLOCATE 22
forward
search 75

FORWARD subcommand 20
full-screen mode 1

G

GET subcommand
  in typewriter mode 57
  practice exercise using 44
global changes
  in typewriter mode 52

H

HELP display 34
HELP macro 34
highlighting
  define
    with SET COLOR 94
    with SET CTLCHAR 106
    with SET RESERVED 106
horizontal screen
  multiple 87

I

imbed all or part of one file in another
  using GET
    in typewriter mode 57
information message display
  in typewriter mode 47
initial setting
  of PF keys 8
INPUT line 53
input mode
  on typewriter terminal 48
INPUT subcommand
  practice exercise using 39
  to enter input mode
    on typewriter terminal 48
  to enter line in typewriter mode 53
input zone
  changing size 93
insert
  a blank line 12
  from another file
    in typewriter mode 57
  in input mode 11
  in power typing mode 7, 11
  lines using INPUT
    in typewriter mode 48
  mode key in XEDIT 7, 11
  part of
    example 33
  using CINSERT
    in typewriter mode 51
  using PA2 key 11
  using SET NULLS 11
  using the insert mode key 11
  whole file
    example 30
  word
    in typewriter mode 51
    using CINSERT 25, 83
insert mode key in XEDIT
  practice exercise using 40
invoke
  editor
    in typewriter mode 47

J

join
  files 29
  lines 9

L

label a line 72
LEFT subcommand 34
line name, target as
  example 73
line number
  displaying 31
line pointer
  in typewriter mode 49
  moved by target 66
line wrapping, automatic 25
line-end character 6
Line= 2
LOAD subcommand 112
locate
  data
    in typewriter mode 50
    using a target 74
    using CLOCATE 22
LOCATE subcommand 67
logical record length 48, 49
logical screen
  multiple 87
LOWERCAS subcommand 34
LPREFIX subcommand
  in typewriter mode 56

M

M or MM pending... 18
M prefix subcommand
  practice exercise using 42
MACRO subcommand 107
macrolanguage 103
macros in XEDIT
  argument string 117
  avoiding name conflicts 107
  creating 103, 113
  cursor position 118
  definition of 103
  examples
    L Prefix macro 120
    prefix macro 114
    profile macro 113
  executing 103
  file identifier 103
  handling blocks 115
  information passed
    to prefix macro 114
    using information 115

macros in XEDIT *(continued)*
information passed *(continued)*
with READ 104
prefix 113
profile 111
sample 109
search order specified 108
source string 115
subcommands used in 104
XEDIT prefix macro 119
MERGE subcommand 34
message examples, notation used in xiv
message line
changing location 93
defining display features 94
location on screen 3
messages
controlling display of 106
displaying in command line 106
displaying on editor screen 105
error 3
in typewriter mode 47
information 3
issued from a macro 105
warning, example 28
modify
tab settings 27
MODIFY TABS 27
move
cursor
to specified location 107
cursor to command line 4
display right or left 34, 35
line
in typewriter mode 55
using M prefix subcommand 17
line pointer
using BOTTOM 20
using DOWN 20
using TOP 20
using UP 20
lines 55
MOVE subcommand
in typewriter mode 55
move through a file
using BACKWARD 20
using BOTTOM
in typewriter mode 50
using DOWN
in typewriter mode 49
using FORWARD 20
using PF keys 10
using TOP
in typewriter mode 50
using UP
in typewriter mode 49
MSG subcommand 106
multiple files
displaying 85
editing
illustration 87
ending editing sessions for 86
on one screen
example 91
multiple logical screens
defining 87
example 89
multiple views
of different files
example 91
of same file
example 89
making changes in 88
order of processing in 89

N

name
avoiding conflicts of macro 107
line 35, 72
NOT symbol used in string target 75
notation used in message and response examples xiv
NULLKEY 11
number of files being edited 4

O

operand
target as 68
OR symbol used in string target 76
order of processing with multiple screens 89

P

P pending... 18
P prefix subcommand
practice exercise using 42
PA2 key
practice exercise using 40
path name, BFS 1, 2, 47
pending list 89, 116
pending notice
..... pending 15
"" pending 17
C or CC pending... 18
canceling 19
DD pending 12
defining display features 94
F pending... 18
location on screen 4
M or MM pending... 18
P pending... 18
PF keys
changing settings of 9
displaying settings of 8
initial settings of 8, 36, 37
using 8
pound sign 6
power typing mode
example 7
inserting characters in 7
practice exercise using 40
typing data in 4, 6
using line-end character in 6
POWERINP subcommand
practice exercise using 40
practice exercise 39

prefix area
changing location or display
example 95
defining display features 94
location on screen 3
simulate in typewriter mode 56
prefix macros
assigning a synonym 116
examples 114
writing 113
prefix subcommands
.xxxx 73
/ 19
A
example 13
C 18
canceling 19
D
example 13
defined 11
F
example 19
list of 36
M
example 19
P 17
practice exercise using 42
SI
example 14
where to enter 3
preserve editing variables 107
PRESERVE subcommand 107
processing with multiple screens, order of 89
profile macro in XEDIT
definition of 111
example 113
programmed symbol set
define
with SET COLOR 94
with SET CTLCHAR 106
with SET RESERVED 106
PUT subcommand
in typewriter mode 58
practice exercise using 44

Q

QQUIT subcommand
in typewriter mode 57
QUERY LRECL subcommand 49
QUERY PF subcommand 8
QUERY POINT subcommand 73
QUERY RING 86
QUERY TABS subcommand
in typewriter mode 61
practice exercise using 39
QUIT subcommand
in typewriter mode 57

R

range of operation of subcommands, defining 68
READ subcommand 104
record format 2
record length
in typewriter mode 48, 49
recover
deleted lines
in typewriter mode 54
RECOVER subcommand
example 14
in typewriter mode 54
practice exercise using 42
redefine a character
in typewriter mode 61
redisplay
subcommand 11
reexecuting a subcommand 11
refer to a line 72
relative displacement, target as
example 71
repeat
the display of a subcommand 11
the execution of a subcommand 11
REPLACE subcommand
in typewriter mode 55
replace, data
globally
in typewriter mode 52
selectively
in typewriter mode 53
using CHANGE
in typewriter mode 52
using COVERLAY 83
using CREPLACE 83
reposition data 34
RESET key
to end insert mode 7
RESET subcommand 19
response examples, notation used in xiv
restore
editing variables 107
RESTORE subcommand 107
restriction
disconnect mode 62
REXX
file used as XEDIT macro 103
RIGHT subcommand 34
ring of files
editing 86
illustration of 85

S

save
automatic 29
editing variables 107
scale
defining display features 94
location on screen 3
screen
changing 93
layout 2
size, defining 88
scroll
using BACKWARD 20
using FORWARD 20

scroll *(continued)*
using PF keys 10
search
order
macros and subcommands specified 108
search direction specified 75
search, data
using a target 74
using CLOCATE
in typewriter mode 50
selective change
example 24
in typewriter mode 53
SET ARBCHAR subcommand 35, 78
SET AUTOSAVE subcommand
in typewriter mode 57
practice exercise using 39
SET CASE subcommand 35, 79
SET CMDLINE subcommand
example 96
SET COLOR subcommand 94
SET CTLCHAR subcommand 106
SET CURLINE subcommand
example 97
SET HEX subcommand 77
SET IMAGE subcommand
in typewriter mode 60, 61
SET MACRO subcommand 107
SET MSGLINE subcommand
example 99
SET MSGMODE subcommand 106
SET NULLS subcommand 11
SET NUMBER subcommand
description 94
to determine absolute line number 31, 69
SET options, combining 80
SET PFn subcommand 8
SET POINT subcommand 35, 72
SET PREFIX subcommand
example 95
SET RESERVED subcommand 106
SET SCALE subcommand
example 98
SET SCREEN subcommand
example 89, 91
SET SPAN subcommand 78, 80
SET SYNONYM subcommand 107
SET TABLINE subcommand
example 99
SET TABS subcommand
in typewriter mode 61
practice exercise using 39
SET VARBLANK subcommand 78, 80
SET VERIFY subcommand 35
shift
display right or left 35
SI prefix subcommand 14
simple string expression as target
example 78
format 76
size
logical screen 87
of file 2
Size= 2
sort 35
SORT macro 35
source string
passed to prefix macro 115
span
lines 80
special characters in typewriter mode
altering 61
using 60
split
lines 9
screen 87
status
..... pending 15
"" pending 17
C or CC pending... 18
DD pending 12
F pending... 18
M or MM pending... 18
of editing session 4
P pending... 18
status area
defining display features 94
during macro processing 104
location on screen 4
stop editing 28
store lines in temporary file for imbed later
in typewriter mode 58
string expression
complex
target as 78
simple
target as 74
string target 74
string, locating
using a target 74
using CLOCATE
in typewriter mode 50
structured input 14
subcommands in XEDIT
defining range of operation 68
entering on display terminal 4
entering on typewriter terminal 47
used in macros, list of 104
with target operands 65
writing your own 103
summary
of initial PF key settings 36, 37
of prefix subcommands 126, 127
of subset for full-screen 35
of subset for typewriter terminals 62
of XEDIT subcommands and macros 121
SUPERSET subcommand 107
symbolic name
assigned 35
synonym
assigning 116
not checking for 107
syntax diagrams, how to read xi

T

tab
changing settings 27

tab *(continued)*
displaying
in typewriter mode 61
example 27
in typewriter mode 60
modifying settings 27
setting
in typewriter mode 61
tab characters
typewriter mode 60
tab key
in typewriter mode 60
using PF key as
example 27
tab line
defining display features 94
displaying 94
example 99
tailor the screen 93
target
as complex string expression
example 81
as line name
example 73
as operand of LOCATE 67
as relative displacement
example 71
as simple string expression
example 78
format 76
as subcommand operand
example 68
definition 65
entered alone 66
entered before subcommand 67
how to express 65
types 69
used in PUT 58
used in subcommands 65
used to change current line 66
used to move line pointer
example 66
used with SET ARBCHAR 78
used with SET CASE 79
used with SET SPAN 80
used with SET VARBLANK 80
temporary file for later imbed
in typewriter mode 58
TOP subcommand
in typewriter mode 50
practice exercise using 40
translating characters 34
Trunc= 2
truncation column 2
TYPE subcommand 49
typewriter
mode 47
typing lines to terminal 49

U

UP subcommand
in typewriter mode 49
practice exercise using 40
UPPERCAS subcommand
example 68

V

vertical screen
multiple 87

W

write
macros 103
your own subcommands 103
write a file on disk
in typewriter mode 56

X

XEDIT command
in typewriter mode 47
practice exercise using 39
used to bypass profile macro 111
used to specify profile macroname 112
XEDIT subcommand
issued from a logical screen 88
issued from a prefix macro 119

Product Number: 5741-A09

Printed in USA

SC24-6338-74